



Department of Computer and Information Sciences

ENHANCING EXTREMIST DATA
CLASSIFICATION THROUGH
TEXTUAL ANALYSIS

KOLADE OLAWANDE OWOEYE

This dissertation is submitted in part fulfilment of requirements for the
degree of MPhil in Computer and Information Sciences.

April, 2023

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for any examination which has led to the Award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom copyrights act as qualified by the University of Strathclyde Regulation 3.50. Therefore, Due acknowledgement must always be made of the use of any material contained in or derived from this thesis.

Acknowledgements

Firstly, I would like to express my appreciation to my dissertation supervisor, Dr. George Weir. His continuous support, unquantifiable assistance and guidance assisted me to complete my MPhil study. Also, I wish to thank Dr. Richard Frank, an Assistant Professor in the School of Criminology at Simon Fraser University (SFU), Canada, who provided me with useful data for my MPhil dissertation.

My heartfelt gratitude goes to my wife, Mrs. Oluwatoyin Owoeye and my Children, Samuel and Gideon for their patience and support during my research study.

I owe unalloyed appreciation to my parents Dr and Mrs Owoeye who always encouraged me throughout the period of writing this thesis and to my brothers, Dr. Olumide Owoeye, Engr. Tope Owoeye, Mr. Oluwafemi Owoeye and my sister, Barrister Titilope Owoeye for their moral support and encouragement towards the successful completion of this MPhil programme.

Finally, I appreciate the Tertiary and Trust Fund (TETFund) for their support in my research study.

Contents

List of Figures.....	xiv
List of Tables.....	xvii
List of Publications.....	xviii
Abstract.....	xx
1. Introduction.....	3
1.1 Problem Statement.....	5
1.2 Aims and Objective of the Study.....	6
1.3 Research Question.....	7
1.4 Overview of Research Method.....	8
1.5 Overview of Research Tools.....	8
1.6.1 Data Sources.....	9
1.6.2 Data Pre-Processing.....	10
1.7 Scope of the Study.....	11
1.8 Limitation of the Study.....	11
1.9 The Contributions of the Research Work.....	12
1.10 Organisation of Thesis.....	12
2. Related Works.....	13
2.1 Text Classification.....	13
2.1.2 Bag of Words and Vector Space Model.....	15
2.1.3 Topic Modelling.....	16
2.1.4 Sentiment Analysis.....	20
2.1.5 Posit-Based Classification Method.....	23
2.1.6 Deep Neural Network Classification Method.....	25
2.2 Imputation Methods.....	25
2.2.1 MICE Imputation.....	27

2.2.2	K-Nearest Neighbours(KNN) Imputation.....	28
2.2.3	MEAN Imputation.....	28
2.2.4	MissForest Imputation.....	29
2.3	Research Gaps	30
3	Methodology.....	31
3.1	Experimental Design.....	32
3.2	Data Sources.....	34
3.2.1	The Dataset’s Complexion.....	36
3.2.2	Attribute of the Dataset.....	40
3.3	Data Preparation.....	42
3.4	Tools.....	43
3.5.1	Feature Extraction Process.....	47
3.5.2	Imputation Methods.....	48
3.6	Posit Experimental Set-Up.....	49
3.6.1	Posit Textual Analysis (Word-Level Feature).....	50
3.6.2	Extended-Posit Analysis.....	52
3.7	The Composite Analysis.....	53
4.	Machine Learning.....	54
4.1	Machine Learning.....	54
4.2	Algorithms.....	57
4.3	Model Metrics.....	57
4.3.1	Validation.....	58
4.3.2	Evaluation Metrics.....	60
4.4	Machine Learning Set-up.....	60
4.4.1	Hyper parameter Turning Set-up.....	64
4.4.2	Feature Selection.....	65

4.5	Classification Result.....	66
4.5.1	Random Forest Classification Result.....	66
4.5.2	Sentiment Based Framework using 27 Features (MF Imputation).....	69
4.5.3	Sentiment Based Framework using 27 Features (KNN Imputation).....	71
4.5.4	Sentiment Based Framework using 27 Features (MICE Imputation).....	74
4.5.5	Feature Selection: Mice Imputation Features.....	76
4.5.6	Posit-Based Classification Framework.....	78
4.5.7	Feature Selection: Posit Features.....	80
4.5.8	Extended-Posit Feature Set (71 features) Classification Framework...	82
4.5.9	Feature Selection: Extended-Posit Features.....	84
4.5.10	Composite Classification Framework.....	86
4.5.11	Feature Selection: Composite Features.....	88
4.6	J48 Decision Tree Classification Result.....	89
4.6.1	Sentiment Based Framework using 27 features (MF Imputation).....	91
4.6.2	Sentiment Based Framework using 27 features (KNN Imputation).....	94
4.6.3	Sentiment Based Framework using 27 features (MICE Imputation).....	96
4.6.4	Feature Selection: Mice Imputation Features.....	98
4.6.5	Extended Posit.....	100
4.6.6	Feature Selection: Extended-Posit.....	102
4.6.7	Posit.....	104
4.6.8	Feature Selection: Posit.....	105
4.6.9	Composite Based Classification Framework.....	108
4.6.10	Feature Selection: Composite Based Classification Framework.....	109

4.7	KNN Classification Results.....	110
4.7.1	MF Imputation.....	113
4.7.2	KNN Imputation.....	116
4.7.3	MICE Imputation.....	119
4.7.4	Extended-Posit Classification.....	122
4.7.5	Posit-Based Classification.....	125
4.7.6	Composite Based Classification.....	128
4.8	The Validation Of Nigerian Extremism Dataset.....	129
4.9	J48 Classification Results.....	129
4.9.1	Posit-Based Classification Results.....	131
4.9.2	Extended-Posit Based Classification.....	133
4.10	Random Forest Classification Results.....	133
4.10.1	Posit Classification Based Results.....	135
4.10.2	Extended Posit-Based Classification Result.....	137
5.	Neural Networks.....	138
5.1	Neural Networks.....	138
5.1.1	Multilayer Perceptron (MLP).....	139
5.1.2	Recurrent Neural Network (RNN).....	142
5.2	Optimisation Algorithm.....	143
5.2.1	ADAM.....	144
5.3	Loss Function.....	145
5.4	Activation Function.....	146
5.5	Regularisation.....	147
5.5.1	Dropout.....	148
5.5.2	Early Stopping.....	149
5.6	Google Colaboratory.....	150
5.7	TensorFlow.....	150
5.8	Validation Metric.....	151

5.9	TensorFlow Implementation.....	153
5.91	The MLP (multi-layer perceptron).....	153
5.92	RNN.....	153
5.93	Early Stopping.....	154
5.10	Classification Results.....	154
5.11	RNN Classification Results.....	155
5.11.1	Sentiment-Based Framework (MF Imputation).....	156
5.11.2	KNN Imputation.....	158
5.11.3	MICE Imputation.....	160
5.11.4	Posit Classification Frameworks.....	162
5.11.5	Extended Posit Classification Framework.....	164
5.11.6	Composite Classification Framework.....	166
5.12	MLP Classification Results.....	166
5.12.1	Sentiment-Based Framework: (MF, KNN and MICE Imputation)...	171
5.12.2	Composite-Based Classification Framework.....	173
5.12.3	Posit-Based Classification Results.....	175
5.12.4	Extended Posit Classification Results.....	176
6.	Result Analysis and Evaluations.....	177
6.1	Sentiment-Based Classification Framework.....	178
6.2	Composite-Based Classification Framework.....	179
6.3	Feature Selection vs Model Optimization.....	181
6.4	Machine Learning and Neural Network Algorithms.....	181
6.4.1	Overall Classification Result.....	182
6.5	Validation of Nigerian Extremism Webpages.....	183
6.6	Results Comparison With the Literature.....	184
6.7	Human-Verification of Manually Labelled Data.....	195
7.	Conclusions and Future Work.....	186

7.1	Conclusions.....	186
7.1.2	The Contributions from the Thesis.....	189
7.2	Future Work.....	189

List of Figures

1.1	The Proposed Composite Framework.....	8
2.0	MICE Imputation Framework Source.....	27
3.1	The Architecture of the Research Methods.....	32
3.2	Total words for the Extremism (ICCRC) Data.....	36
3.3	Number of Sentences for Extremism (ICCRC) Data.....	37
3.4	Average Sentence Length for the Extremism (ICCRC) Data.....	38
3.5	Average Word Length for the Extremism (ICCRC) Data.....	38
3.6	Type/Token Ratio for the Extremism (ICCRC) Data.....	39
3.7	Sentiment Experimental Set-up.....	42
3.8	Sentiment Feature Extraction Process.....	44
3.9	Data Frame of the Dataset.....	46
3.10	Word-level Feature Extraction Process using Posit Analysis.....	50
3.11	Processed Features using Posit.....	51
3.12	Composite Feature Extraction Process.....	53
4.1	Validation Curve-MF Imputation.....	68
4.2	Sentiment Based Framework using 27 features (KNN Imputation)....	69
4.3	Validation Curve-KNN Imputation.....	70
4.4	Sentiment Based Framework using 27 features (Mice Imputation)....	71
4.5	Validation Curve-MICE Imputation.....	74
4.6	Mice Imputation Confusion Matrix.....	74
4.7	The Wrapper Method in MICE Imputation using Random Forest.....	75
4.8	The Embedded-Method in MICE Imputation using Random Forest...	76
4.9	Validation Curve- Posit.....	77
4.10	Posit Confusion Matrix.....	78
4.11	The Wrapper Method in Posit using Random Forest.....	79

4.12	The Embedded Method in Posit using Random Forest.....	80
4.13	Validation Curve-Extended Posit.....	81
4.14	Extended Posit Confusion Matrix.....	82
4.15	The Wrapper Method in Extended-Posit using Random Forest.....	83
4.16	The Embedded Method in Extended-Posit using Random Forest.....	84
4.17	Validation Curve-Composite.....	85
4.18	Composite Confusion Matrix.....	86
4.19	The Wrapper Method in Composite using Random Forest.....	87
4.20	The Embedded Method in Composite using Random Forest.....	88
4.21	Validation Curve-MF Imputation.....	90
4.22	MF Imputation Confusion Matrix.....	91
4.23	Validation Curve-KNN Imputation.....	93
4.24	KNN Imputation Confusion Matrix.....	94
4.25	Validation Curve-Mice Imputation.....	95
4.26	Mice Imputation Confusion Matrix.....	96
4.27	Wrapper Feature Selection Method-Mice Imputation.....	97
4.28	Embedded Method Feature Selection Mice Imputation.....	98
4.29	Validation Curve-Extended Posit Imputation.....	99
4.30	Extended Posit Imputation Confusion Matrix.....	100
4.31	Wrapper Method-Extended Posit.....	101
4.32	Embedded Method for Extended-Posit.....	102
4.33	Validation Curve-Posit Imputation.....	103
4.34	Posit Imputation Confusion Matrix.....	104
4.35	Wrapper Method Feature Selection-Posit.....	105
4.36	Embedded Method Feature Selection Posit.....	105
4.37	Validation Curve Posit Mice Imputation.....	107
4.38	Composite-Based Confusion Matrix.....	108
4.39	Wrapper Method for Composite.....	109
4.40	Embedded Method for Composite.....	109

4.41	Validation Curve-MF Imputation.....	112
4.42	MF Imputation Confusion Matrix.....	113
4.43	Validation Curve-KNN Imputation.....	115
4.44	KNN Imputation Confusion Matrix.....	116
4.45	Validation Curve-Mice Imputation.....	118
4.46	Mice Imputation Confusion Matrix.....	119
4.47	Validation Curve-Extended Posit.....	121
4.48	Extended Posit Confusion Matrix.....	122
4.49	Validation Curve-Posit.....	124
4.50	Posit Confusion Matrix.....	125
4.51	Validation Curve-Composite.....	127
4.52	Composite Confusion Matrix.....	128
4.53	Validation Curve-Posit.....	130
4.54	Posit Confusion Matrix.....	131
4.55	Validation Curve-Extended Posit.....	132
4.56	Extended-Posit Confusion Matrix.....	133
4.57	Validation Curve-Posit.....	134
4.58	Posit Confusion Matrix.....	135
4.59	Validation Curve-Extended Posit.....	136
4.60	Extended-Posit Confusion Matrix.....	137
5.1	A Neural Network with 2 Hidden Layers.....	139
5.2	Recurrent Neural Network and a Feed-Forward Neural Network.....	141
5.3	Different Types of RNN.....	142
5.4	An illustration of a RNN with its three gates.....	142
5.5	Gradient Descent Algorithm.....	143
5.6	Structure of different training sets.....	147
5.7	Normal Neural Network before and after applying Dropout.....	148
5.8	Training Set Accuracy Source.....	149
5.9	Model Accuracy Curve-MF Imputation Curve-MF Imputation.....	155

5.10	Model loss Curve-MF Imputation.....	155
5.11	MF Imputation Confusion Matrix.....	155
5.12	Model accuracy Curve-KNN Imputation Curve-KNN Imputation....	157
5.13	Model loss CURVE-KNN Imputation.....	157
5.14	KNN Imputation Confusion Matrix.....	158
5.15	Model loss Curve-MICE Imputation.....	159
5.16	Model accuracy Curve-MICE Imputation.....	159
5.17	MICE Imputation Confusion Matrix.....	160
5.18	Model Accuracy Curve-Posit Imputation.....	161
5.19	Model loss Curve-Posit Imputation.....	161
5.20	Posit Imputation Confusion Matrix.....	162
5.21	Model Accuracy Curve-Extended Posit.....	163
5.22	Model loss Curve-Extended Posit.....	163
5.23	Extended Posit Imputation Confusion Matrix.....	164
5.24	Model Accuracy Curve-Composite.....	165
5.25	Model loss Curve-Composite.....	165
5.26	Composite-Based (Posit MICE) Confusion Matrix.....	166
5.27	Model Accuracy Curve-MF Imputation.....	167
5.28	Model loss Curve-MF Imputation.....	167
5.29	Model Accuracy Curve-KNN Imputation.....	168
5.30	Model loss Curve-KNN Imputation.....	168
5.31	Model Accuracy Curve-Mice Imputation.....	168
5.32	Model loss Curve-MICE Imputation.....	168
5.33	MF Imputation Confusion Matrix.....	170
5.34	KNN Imputation Confusion Matrix.....	170
5.35	Mice Imputation Confusion Matrix.....	171
5.36	Model Accuracy Curve-Composite.....	172
5.37	Model Loss Curve-Composite.....	172
5.38	Composite Confusion Matrix.....	173

5.39	Model Accuracy Curve-Posit.....	174
5.40	Model loss Curve-Posit.....	174
5.41	Posit Confusion Matrix.....	175
5.42	Extended Posit Confusion Matrix.....	176

List of Tables

3.1	Complexion Analysis of Extremist (ICCRC) Dataset.....	39
3.2	Noun Keyword List.....	44
4.1	Confusion Matrix 2x2.....	59
4.2	The Model Parameters.....	62
4.3	The Random Forest Model Parameters.....	66
4.4	Gridsearch Best Parameters for MF Imputation.....	67
4.5	MF Classification Result using Random Forest.....	68
4.6	Gridsearch Best Parameters for KNN Imputation.....	69
4.7	KNN Classification Result using Random Forest.....	71
4.8	Gridsearch Best Parameters for MICE Imputation.....	72
4.9	Mice Imputation Classification Result using Random Forest.....	73
4.10	Gridsearch Best Parameters for Posit Data.....	76
4.11	Posit Classification Result using Random Forest.....	78
4.12	Gridsearch Best Parameters For Extended-Posit Data.....	80
4.13	Extended-Posit Classification Result using Random Forest.....	82
4.14	Gridsearch Best Parameters for Composite Data.....	84
4.15	Composite-Based Classification Result using Random Forest.....	86
4.16	The J48 Model Parameters.....	89
4.17	Gridsearch Best Parameters for MF Imputation.....	89
4.18	MF Imputation Classification Result.....	91
4.19	Gridsearch Best Parameters for KNN Imputation.....	92
4.20	KNN Imputation Classification Result.....	93
4.21	Gridsearch Best Parameters for Mice Imputation.....	94
4.22	Mice Imputation Classification Result.....	96

4.23	Gridsearch Best Parameters for Extended Posit.....	98
4.24	Extended Posit Imputation Classification Result.....	100
4.25	Gridsearch Best Parameters for Posit.....	102
4.26	Posit-Based Classification Result.....	104
4.27	Gridsearch Best Parameters for Composite.....	106
4.28	Composite-Based Classification Result.....	107
4.29	The KNN Model Parameters.....	110
4.30	Gridsearch Best Parameters for MF Imputation.....	110
4.31	MF Imputation Classification Result using KNN.....	113
4.32	Gridsearch Best Parameters for KNN Imputation.....	114
4.33	KNN Imputation Classification Result using KNN.....	116
4.34	Gridsearch Best Parameters for MICE Imputation.....	117
4.35	Mice Imputation Classification Result using KNN.....	119
4.36	Gridsearch Best Parameters for Extended-Posit.....	120
4.37	Extended Posit Classification Result using KNN.....	122
4.38	Gridsearch Best Parameters for Posit.....	123
4.39	Posit Classification Result using KNN.....	125
4.40	Gridsearch Best Parameters for Composite.....	126
4.41	Composite-Based Classification Result using KNN.....	128
4.42	The Model Parameters.....	129
4.43	Posit-Based Classification Result using J48.....	130
4.44	Extended Posit-Based Classification Result using J48.....	132
4.45	Posit-Based Classification Result using Random Forest.....	134
4.46	Extended Posit-Based Classification Result using J48.....	136
5.1	MF Imputation Classification Result using RNN.....	155
5.2	KNN Imputation Classification Result using RNN.....	157
5.3	MICE Imputation Classification Result using RNN.....	159
5.4	Posit Classification Result using RNN.....	161
5.5	Extended Posit Imputation Classification Result using RNN.....	163

5.6	Composite-Based Classification (Posit-Mice) Result using RNN.....	165
5.7	MF Imputation Classification Result using MLP.....	169
5.8	KNN Imputation Classification Result using MLP.....	169
5.9	Mice Imputation Classification Result using MLP.....	170
5.10	The Composite Classification Result using MLP.....	172
5.11	Posit Classification Result.....	174
5.12	Extended-Posit Classification Result.....	176
6.1	Comparison of the Classification Methods.....	179
6.2	Result Comparison between Wrapper and GridsearchCV Methods using J48.....	180
6.3	Result Comparison between Wrapper and GridsearchCV Methods using Random Forest.....	180
6.4	Results comparison with the literature.....	184

List of Publications

- i. G. Weir, K. Owoeye, A. Oberacker and H. Alshahrani, "Cloud-based Textual Analysis as a Basis for Document Classification," *2018 IEEE International Conference on High Performance Computing & Simulation (HPCS)*, Orleans, 2018, pp. 672-676, doi: 10.1109/HPCS.2018.00110.
- ii. K. O. Owoeye and G. R. S. Weir, "Classification of Radical Web Text Using a Composite-Based Method," *2018 IEEE International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2018, pp. 53-58, doi: 10.1109/CSCI46756.2018.00018.
- iii. K. O. Owoeye and G. R. S. Weir, "Classification of Extremist Text on the Web using Sentiment Analysis Approach," *2019 IEEE International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2019, pp. 1570-1575, doi: 10.1109/CSCI49370.2019.00302.

Abstract

The high volume of extremist materials on the Internet has created the need for intelligence gathering via the Web and real-time monitoring of potential websites for evidence of extremist activities. However, the manual classification for such contents is practically difficult and time-consuming. In response to this challenge, the work reported here developed several classification frameworks. Each framework provides a basis of text representation before being fed into machine learning algorithm. The basis of text representation are Sentiment-rule, Posit-textual analysis with word-level features, and an extension of Posit analysis, known as Extended-Posit, which adopts character-level as well as word-level data. Identifying some gaps in the aforementioned techniques created avenues for further improvements, most especially in handling larger datasets with better classification accuracy.

Consequently, a novel basis of text representation known as the Composite-based method was developed. This is a computational framework that explores the combination of both sentiment and syntactic features of textual contents of a Web page. Subsequently, these techniques are applied on a dataset that had been subjected to a manual classification process, thereafter fed into machine learning algorithm. This is to generate a measure of how well each page can be classified into their appropriate classes. The classifiers considered are both Neural Network (RNN and MLP) and Machine Learning classifiers (such as J48, Random Forest and KNN). In addition, features selection and model optimisation were evaluated to know the cost when creating machine learning model.

However, considering all the result obtained from each of the framework, the results indicated that composite features are preferable to solely syntactic or sentiment features which offer improved classification accuracy when used with machine learning algorithms. Furthermore, the extension of Posit analysis to include both word and character-level data out-performed word-level feature alone when applied on the

assembled textual data. Moreover, Random Forest classifier outperformed other classifiers explored. Taking cost into account, feature selection improves classification accuracy and save time better than hyperparameter turning (model optimisation).

Chapter One: Introduction

1. Introduction

Radicalisation is used often in relation to Jihadism or Islamic extremism [1]. Other concepts of radicalisation are also seen as a component of Fascism or white supremacy [2]. Radicalisation is defined as a desire to rule, of which social movements and its actions serve as the vessels to achieving the power [3]. To this effect, radicals form a uniform society founded on strong, opinionated belief and doctrine. They attempt to create society conformists by suppressing all opposition [3]. In 1985, the earliest piece of investigation on violent radicalism and the Internet appeared, but the vast majority of investigations began after the year 2000 when digital methods of radicalisation became more sophisticated [4].

The spread of extremist documents on the Internet is alarming and has become a major concern for government and security agencies. Terrorist and extremist groups adopt digital method such as Web technologies for various functions including dissemination of information, propaganda, fundraising, recruitment and assignment of deadly missions [5-10]. The potential dangers of online extremism cannot be over-emphasised. For example, three thousand people were killed in the 9/11 terrorist attacks in the United States [7] while four people were killed and many injured in an extremist attack at Westminster, London [8] to mention a few. However, a survey from the National Consortium for the Study of Terrorism and Responses to Terrorism (START) [9], also reported 2,794 terrorist attacks from 1970 to 2016 in the United States that resulted in 3,659 deaths. The Global Terrorism Index, GTI [9] reported that Boko-Haram in Nigeria was one of the world's deadliest extremist groups in 2014 with a record of 6,700 deaths. Just a single terrorist attack in Nigeria was recorded among the 20 most deadly terrorist attacks worldwide in 2016. In 2014, nine similar attacks happened in that same country, Nigeria.

Examples of extremist Websites are jihadi Websites, far-right propaganda and bomb-making instructional Websites. Many law enforcement and intelligence agencies are interested in countering the use of the Internet for extremism due to the rapid increase of extremist documents online, this situation has created the need for efficient automated systems for the classification and identification of Web pages with extremism contents. An automated method to classify or identify such radical documents on the internet is one of the counter-terrorism measures against internet cyber threats of which text document is the most common content type on the Web. However, manual classification of such content on the Internet is impractical due to the existence of billions of Web pages of diverse uses.

The number of different text feature representation (a basis of text feature) methods to be considered in this research for building a classification model are sentiment analysis, Posit (word-level information), and Posit (both word and character-level information). In an attempt to further improve the textual content classification, a computational framework known as the composite-based classification method is proposed which is based on the combination of a machine learning algorithm, and the hybrid of both sentiment and syntactic features of the Web texts, to build a model for the automatic classification of extremism Web pages. A mix of sentiment and syntactic features derived from the textual data is regarded as composite features, a basis for text features representation. The rationale behind the hybrid features in the composite approach is to explore the richer feature set that feeds into building a classification model. Sentiment analysis generates sentiment features in unstructured data while Posit provides the quantitative syntactic features that ‘enrich’ the information given by the text corpus.

The effectiveness of the classification frameworks would be analysed and tested on two different text corpora. The first dataset is created with Web crawlers at the International Cyber Crime Research Centre (ICCRC) at Simon Fraser University in Burnaby, British Columbia, Canada, while the second corpus is data retrieved from Nigerian Websites.

After collection and pre-processing, each text corpus contains three manual classes that cover the themes "pro-extremist," "neutral" or "anti-extremist" based on the contents of the data. For example, pro-extremist expresses extremism contents from "extremist and jihad organisation Websites". The neutral group reflects contents from the media/news that impartially report terrorist events. The anti-extremist class contains items that express views against terrorism. The Web data retrieved from Nigerian domain sources would be used to test the validity of a trained dataset obtained from ICCRC. The assumption is that, since the Nigerian data is potentially similar to the ICCRC data, in the sense that we are interested in the same classification categories, if it performs well, this could be taken as validation for the approach since it would seem to work well across differently sourced data sets (for the same classification tasks).

The objective of the thesis is to develop a robust automatic Web-content classification model. Therefore, the manual classification of the Terrorism and Extremism Network Extractor (TENE)-sourced Web pages (a Canadian ICCRC data) serves as a threshold to measure the success of our automated method. The thesis explores different classification algorithms namely, neural networks (Multilayer Perceptron (MLP), Recurrent Neural Network (RNN)) and machine learning algorithms (such as, J48, Random Forest (RF) and K-Nearest Neighbours (KNN)) which are implemented using both TensorFlow and Sckit-learn API respectively.

1.1 Problem Statement

The adoption of an internet presence on such platforms as YouTube, Facebook, Twitter and other online forums gave extremist groups like Boko-Haram and ISIS the opportunity to dramatically increase their membership. Extremist's activities include dissemination of information, propaganda, fundraising, recruitment and assignment of deadly missions. In such contexts, the Internet poses a threat to national security. One

form of counter-terrorism measure is the classification of such extremist documents (Web pages) on the Internet.

The manual classification for such radical documents on the Web is practically difficult and time-consuming due to the existence of billions of Web pages of diverse uses. In response to this challenge, an automated classification system is needed for such a task. However, building such a classification model requires transforming text of unstructured data into a form that can be utilised by various machine learning algorithms, which is a key challenge in enhancing data classification through textual analysis. Traditional methods such as bag of words and vector space models [11-12] have been proposed for the text document representations used in a classification model. In the vector space model, text is extracted into word sequence and the weight for the features is computed as a weight vector, then a classifier is developed based upon the weight vector space. However, text has many features and the dimensionality of the vector space can be very high, which leads to time and space complexity in the classification model process. However, a method to extract fewer but more useful features is crucial to building efficient classification systems.

In recent times, methods such as Sentiment [5] and Posit analysis, both at word-level [14] and character-level [15], to mention a few, have been widely used to characterise a set of text for use in a classification model. Sentiment analysis method such as [5], [13-14] relies on the use of keywords (frequently used keywords) to obtain sentiment in each Webpage and reducing the number of keywords to top k-nouns (which often carry the sentiment) poses a challenge as the chances are certain that some Webpages in each class have few or none of the selected keywords. This situation results in the non-capture of some sentiment values from larger Web page data, thereby hindering the training process of useful sentiment features of the Web pages. Such data incompleteness is regarded as missing data which can impair the classification accuracy when machine learning algorithms are applied.

Posit, on the other hand, is designed to generate quantitative data at the level of word and part-of-speech content of texts. It creates data based on word-level information which could be a disadvantage when applied to short data such as the content of tweets, as many of the original features may result in zero values.

To address the challenges mentioned above, this thesis proposes different imputation approaches to address the data incompleteness (missing data) faced by the sentiment analysis method (the sentiment analysis approach that utilises top-k noun keywords to obtain sentiment from text corpus before being fed into a machine learning). Moreover, a novel framework is also proposed to improve the textual content classification method further, the proposed framework is known as the composite-based classification method. The rationale behind the hybrid features in the composite approach is to explore the richer feature set that feeds into building a classification model.

In addition to the previously observed limitation in using Posit analysis on short text (such as twitter text), the system has been upgraded to complement the conventional word-level statistics (27 default word-level features) with an extra 44 character features for each instance of text data [5]. The new addition include quantitative information on individual alphanumeric characters as well as a subset of special characters such as question marks, exclamation marks, asterisks, periods, dollar signs, etc. Consequently, each data item is represented by a set of 72 features [5]. However, Posit with the different data level information will be applied to non-short text data. This is to establish both methods' effectiveness and improvement in building a textual classification model. Hence, keyword modelling in sentiment-rule based analysis and Posit-textual based classification models will be revisited in this thesis. The classification models to be considered are both neural network models and traditional machine learning algorithms. A GridSearchCV algorithm will be explored for hyperparameter tuning to obtain the optimal values for each of the machine learning models. However, the methods to be examined in this thesis raise some questions when classifying extremist Web data, these questions are presented in section 1.3.

1.2 Aims and Objectives of the Study

The thesis aims to develop a Web-content classification model. The specific research objectives are to:

- i. Develop an enhanced textual content classification method, a composite-based classification method.
- ii. Evaluate different imputation methods (such as KNN, MICE and MissForest) applied to compensate for missing values on sentiment-based feature set.
- iii. Develop Posit (word-level) and Posit (word and character-level)-based classification methods.
- iv. Evaluate the performance of composite-based classification with existing methods namely, Sentiment-Rule, Posit-based classification and Extended Posit.

1.3 Research Question

This section presents the research questions that are focused on the frameworks to be considered in this thesis which include:

- i. Can the imputation method efficiently compensate for missing values faced by feature set obtained via sentiment analysis (a procedure that utilises top-k noun keywords to obtain sentiment values from text corpus) before being fed into machine learning for the classification task?
- ii. Can the composite approach (the combination of sentiment and syntactic features in textual content as a basis for text features) be effective to create a well-working machine learning model?
- iii. What is the cost of model optimisation (hyperparameter turning) over feature selection when creating a machine learning model?

- iv. Considering the selected machine learning and neural network algorithms (such as RNN, MLP, KNN, J48 and Random Forest) on a pre-processed feature, which model produces the best classification accuracy on extremist Web textual data?
- v. Can a model based on the dataset used for these experiments be validated on another dataset of a similar domain but a different source?

1.4 Overview of Research Method

A different experiment is performed to test each research question. The experiments consist of different classification frameworks developed for the classification of extremist Web content. Each framework provides a basis of text representation before being fed into a machine learning algorithm. For example, the sentiment features of the Web pages will be generated through the sentiment analysis, where linguistic markers (top-k noun keywords) are used to pinpoint the sentiment of each Web page. Then, a lexical approach, a Sentistrength [16] resource, would then assign a sentiment value to each of the Web pages. Thereafter, we propose different imputation approaches to address the data incompleteness (missing data) faced by sentiment analysis (the method that relies on the use of top-k noun keywords to obtain sentiment around each Web page). The imputation approach maintains all situations by substituting an approximated value based on other available data for missing data [17]. The feature set can then be analysed using standard procedures for comprehensive data analysis once all missing values have been imputed. Syntactic features of textual contents of a Web page would be obtained through a textual analytic tool called Posit. Posit is a Unix-Scripting program that is capable of generating frequency data, as well as Part-of-Speech (POS) tagging in unstructured textual data. The composite features explore the combination of both sentiment and syntactic features of textual content of a Web page as a basis for document classification. The composite framework is illustrated in Figure 1.1.

Various machine learning algorithms that will be applied to each feature are neural network models (RNN and MLP) and traditional machine learning algorithms such as (J48, Random Forest and KNN), this is to generate a measure of how well each page can be classified into their appropriate classes. Then, the feature selection algorithms, both wrapper method type (Recursive feature elimination (RFE) and the embedded method will be applied to the various feature sets. Applying these feature selection algorithms to the classification models will allow us to explore the full range of effectiveness and the cost on the feature subsets performances (feature optimisation).

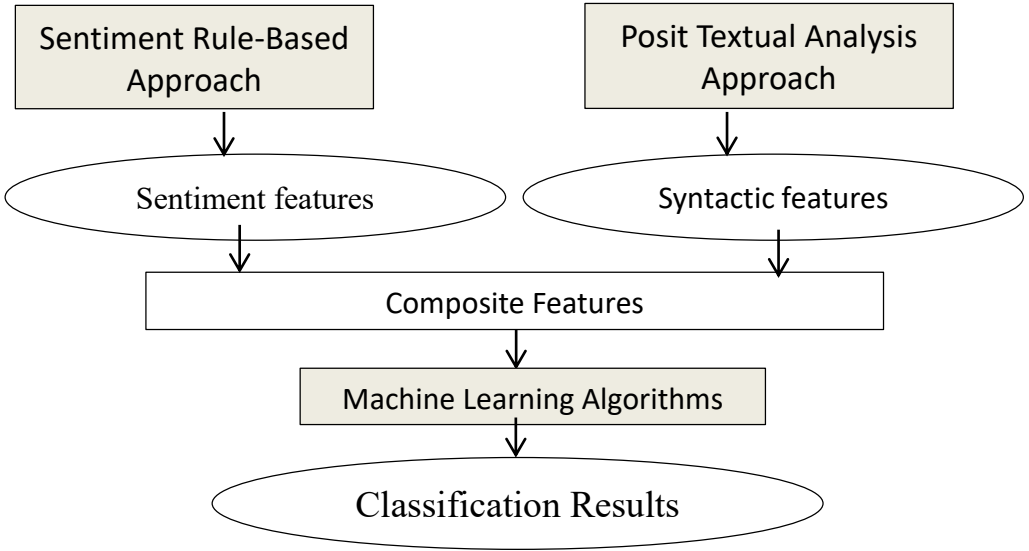


Figure 1.1: The Proposed Composite Framework

1.5 Overview of Research Tools

Tools explored for the study are Google Colab GPU, and Python libraries (Pandas, Scikit-learn, TensorFlow) are used to implement the experiments and to output the results into files. The details of the tools will be re-visited later in this thesis.

1.6.1 Data Sources

In this section, we describe the two different data sources that would be explored to test the effectiveness of the methods.

i. Extremism Dataset (ICCRC)

The Web pages that comprise the initial extremism dataset were obtained from extremist and associated Websites using the TENE-WebCrawler which is a software developed at the International Cyber Crime Research Centre (ICCRC), Simon Fraser University, Canada. This crawler traverses the Internet following links based on keyword searches, extracting Web pages and analysing each page visited [5]. One collection of such Web pages was initially classified manually by ICCRC workers, with each Web page classified as "pro-extremist," "neutral," or "anti-extremist" based on its content. The total number of manually classified Web pages was 7500, with 2500 Webpages in each category.

ii. Nigerian Extremism Dataset

A second extremism dataset was compiled from the content from Nigerian Websites. Websites with extremism topics were retrieved from Nigerian Websites with the aid of Beautiful Soup framework [18]. The compiled Nigerian dataset consisted of 70 text documents from different websites, classified manually based on their content, the Webpages were manually categorised into the three classes, pro-extremist, anti-extremist and neutral. The result was 70 Web text documents in each class. These data were later classified manually with the aid of a qualitative research tool, NVivo [19]. The details of the data sources are further explained in the later part of the thesis.

1.6.2 Data Pre-Processing

Raw data that contains noise is unclean, and thus degrades the classification result's quality. Pre-processing, on the other hand, aids in the processing of noisy data and improves the effectiveness of machine learning algorithms. Before being input into

machine learning, the dataset is pre-processed to improve classification accuracy. The pre-processing steps include noise cleansing, dealing with missing values and scaling of the data. The detailed process will be re-visited later in this thesis.

1.7 Scope of the Study

This research study focused on the classification of online radicalisation text contents (extremist Webpages). The research centred on analysing extremist content to gain deeper insight into which class each content of the Website belong, either neutral, anti-extremist or pro-extremist. The neutral content reports routinely on terrorist events from what might arguably be a more impartial and journalistic perspective. The anti-extremist content reveals opposition to violence while pro-extremist content expresses extremist contents. Input to the analysis algorithms comprises documents containing varying degrees of radical and related content and the output is an analysis that details sentiment and syntactic characteristics of the content.

On the basis of such analyses, machine learning algorithms were used for the text classification. The technique follows a certain trend namely, manual labelling of documents into categories, document representation [5, 14 and 15], training a classifier on seen data and evaluation on an unseen test set [20]. Various text document representations considered in the study include syntactic, sentiment and composite-based features. The classifiers explored for the study include, RNN, MLP, J48, Random Forest and KNN. In order to assess the performance of this classification, the metrics considered were Precision, Recall, F-measure and accuracy. These aspects will be detailed in Chapter 2. Many terrorist organisations have created Websites on the Internet for various purposes such as fundraising, propaganda and recruitment. Hence, Websites considered for this study include Weblogs (also commonly known as blogs), and online forums from Websites. This study considers textual content as the data type, which explores the structure and content of a document such as sentiment and syntactic features. Various types of features used to assist techniques to categorise online

radicalisation text on the Web are the link based and content-based features commonly used features in the literature [5][13][14].

However, the study focused on content-based features because it often used in text classification techniques. The content-based features explore the structure and content of a document such as lexical (frequency of letters, average word length etc.), syntactic (frequency of function words etc.). Extremists use different languages associated with radicalisation. However, in this study, the language of all considered documents is English text. Of course, many genres of extremism or radicalisation exist on the Internet, including Middle Eastern extremism (a pro-caliphate Islamic political party and the Website of the Muslim Brotherhood), US domestic extremism and Anti-Semitism (of public safety agencies, or groups like the Global Counterterrorism Forum). We have covered these varieties of extremism in our study.

1.8 Limitation of the Study

Much of the data on terrorist organisations is For Official Use Only or Law Enforcement Only. Hence, extraction of the Web contents from open Web data through Web-Crawler software was an option explored as a source of data used. The research is based on Web content classification.

1.9 The Contributions of the Research Work

The thesis compares and contrasts various types of imputation approaches used to account for missing data in sentiment analysis (the approach that relies on the use of top-k noun keywords to obtain sentiment around each Web page). According to the findings of the study, composite features are preferable to just syntactic or sentiment features in terms of classification accuracy when utilized with machine learning algorithms. Furthermore, the extension of Posit analysis to include both word and character-level data out-performed word-level feature alone when applied on the assembled textual data. Moreover, the Random Forest classifier outperformed the other classifiers that were

tested. Feature selection increases classification accuracy and saves time better than hyperparameter turning.

1.10 Organisation of Thesis

The remainder of the thesis is organised as follows. Chapter Two presents a review of existing literature on text classification methods. Chapter three contains a description of the data and the methods adopted in the thesis. Chapter Four describes the implementation and the results for the machine learning models, Chapter Five describes the implementation and the results for the Neural Network models, and Chapter Six describes the analysis and evaluations of the results. Conclusions and future work are presented in Chapter Seven.

Chapter Two: Related Work

This chapter discusses existing concepts of the field of textual classification. Furthermore, related research is going to be examined to place this thesis into context. The target of this thesis is to analyse texts in the context of a predefined set of topics, the strengths and weaknesses of related research would be examined to fill the research gap.

2.1 Text Classification

Text classification is the act of labelling documents into categories with respect to their content. The process can be manual or automated, and used to easily sort and manage texts, images or videos. Lists of the textual classification methods described in this section include Topic Modelling, Sentiment, Posit, Bag of Word and Vector Space classification-based models.

2.1.2 Bag of Words and Vector Space Model

This section reviews existing studies on Bag of Words and Vector Space Model used for Web contents classification.

Both Bag-of-Words and Vector Space models are types of representations of text features used in Information Retrieval and Natural Language Processing. The Bag-of-Words describes the instances of words within a document and a sentence is represented as a bag of words vector (a string of numbers) [21]. In the Vector Space model, the text is extracted into word sequence and the weight for the features is computed as a weight vector, then a classifier is developed based upon the weight vector space [23]. Existing research work on both Bag of Words and Vector Space models is discussed below.

A technique for text analysis that explores the Bag of Words model for a predictive analysis was proposed [21]. The research was conducted with a set of unstructured data which were pulled together from the domain of natural language processing (NLP), in a bid to gain a wide scope of attraction from various researchers and on-field practitioners to create the right impression of forecasting and predicting insights in a simple and explanation fashion. The research detailed the operations of the framework for the fast usage of Bag of Words model for text mining processing. Short text multi-class classification problems in the Bags of Words model were addressed using word vector enrichment of flow frequency words [22]. The research was keen to explore three different aspects of the problem about the classification of three different domains which happen to be Reuter news article classification, classification of journal article titles, and text snippets classification. The research employed bag of words model in underpinning the variables from both the general and specific domains. The outcome of their study showed that a mix of the information in the unsupervised word vector model with a supervised linear model enhances classification performance when compared with other classifiers to address other text classification problems. The technique explored in the research was effective because it requests no change to the linear classifier throughout the training, the technique only applies to the text being classified.

The analysis of a vector space model for data classification on the Internet of a thing (IoT) was discussed [23]. The objective of the research was to give a perception of how the accessibility of information could trigger an increase in the IoT. The research brought into the limelight the proposal of a new text classification algorithm design through the aid of the vector space model. The algorithm developed triggers a rise in feature selection and weighting method through the approach of synonym replacements made to the traditional text classification algorithms. The result obtained from the experiment showed a better performance when compared with existing algorithms. The vector Space model used to classify Arabic text was discussed [24]. The research applied KNN algorithms to explore the different variations of vector space models (VSMs), with the keen intention of creating new segments of the Arabic text classifier to

segment the Arabic text. The results obtained from the research showed that the cosine categories' performed better than the dice and Jaccard algorithms.

2.1.3 Topic Modelling

This section discusses existing methods used in a topic model for classifying extremist Web content.

A topic model is a form of statistical model for detecting the conceptual "topics" that appear in a group of documents. It is also a frequently used text-mining tool to unravel hidden semantic structures in a text document. Topic modelling and critical discourse analysis were combined to obtain the patterns of the representation around the keyword terms Islam and Muslim in a word corpus of a sizeable Swedish Internet forum ranging from the year 2000 to 2013 [25]. The corpus used in the course of the research was derived from the flashback, one of the biggest Web forums in the world. The outcome of the study indicates that Muslims are observed as a homogenous forum that can be attributable to conflicts and violent acts.

A new Seed-guided Multi-label Topic Model (SMTM) was proposed [26]. SMTM performs multi-label classification efficiently for a group of documents without any labelled document with just a few words relevant to each class of the document. In the proposed method, a single category topic is attached to each class of document which gives the meaning of the class. However, in a process of operating with multi-label documents, the research distinctly models the class sparsity in the method by exploring the techniques, spike and slab prior and weak smoothing prior. SMTM automatically chooses the appropriate class for each document without using any threshold tuning. In addition, a seed-guided biased GPU sampling method to monitor the topic inference of SMTM was also developed for the supervision of the seed words. The effectiveness of the model on the two public datasets showed that the proposed method achieved good classification results.

A topic modelling algorithm such as Latent Dirichlet Allocation (LDA) has been used largely to design documents as a collection of topics. LDA is a generative probabilistic model for the gathering of discrete data [27]. Some studies have shown how it has been used to analyse the same Web contents to detect important topics in extremist online forums. For example, a study in [28] explored the Latent Dirichlet allocation (LDA) algorithm. The authors developed a framework to detect latent topics by analysing the contents of dark websites. A Web-crawler was explored to extract the Dark Web contents used for the analysis. Then, the Latent Dirichlet allocation (LDA) algorithm was applied to analyse the Web content to reveal latent topics from Websites of terrorists or extremists. The result of the experiment showed that LDA-based analysis allocates a probability to a document and covers the exchangeability of both words and documents.

A new low-dimensional text representation approach for topic classification was developed [29]. The model was developed based on the multi-level LDA representation. A Latent Dirichet Allocation (LDA) model was explored to extract possible topic clusters in the dataset. The effectiveness of the model was implemented on two datasets. The first dataset was obtained from the FriendFeed social network, manually interpreted with ten classes, while the second was an ideal text classification benchmark, Reuters 21578, the R8 subset (interpreted with eight classes). Eventually, the result from the proposed classification model gave improved results for both datasets.

2.1.4 Sentiment Analysis

The proposed method presented in this thesis is an underpinning of sentiment analysis that uses keywords as a linguistic marker technique to pinpoint sentiment in a Web page. This section details existing methods on sentiment-based classification method.

Sentiment analysis tends to determine opinion or emotion in unstructured textual data. Methods used in sentiment analysis include machine learning and semantic orientation.

Sentiment analysis uses a computational approach to obtain opinionated contents and classifies the overall review of the topic into positive, negative and neutral. It also reveals users' intentions, emotions and opinion hidden in the unstructured text [30]. Sentiment analysis uses a computational approach to obtain opinionated content and classifies the overall review of the topic into positive, negative and neutral. The techniques used by sentiment analysis for classification include machine learning and lexicon-based approaches. The studies on sentimental analysis of public opinions as expressed on social media in Ghana as regards government policies and decisions using machine learning algorithms were carried out [31]. The research used the Naïve Bayes, Support vector machine and random forest algorithms for analysis. It was discovered that the Naïve Bayes classifier was adjudged the best with an accuracy of 99%. The use of sentiment analysis as a tool of supervised machine learning algorithm to classify Lithuanian news website contents, especially those that pertain to financial issues was examined [32]. The results revealed that the non-balanced dataset produced the highest accuracy through the Naïve Bayes algorithm with the support vector machine coming behind at a lower level of accuracy.

The sentiment analysis of social media texts using machine learning techniques such as the Support Vector Machine (SVM), Naïve Bayes (NB) and the Artificial Neural Networks (ANN) techniques were examined. The study revealed the ANN technique had the best classification accuracy to the tune of 90% [33].

The work explored millions of tweets from more than 25,000 common users that were manually tagged, reported and suspended as a result of their involvement with extremist movements by Twitter and another sample of tweets was obtained randomly from 25, 000 common users who are open to extremist content. All the information was used for the forecasting tasks. Eventually, the performance of the framework revealed a 93% success rate for extremist user detection and an 80% rate for predicting content adopters. Another method used in sentiment analysis for the classification of a text document is semantic orientation. This operates by depending on a method that utilised a corpus annotated for

sentiment or a sentiment value derived from a dictionary of words in classifying text documents [34]. Many studies have explored a hybrid of both data mining algorithm and semantic orientation (a lexical approach) in classifying or identifying extremism on Web pages, such as [5] [13] [35].

An authorship analysis framework was implemented on the linguistic features extracted from online messages in [35]. The result was evaluated to determine the stylistic features of terrorist communications. A multilingual model comprising a set of algorithms and related features was used to detect Arabic messages and their language's unique peculiarities on an Arabic and English Web forum associated with radical groups. Two classifiers namely, C4.5 and Support Vector Machine were used on the features. The results from their model indicated that SVM out-performed C4.5, and a high degree of success in identifying the communication pattern was produced.

Twenty thousand Webpages were collected with the aid of a WebCrawler to assess differences in five sentiment classes namely: anti-extremist sites, radical Islamic sites, radical right sites, sites that did not discuss extremism and news source sites discussing extremism [13]. That is, pages that relate to extremism or not. 198 frequently used keywords were identified through the aid of POS tagging. These keywords were used to calculate sentiment values for each page through sentiment analysis. The result obtained showed that the radical Islamic text class was classified at a much higher rate of success than the radical right text class. A WebCrawler called TENE-WebCrawler was designed to make a decision on each Web page it downloaded whether the page is pro-extremist, anti-extremist or neutral [5]. The process was achieved through the use of frequently used keywords as linguistic markers to pinpoint the sentiment on each page. The method was achieved through the combination of semantic orientation and data mining techniques to produce their classification.

Sentiment and social analysis were combined as a technique used to survey the agenda of a radical group on YouTube [36]. The polarity for each topic discussed within the group was obtained and explored to model individuals' behaviour. Eventually, it was spotted that extremism and intolerance were prominent among female users. Hierarchical clustering was applied to divide extremist Web pages into politics and religion categories [37]. Data retrieved from the Dark Web Portal Project was used to conduct the first proposed method to detect cyber recruitment efforts [38]. A sentiment-based classification method was employed for Twitter analysis classification [39]. Web Forums were used for opinion classification [40]. Twenty-eight (28) different extremist religion forum discussions translated from Arabic to English were compiled for annotation. Thereafter, the authors used a set of textual features and Bayesian criteria to classify the corpus. An accurate result was obtained, and the most predictive terms were highlighted [41]. Machine learning algorithms such as Naïve Bayes and Support Vector Machine were used to classify positive and negative features in given data [42].

The intensity of the sentiments of extremism was unraveled through sentiment analysis of social media multilingual textual data [43]. The research proposed a method that classifies textual views into four groups such as high extreme, low extreme, moderate, and neutral with respect to the degree of their extremism. A multilingual lexicon that was endorsed by domain experts which scored 88% precision was explored for the classification. Linear Support Vector Classifier and Multinomial Naïve Bayes algorithms were applied to the multilingual dataset. Eventually, Linear Support Vector Classifier produced better accuracy than Multinomial Naïve Bayes with an accuracy of 82%.

The semantic composition problems such as negative reversing and intensification associated with the use of conventional methods of annotating the sentiment of unlabelled documents which are based on sentiment lexicons or machine learning were discussed [44]. The research developed a sentiment-based classification method using negative and intensive sentiment added information to obtain the linguistic feature of

negative and intensive words as well as the topic information [44]. The method was applied to two datasets namely, a Movie Review and Stanford Sentiment Treebank. Eventually, the method was able to solve the domain-specific problem without depending on the external sentiment lexicons

Temporal sentiment analysis involves the findings of the sentiment pattern within a given period, a means for investigating the temporal patterns were proposed with the use of keywords in the comments [45]. A keyword based temporal sentiment analysis was developed, which comprises a sentiment classification technique and keyword clustering, in relating a few major events that happened during the period of investigation (19 November –20 December 2014). The results obtained in the experiment showed that temporal sentiment analysis with the use of keyword clustering can be explored to create the changes in opinions from the public relating to situation-events in a historically major election campaign in a developing country. The result revealed crucial information about the difference in the opinions during the election campaign which is difficult to discover by other means.

2.1.5 Posit-Based Classification Method

The proposed method presented in this thesis is an underpinning of Posit textual analysis that generates syntactic features of textual content from a Web page which are useful input for classification models. Existing studies on the Posit method are discussed in this section.

The Posit textual analysis toolset is a program written mainly in UNIX script and is capable of generating a detailed syntactic and frequency analysis of a textual corpus [46]. Posit outputs quantitative data from any text, including, word count, number of characters and sentences, number of tokens and types, n-gram frequencies and finally, part-of-speech tagging (POS) [47]. By default, the Posit produces data on 27 features. The features include noun types, possessive pronoun, personal pronouns, average

sentence length, determiners, adverbs values for total words (tokens), total unique words (types), type/token ratio, number of sentences, number of characters, average word length, verb types, adjective types, adverb types, preposition types, personal pronoun types, determiner types, types, interjection types, particle types, nouns, verbs, prepositions, adjectives and interjections. Posit extracts syntactic and quantitative values for textual data using part of speech tagging. It uses frequencies of syntactic features to characterise the given text. Posit textual analysis has been deployed for a diachronic analysis of English textbooks used in Japan. Posit was employed for the analysis and categorisation of a Scottish newspaper corpus [47].

Two different techniques were used for the automatic classification of extremist Web pages were collected from the Terrorism and Extremism Network Extractor (TENE) Web-crawler, a custom-built piece of software that browses the World Wide Web, gathering a large volume of data, retrieving the pages it visits, analysing them, and recursively following the links out of those pages. The techniques were contrasted [14]. The research aimed to determine the best automated classification system among the two approaches that can efficiently place each Webpage into the appropriate classes. The two approaches are Posit-textual analysis and a Sentiment classification rule-based technique that utilises top-k noun keywords to obtain the sentiment around each Webpage [5]. These techniques were applied separately on the extremist Web pages. A classification model was then developed on the features generated by each technique, using the J48 decision tree as the classifier algorithm. Eventually, the results obtained indicated that Posit-based classification results outperformed the results obtained from the sentiment-based classification method.

A machine learning algorithm was applied to the features which are numerical representation of texts generated from three different data sets through a tool known as Posit [48]. The tool generates features, such as parts-of-speech types and tokens instances and average sentence length. In addition to the aforementioned features, the bi-gram features were also included as the proposed added features. The effectiveness of

the method was tested on three datasets namely, drug, extremism-related texts and DBpedia text data. The objective of the research was to test the classification accuracy of the combination of Posit and n-gram features when a machine algorithm is applied. Then, the classification model was conducted on the datasets including 2-gram features. The results from the research indicated that the proposed added features (2-gram features) combined with the Posit features gave a limited improvement on the overall classification. In addition, the DBpedia dataset revealed that classifying a text corpus with numerous topics is inappropriate with the feature sets produced. The study also showed that transforming a text corpus to its numerical information produced by Posit is effective for classifying big datasets when a machine learning algorithm is applied.

A Posit tool was proposed that will allow agencies to separate and identify distrustful social network content. Posit analysis showed 99.8% precision in classifying fake news. Using Posit improves the possibility of achieving this aim, although it is still under research [49].

Three million social media posts were utilised for an automated classification system [15]. The posts were labelled by Russia's Internet Research Agency into fake or real news. TENE-WebCrawler developed at the International Cyber Crime Research Centre (ICCRC), Posit Toolkit, an improved version of Posit [46] and TensorFlow were the techniques employed for identifying hostile disinformation activities in the Cloud. The posts were classified with a slight increase in performance of Posit toolkits against the TensorFlow approach. The new Posit toolkit extends the basic word-level features to generate more 44 character features for each case of text data. The aforementioned features contain information on alphanumeric characters, and a subset of special characters, such as questions marks, exclamation marks, asterisks, periods and dollar signs. The augmentation of Posit to embrace character-level as well as word-level data produces the domain-neutral complexion of Posit analysis. Consequently, each data item (tweet) in the extended Posit analysis was represented by a set of 72 features. Each feature set from the techniques was fed into WEKA where J48 and Random Forest

classifiers were applied. The result from their study indicated that the upgraded version of Posit outperformed the result obtained from the TensorFlow implementation at a success rate of 90.1%.

2.1.6 Deep Neural Network Classification Method

Existing studies on the Deep Neural Network classification method are discussed in this section.

An overview of character-level Convolutional Networks as a method for text classification was described [50]. In the studies, the authors developed character-level convolutional networks for text classification. From their experiment, it was shown that a convolutional network could be implemented directly to a unique set of words in the absence of any information on the syntactic or semantic structures of the languages. Different datasets were explored to show that a character-level convolutional network could attain competitive results.

Users' posts on Twitter were classified into extremist and non-extremist groups using deep learning sentiment analysis techniques to detect and combat the spreading of bad ideology among different social media users [51]. The research proposed long short-term memory with Convolutional Neural Network (CNN-LSTM) model to achieve the research objective. The users' sentiments from the Twitter posts were classified based on their emotional affiliation such as positive or negative emotions with respect to extremist content. However, the proposed model lacks the automatic means of storing Twitter content, context-aware features, proper visual display and investigating other extremists. The authors recommend that using context-aware features and advanced techniques like an attention-based mechanism for extremist affiliation detection with multi-class labels will improve the performance of the system [51].

An approach to detecting terrorism based on sentiment analysis of users' posts on Twitter was developed [52]. According to the study, users' sentences on the Twitter platform are analysed and categorised into three areas namely positive, negative and neutral about the sentiment opinion of users leading to an act of terrorism. To achieve this, the Naïve Bayes algorithm was improved and used to predict the categories in which any given Twitter post belonged. This is done by looking for certain keywords which the users have used in the post, assigning a score to it concerning terrorism, comparing it with the previous posts, and ranking the value obtained to know their influence on the subject. This however does not only provide benefits as terrorist detection but also helps to determine the categories of text especially in combating digital issues.

The multiclass event classification from texts on social media about the Urdu language text was examined [53]. Deep learning techniques such as the convolutional neural network (CNN), recurrence neural network (RNN) and the deep neural network (DNN) were applied. However, the DNN classifier outclassed other algorithms with 84% accuracy in the extraction and classification of text. Sentiment analysis on the opinion of people expressed on Facebook as regards the COVID-19 pandemic in low-resource languages with a special inclination to the Albanian language was conducted [54]. Three neural networks including the 1D-CNN, BiLSTM and the 1D-CNN + BiLSTM models were deployed revealing that the optimal combination of the BiLSTM with an attention model yielded the best performance at 72.09%. Supervised machine learning techniques were compared for sentiment analysis of Covid-19 tweets [55]. The LSTM model was compared with the Vader sentiment analysis and the GloVe feature extraction approach and it was discovered that the LSTM has more accuracy than other techniques at 93% accuracy.

A framework was developed using a Recurrent-Convolutional Neural Network, based on pre-trained word embedding to address the problem of the automatic classification of the extremist activities on Twitter, most especially the Islamic State of Iraq and al-Sham

(ISIS) activities [56]. The method was implemented on 15,684 ISIS propaganda tweets, a mix of neutral tweets, connected to ISIS, and random ones, creating imbalances up to 1%. The proposed method was compared with other methods such as, a character-based CNN model, a RCNN, merged with max-pooling (based on pre-trained FastText word embeddings) and SVM trained on bag-of-character and bag-of-word n-grams. The method was evaluated based on varying the training schemes and the test conditions. The result obtained from the research was able to demonstrate that the proposed framework attained a F1 score as high as 0.9 when trained with the same imbalance.

2.2 Imputation Methods

This section describes imputation approaches used to compensate for missing values.

A critical issue in the classification task is the missing values found in some datasets. Missing data is defined as values or data for some variables in a dataset that is not recorded (or non-existent) [1]. Most classifiers cannot cope with null entries which could be missing data. Missing data, in this case could impair the accuracy of data analysis or when classification algorithms are applied, as the value of the data has degraded. A classifier learns from data and misrepresentation of facts in data will lead to wrong information learned and hence incorrect or biased classification occurs. A better approach to the missing data is the imputation method. Listed below are some of the widely used imputation methods.

2.2.1 MICE Imputation

Multiple or Multivariate Imputation by Chained Reactions also known as MICE imputation is a means of handling non-response bias which occurs when certain respondents do not respond to a survey leading to the presence of missing data [57]. Therefore, the MICE or multiple imputation is a method used to replace missing data values in a data set given the conditions that the data is missing completely at random (MCAR), non-ignorable missing or missing at random (MAR) [58]. In simple terms, the

MICE imputation points to an approach where missing values in a dataset are replaced with probable data which are sourced from the distribution but modelled for each missing value through the use of chained equations [17].

Meanwhile, the MICE imputation has been preferred by statisticians because of its flexibility in handling varying nature of data such as the continuous or binary data and other simulation studies while it also addresses intricacies emanating from bounds or survey skip patterns [57], [59], [60]. In other words, the algorithm can correspondingly, [61] noted that although there are various means of handling missing data, the complete case analysis though simple to adopt is not as efficient as the MICE imputation because it requires more missing data assumptions which may rarely come by in real life computations and as such lead to being biased. Furthermore, the single imputation method has also been discovered to fall short on the grounds of accounting for uncertainty which will also lead to inaccurate results [61], [62]. Therefore, the MICE imputation is more beneficial on the grounds of flexibility as it can be applied to a different range of the dataset. Also, because it multiple times fills in the missing values by creating multiple and seemingly complete datasets, the missing values are filled in premised on the observed values while it further accommodates and handles uncertainty by providing accurate standard errors [57]. MICE imputation model is advantageous because it can account for the data creation system as well as the preservation of the uncertainty that pertains to the dataset [61]. Furthermore, the MICE approach was developed to address the problems that are associated with the multivariate imputation approach which was noted by [17]. These problems include the circular dependence that can occur in the dataset as the imputed data values may lose their specific independence because they may indirectly depend on other values used to model them.

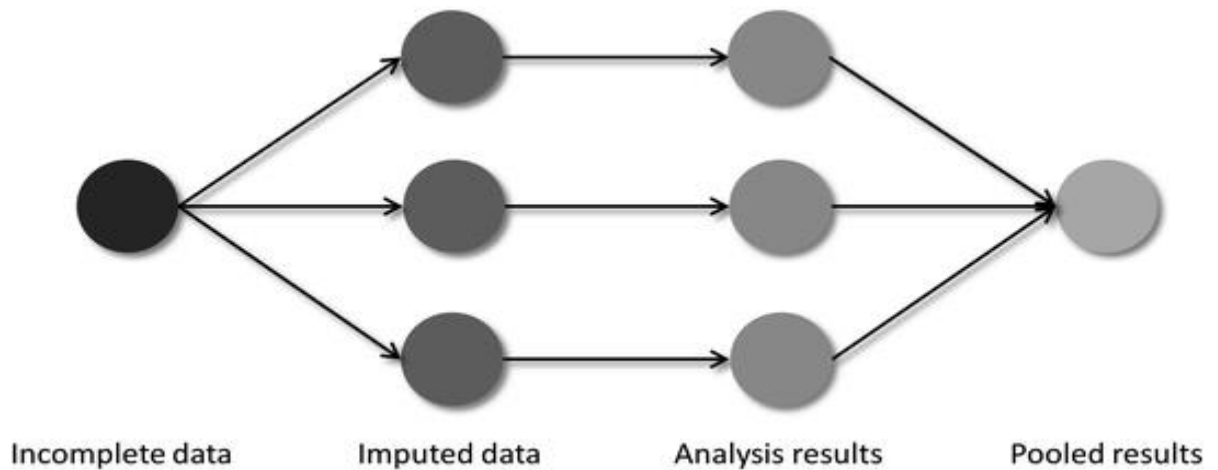


Fig 2.0: MICE Imputation Framework Source: [64]

2.2.2 K-Nearest Neighbors (KNN) Imputation

The KNN imputation approach seeks to fill in missing data by most similar values or the nearest neighbors of the instance of interest. As such, the similarity between the instance of interest and the missing data is determined by a distance function algorithm [65]. The KNN imputation has been considered advantageous because quantitative and qualitative values can be predicted by the approach and it does not equally have to provide a model to predict each missing data [65]. Also, this approach can retain the variance-covariance structure of the dataset as far as the $k=1$ condition is satisfied [66]. In addition, the approach is less susceptible to model misspecification due to its non-parametric nature which does not require models in relating datasets [67]. In precis, the KNN is good on the basis of simplicity, comprehensibility and scalability [68].

On the other hand, the limitations of the KNN include its high cost of computation which makes it difficult for it to be applied to real-time situations, high storage prerequisite and responsiveness to noise [68] In addition, it involves several pre-processing procedures like the screening or data splitting among others which is time consuming. Also, [68]

noted that noise tend to cause issues related to difficulties in convergence and accuracy of classification.

2.2.3 MEAN Imputation

The Mean imputation is a method where the missing value in a dataset is filled in by the mean value of the available or non-missing data values [69]. That is, it is a single imputation approach where some mean values of ascertained data is used to replace missing slots in a data set [70]. One advantage of this approach is the absence of complexity that is associated with its calculation. Furthermore, the approach preserves the mean value of the observed data especially when data is missing at random, as such the mean value still remains unbiased [70]. Furthermore, the use of the mean imputation also guarantees a complete sample size as the approach ensures that the full sample size is kept and as such will not lead to problem in parameter estimates [61]. However, the size of the covariance and correlation tends to reduce and as such tends to cause bias in estimation especially when the relationship between variables is to be considered [71]. Furthermore, this tends to reduce the standard error of the mean and consequently, the probability values attached to the variables under consideration will equally be reduced leading to another major bias in estimation [69], [72].

2.2.4 MissForest Imputation

The MissForest imputation is a non-parametric approach premised on the random forest algorithm that can handle any kind of data whether they are characterized by mixed variables, high dimensionality or non-linear relations [73]. However, the only requisite for the execution of the approach is that the observation must be pairwise independent [73]. The approach presents an estimate of the imputation error which [73] assumes that it is accurate to a very large degree. MissForest imputation outperformed other imputation methods explored in their study [73]. The approach is centered on random forest-based iterations which occur after the mean/median imputation has been done to predict a transformed dataset to fill in the missing data [74]. Also, contrary to the

provisions of the KNN technique, this approach does not require pre-processing activities while it also efficiently handles noise and multi-collinearity in the dataset. In addition, the approach is not subject to the curse of dimensionality and requires no tuning because of its non-parametric approach [74]. However, the approach has some limitations which include its nature as an algorithm rather than a model object. As such, it cannot be stored and therefore requires fresh processing each time there is a need for missing data imputation which may not be comfortable in some quarters [75]. Furthermore, it wastes more imputation time because it increases the number of predictors and observations in a bid to fill missing values while it equally subjects the dataset to the lack of interpretability of random forests [75].

2.3 Research Gaps

This section describes the gaps in the published research.

Textual classification methods such as Machine Learning, Semantic Orientation [34], Topic modelling [26-27], Posit [14 48, 49, 15], the linguistic maker (keyword) model are used in the Sentiment analysis 45[5, 13, 75], a bag of words [11, 21] and vector space model [12, 23] have been reported in the literature. However, conventional methods such as a bag of words and vector space model are faced with many limitations such as high dimensional feature vector encountered due to large size of vocabulary, the model disregards semantics of the word (the word ‘automobile’ and ‘car’ could be used in the same context) [11]. Also, highly sparse vectors occur when there is a nonzero value in the dimensions related to the words that appear in the sentence [11]. Moreover, a vector space model also suffers from synonym and polysemy. The model is semantically insensitive (documents with similar context but different term vocabulary cannot be connected). Hence, a negative match occurs. It theoretically assumes that terms are statistically independent. Lastly, long documents do have poor similarity values [12].

In recent times, methods such as Sentiment [5] [13] and Posit analysis, both at word-level [14] and character-level [15], to mention a few, have been explored to characterise

a set of text for use in a classification model which is the focus of this research because they are hot-trend methods useful in market and scientific research in the area of Machine Learning and Natural Language Processing. However, the use of the linguistic marker technique in some sentiment analysis (i.e. the use of frequently used keywords) such as [5] [13], relies on the use of keywords to obtain sentiment around each Web page and reducing the number of keywords to top k-nouns (which often carry the sentiment) will reduce the dimension and matrix sparseness, this poses another challenge as chances are there that a fraction of the Webpages in each class has few or none of the selected keywords thereby leading to non-capture of sentiment values of such Webpage, this may impair classification accuracy of such Webpages.

However, the aforementioned are the knowledge gaps observed in the literature and the focus of this research is the means of analysing text to extract useful feature information that would enhance automated classification.

Chapter 3: Methodology

This chapter contains the description of the data and the methods adopted in the thesis.

3.1 Experimental Design

The experiments conducted in this research are developed on the foundation or justification of research questions that could be evaluated to determine if the assertions are correct or not. Sections 1.3 detailed our research questions. However, it is important to sustain a consistent experimental setup to achieve valid conclusions. The effectiveness of the methods explored in the study is analysed and tested on ICCRC extremist data. The Web data retrieved from Nigerian domain sources would be used to test the validity of a trained dataset obtained from ICCRC. The six different text feature representations considered in this thesis for the various classification tasks are Sentiment (KNN, MICE and MissForest Imputation), Posit, Extended Posit-based analysis and the proposed method, a Composite-based analysis. The concepts and the design of the methods carried out in the study are described in Figure 3.1. The descriptions of the datasets, data preparation and data pre-processing phases of the model development for the experiments are given below.

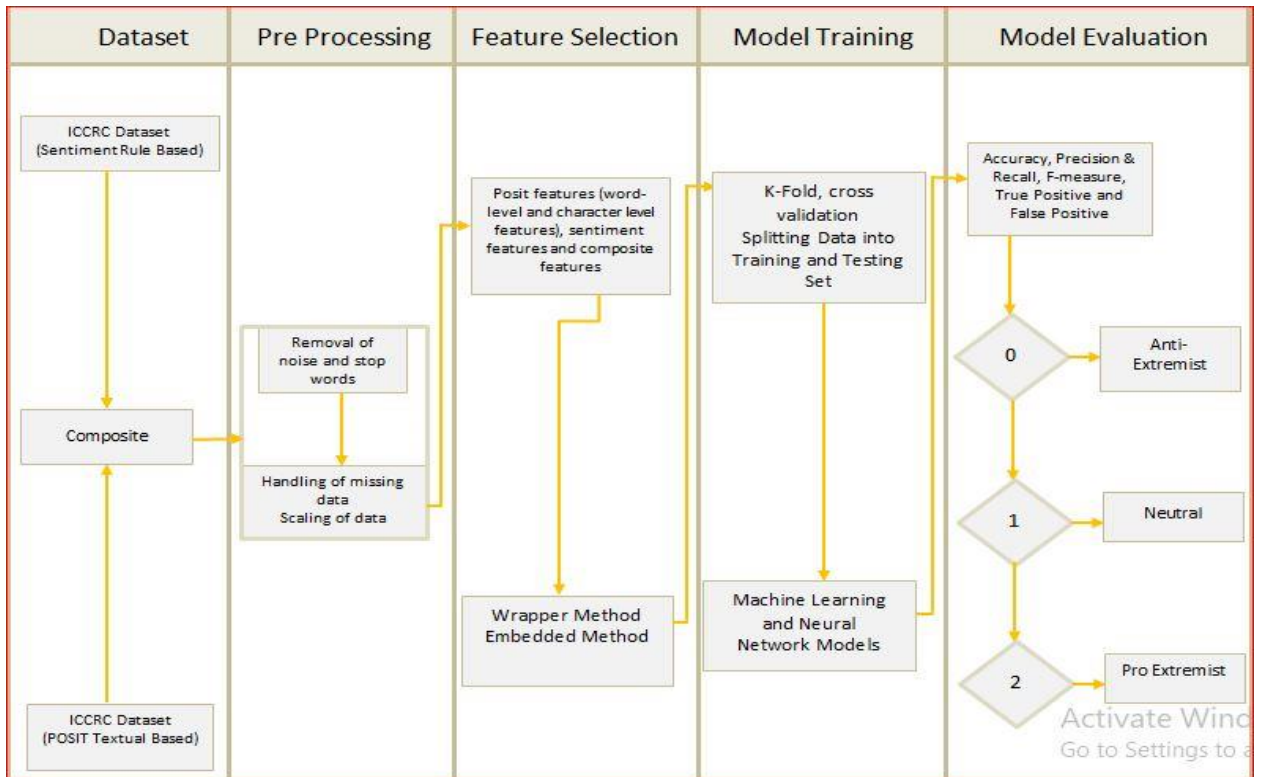


Figure 3.1: The Architecture of the Research Methods

3.2 DATA SOURCES

In the following, two manually classified datasets are presented. The origin of each dataset differs and this produced a broad range of texts with similar topics. These datasets have been employed to test the effectiveness of the frameworks for efficient text classification. General descriptions of the dataset are given below:

(i) Extremism dataset (referred to as ‘ICCRC dataset’): The Webpages were obtained from extremist Websites using the TENE-WebCrawler. The TENE-WebCrawler is a software developed at the International Cyber Crime Research Centre (ICCRC), Simon

Fraser University, Canada. This crawler follows links based upon keyword searches through the Internet, extracts Web pages and analyses each page visited [5]. One set of such Web pages was initially subjected to manual classification by ICCRC personnel, whereby each Webpage was grouped as "pro-extremist", "neutral" or "anti-extremist" based on its contents. The data retrieved for this manual classification process comprised 7500 Web pages manually classified as indicated above.

The Webpages were classified with respect to their content. For example, the neutral group reflects content from the media/news that reports impartially on terrorist events. In the neutral class, 2500 Web pages were derived from 30 Websites. The anti-extremist class contains Web content that reports the countering of terrorism and operations of intelligence agencies. The anti-extremist class consists of 2500 Webpages from 10 Websites. Pro-extremist pages express extremist content from extremist and jihadi organisation Websites. Examples of such Web sources are white supremacist forums and America-based neo-Nazi forums. In this class, 2500 Webpages were obtained from 11 different Websites. However, a balanced dataset in ICCRC Extremism Web-data (2500 Web pages in each category) is used, this is to create unbiased results among each class.

(ii) Nigerian Extremism Dataset: Websites with extremism topics were retrieved from Nigerian Websites with the aid of the Beautiful Soup framework [18]. This is a Python library used to download Web page content, automatically scrape HTML data from a Webpage and present it in a plain text format. This software was used to retrieve Nigerian domain Websites with extremist topics. The retrieved data comprised 210 Webpages from ten different extremist Websites. These data were later classified manually with the aid of a qualitative research tool, NVivo [19] which permits future analysis of the data. The Webpages are categorised based on its content into three classes; pro-extremist, anti-extremist and neutral, with 70 Web text documents in each class. Examples in the pro-extremist class include content retrieved from recognised extremist websites such as radio-Biafra (Website for the agitation for Biafra nation), Boko-Haram (website for a group of terrorists in the Northern part of Nigeria) IPOB

(website for a group of people in the Eastern part of Nigeria agitating for Biafra nation), and Niger-Delta Avengers (Websites for a group of militants in South-Southern part of Nigeria who feel they are being exploited). The forums consist of content discussing a referendum for independence, agitation for independence, war, opposition to the government, hate speeches, killings, vandalising of oil wells and pipelines and religious radicals (Boko-Haram). This category of Web page content consists of 70 Web pages that were obtained from 10 different Websites. The neutral content was obtained from media sources that could be expected to report generally on terrorist occurrences from a more unbiased, journalistic perspective, including sites such as lindaikeji (blog) and Nairaland (blog). In this neutral class, 70 Web pages were retrieved from 11 Websites. Finally, the anti-extremist category contains 70 Web pages obtained from 9 different Websites. The anti-extremist content reveals opposition to violence, for example, counter-terrorism Websites such as the Nigerian police, and Economic and Financial Crime Commission (EFCC) forums.

3.2.1 The Dataset's Complexion

Complexion analysis unravels any discrepancies in characteristics between the data items to be classified. There might be a situation where a unique feature might excessively influence the automated classification process. However, the complexion analysis helps to make the subsequent examination of key attributes of the data to have knowledge of the likelihood of such influential factors such as number of words, number of characters, number of special characters, as well as maximum, minimum and average values for each of these features. Table 3.1 revealed different distributions within the features such as Total Words, Number of Sentences Type/Token Type and Average Sentence Length in the datasets. Posit analysis is explored to shed light on the complexion analysis of the datasets. The complexion's components are described below such as the data points, spread, skewness and the coefficient of variance.

Data points

The middle of the data set is regarded as the central location which is described by mean, median or mode. Mean is regarded as an average value of the data points, the value in the dataset that appears most is regarded as a mode while the mid number when the data point is placed from low to high is known as the median

Spread or Dispersion

The extent to which a distribution is expanded or compressed is known as dispersion, also known as the spread. Examples of statistical dispersion metrics are mean, variance and standard deviation. For example, when the variance of data in a collection is high, the data is well dispersed. When the variance is at modest (low), however, the data in the set is clustered [76].

Coefficient of variance

A fraction of the standard deviation to the mean is regarded as the coefficient of variation. The coefficient of variation is a statistical measure of the dispersion of data points around the mean (relative standard deviation). When comparing data dispersion between distinct data sets, this metric is widely utilized. The coefficient of variation, unlike the standard deviation, which must always be assessed about the data's mean, is a simple and quick way to compare different data sets. [76]

Skewness

Skewness estimates the asymmetry of a real-valued random variable's probability distribution around its mean. Positive, zero, negative or undefined skewness are possible values in skewness. The tail on the left side of a unimodal distribution is indicated as negative skew while positive skew shows that the tail is on the right-side [77]. Skewness does not follow a general rule when one tail is long and the other is big. A zero value

indicates that the tails on both sides of the mean balance out in the overall distribution, for example, this is true for both symmetric and asymmetric distributions with one long and small tail [77].

3.2.2 Attributes of the Dataset

The ICCRC Extremist data has a mean of the number of total words of 1955 in a text with a maximum of 75127 (illustrated in Figure 3.2). The standard deviation for total_words is far above the mean and above zero, making the data points spread away from the mean showing a lot of variation. With a skewness value of 7.28302, the distribution is skewed right with a tail, also showing a spread of variation towards the increasing positive x-axis- indicating data points/outliers that are greater than the mode, showing some variation. The coefficient of variation is also >1 , showing high variance of data points. From the plotted histogram, the x-axis is the range of values while the y-axis is the frequency for the value ranges.

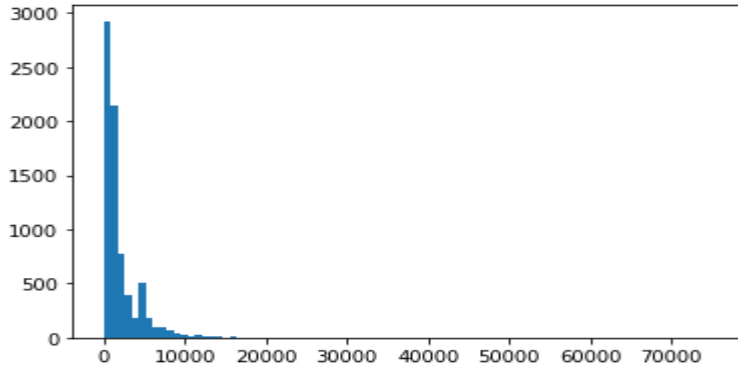


Figure 3.2: Total words for the Extremism (ICCRC) Data

The number of sentences in the ICCRC Extremist set contains 59.9. The highest number of sentences for the ICCRC data is 2450, but the plot shows that bulk of the data contains less than 408 sentences. This is shown in Figure 3.3. The standard deviation for number_of_sentences is above the mean and far above zero as well, making the data

points spread away from the mean showing a lot of variation. With a skewness value of 8.18498, the distribution is skewed right with a tail, also showing a spread of variation towards the increasing positive x-axis- indicating data points/outliers that are greater than the mode, showing some variation. The coefficient of variation is also >1 , showing a high variance of data points. From the plotted histogram, the x-axis is the range of values while the y-axis is the frequency for the value ranges.

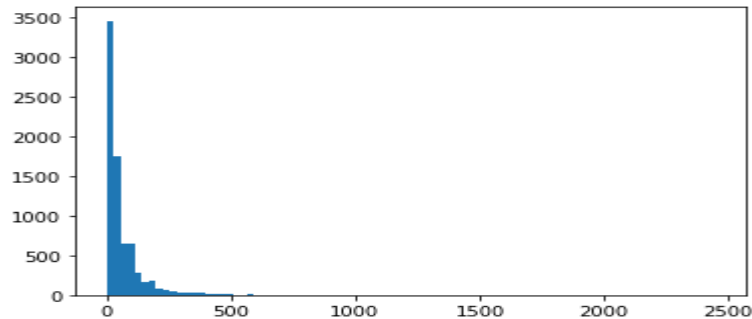


Figure 3.3: Number of sentences for Extremism (ICCRC) data

For the ICCRC extremist data, the mean average sentence length is 48.086, and the maximum is 1716, both of which are doubtful to be actual sentences because the data's maximum is still extremely high (displayed in Figure 3.4). However, the data average sentence length values suggest that the datasets may contain numerous texts that are not structured in sentences. The ICCRC extremist data has compressed plot, with only a few texts around 250. The standard deviation for average sentence length is close to but still above the mean and far above zero as well, making the data points spread away from the mean showing a lot of variation. With a skewness value of 11.5183, the distribution is skewed right with a tail, also showing a spread of variation towards the increasing positive x-axis- indicating data points/outliers that are greater than the mode, showing some variation. Coefficient of variation is also >1 , showing high variance of data points. From the plotted histogram, the x-axis is the range of values while the y-axis is the frequency for the value ranges.

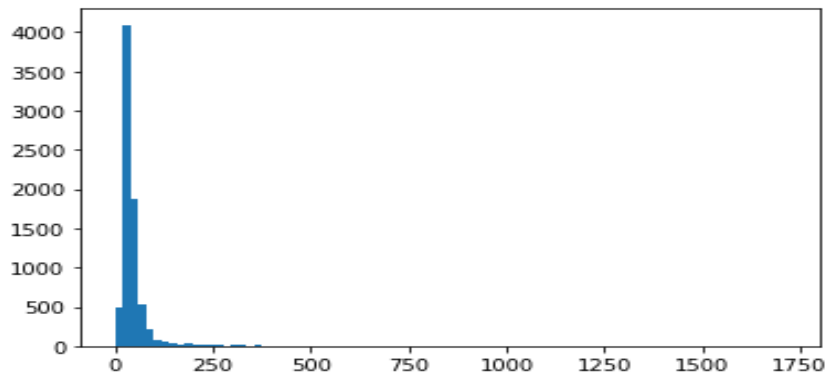


Figure 3.4: Average sentence length for the Extremism (ICCRC) data

The standard deviation for average word length (awl) is well below the mean and close to zero, making the data points spread towards the mean showing less variation. With a skewness value of 3.17162, the distribution is skewed right with a tail, also showing a spread of variation towards the increasing positive x-axis- indicating data points/outliers that are greater than the mode, showing some variation. The coefficient of variation is also $\lll 1$, showing less variance of data points. From the plotted histogram, the x-axis is the range of values while the y-axis is the frequency for the value ranges. There is little variation in this column of the feature. This is shown in Figure 3.5

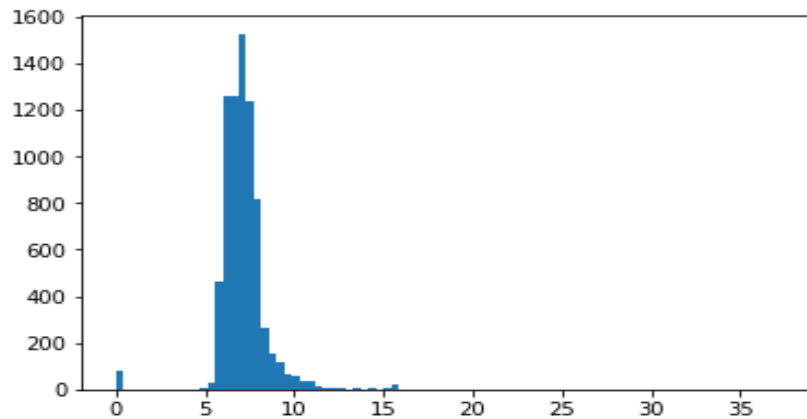


Figure 3.5: Average Word Length for the Extremism (ICCRC) data

The standard deviation for the type/token ratio is below the mean but still above zero, still making the data points spread away from the mean showing a lot of variation. With a skewness value of 1.56646, the distribution is unimodal but has peaking outliers as multiple peaks, also showing a spread of variation towards both positive and negative x-axes- showing some variation. From the plotted histogram, the x-axis is the range of values while the y-axis is the frequency for the value ranges. The low coefficient of variation (0.33667) suggests a low level of variation. This is shown in Figure 3.6. Table 3.1 shows the summary of the complexion analysis of the Extremist (ICCRC) dataset.

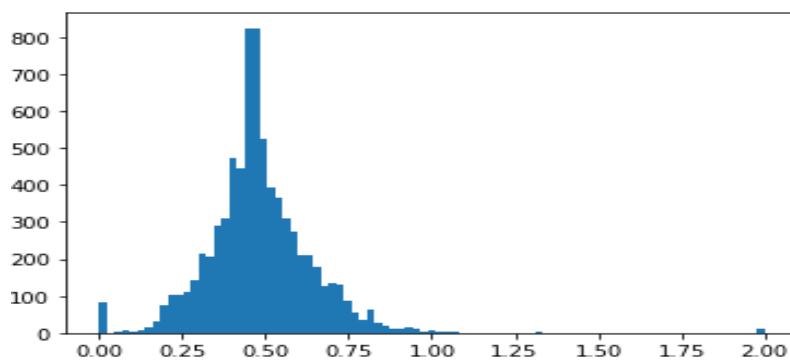


Figure 3.6: Type/Token Ratio for the Extremism (ICCRC) data

Features	Min	Max	Mean	Std Dev.
Total Words	0	75127	1955	2672
Number of Sentences	0	2450	59.962	95.69
Average Sentence length	0	1716	44.086	45.717
Type/Token Ratio	0	2	0.48	0.16
Average Word Length	0	37	7.12	1.51

Table 3.1: Complexion Analysis of Extremist (ICCRC) Dataset

From this analysis, we can see a series of variances on the data's distributions of some of the features in the ICCRC extremist dataset. The same situation was observed in Nigerian data. To avoid tautology, the discussion in Section 3.2.2 is valid for Nigerian data. Hence, both datasets were scaled to avoid situations in which a unique feature may excessively influence the automated classification process. Consequently, a version of each dataset was normalised and the second was standardised to know which scaling technique works best on individual data in each framework. The scaling techniques are further explained in Section 3.3.

3.3 Data Preparation

This section describes the process of transforming and cleaning of raw data before the classification process. The process improves data quality, increases efficient analysis, reduces error and inaccuracies that can occur to data during processing.

Data Cleaning

Raw data containing noise is unclean, such data degrade the quality of the classification result. However, pre-processing helps to process noisy data to enhance machine learning algorithm's performance. To enhance classification accuracy, dataset is pre-processed before being fed into machine learning. The pre-processing steps include noise cleansing, dealing with missing values and scaling the data. The detailed process is described below:

Corpus linguistics entails analysis executed on a text corpus. Therefore, analysing text in terms of frequency distribution of keywords requires the text to be cleaned from unwanted information, this is an important processing step when using machine learning algorithms. However, the text documents retrieved from the extremist Websites were loaded into Python which was scanned as strings of text. Cleaning was performed with

the NLTK (Natural Language Toolkit) and SciKit library. NLTK and SciKit roles include the transforming of all words to lower case to enhance accuracy in the analysis and stop word deletion. Examples of stop words are “the”, “a”, and word length of one or two characters that contain less meaning in large texts.

A Porter Stemmer algorithm is employed to reduce all words to their stem or root. A stemmer converts words such as “Twitter” and “Twitting” to “twit”. Stemmer improves the accuracy of a linguistic analysis and helps to avoid missing potential sentiment in a textual corpus and helps to remove some URL’s in the textual files. A model’s prediction accuracy could be drastically reduced with invalid or missing data and needs to be prevented. However, the data generated for Posit analysis does not return any missing values but returned zero or -1 values for an instance where text was not correctly encoded or not in the English language. However, these erroneous instances were amended by correcting the language and the encoding in the pre-processing stage and the file format explored is a csv file format. In addition to the pre-processing approach, the dataset was standardized and normalised.

Normalisation of the Datasets

Sklearn library is explored in this thesis which is a pre-processing library, it contains functions to normalize and standardize the data [78]. Data was normalised by importing the MinMax method and applying it to our train dataset. The method takes an array as an input and normalizes its values between 0 and 1. It then returns an output array with the same dimensions as the input.

Standardization of the Datasets

Subtracting the mean of each observation and then dividing by the standard deviation is the procedure of standard deviation [78]. The features are rescaled to have the attributes of a typical normal distribution with standard deviations:

$$z = \frac{X - \mu}{\sigma} =$$

$\mu=0$ and $\sigma=1$ where x denotes the observation, σ is the standard deviation from the mean which is set to zero and μ is the mean set to 1. The sci-kit-learn StandardScaler library is explored for this task and scales the data to unit variance. Hence, all the variable values fall within the same range

3.4 TOOLS

The tools employed for concepts and the design of the methods carried out in this study include Google Colab GPU, and Python with its packages (Pandas, Sklearn and TensorFlow) were used to implement the classification models. A description of each tool is detailed in later Chapters of this thesis. Figure 3.7 describes the tools explored for the study.

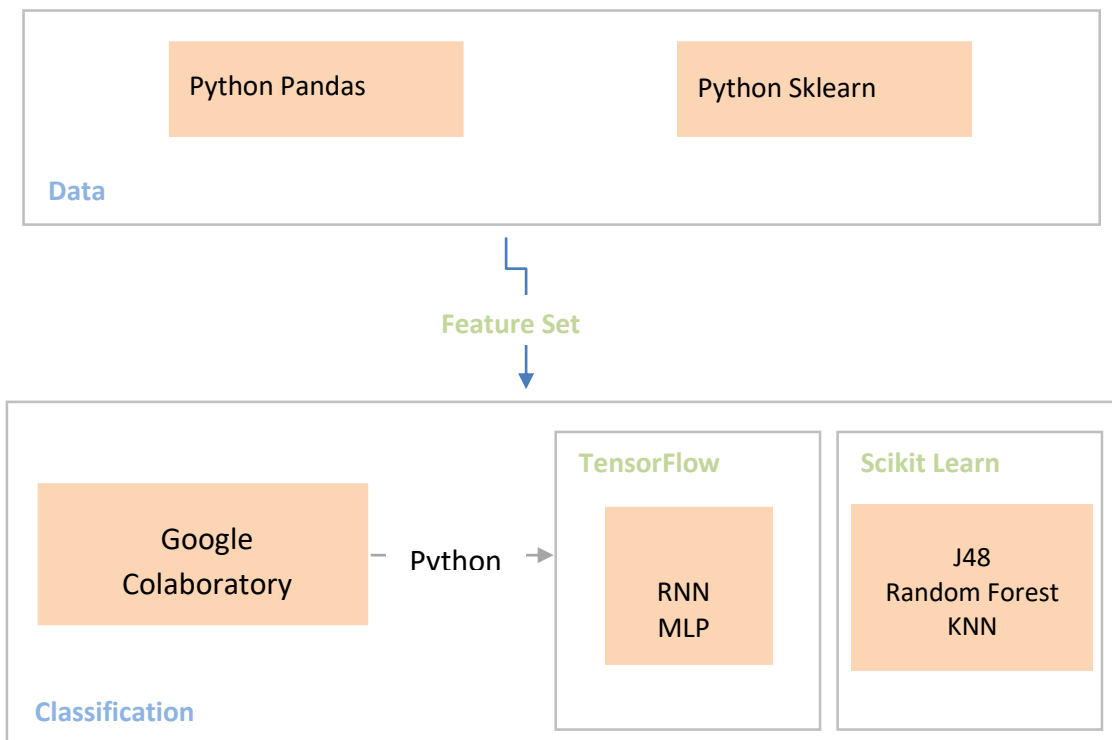


Figure 3.7: The Tools Process

In the setup for the Sentiment analysis experiment, one set of extremist Web pages obtained from ICCRC was split into three folders representing the categories anti-extremist, pro-extremist and neutral, with 2500 Web text files in each folder. In the experiment, a part-of-speech (POS) tagger in Posit analysis was applied to each folder (pro-extremist, anti-extremist and neutral) to tag keywords in their parts of speech.

3.5.1 Feature Extraction Process.

Part-of-speech (POS) tagging in Posit analysis [8] was applied of the extremist Web text where top ten most occurring nouns (keywords) were chosen from each class and later aggregated into one list, after disregarding duplicates, symbols, stop words and non-words, we arrived using 26 keywords across the three categories (pro-extremist, anti-extremist and neutral). Table 3.2 presents the list of the keywords. The noun keywords were utilized to find terms on each page that showed a high level of sentiment. This is because the context around noun keywords contains more sentiment [5], [13]. Additionally, each page had a scope of five words on either side of each term, and the output was input into Sentistrength to generate each page's sentiment value, which was taken from Sentistrength's General Inquirer lexicon. Sentistrength has a high accuracy level for brief non-political Web texts in English [16], hence scope of five words was used.

Consequently, the feature set is contained in a csv format where each page comprises noun keywords with their corresponding sentiment scores and the manual label. SentiStrength can also produce outcomes that are binary (positive/negative), trinary (positive/negative/neutral), or single-scale (-4 to +4) [16]. However, this study explored single scale (-4 to +4) results. The aforementioned approach explained how we converted the Web text obtained to numeric for the machine learning model. Figure 3.8 shows the sentiment feature extraction process.

1. Syria	15. Politics
2. Counter	16. President
3. terrorism	17. Press
4. Program	18. Rights
5. Affairs	19. Safeguards
6. Court	20. Syria
7. Ebola	21. Trial
8. Facebook	22. Twitter
9. Islam	23. CNN
10. Jihad	24. Crime
11. Military	25. Victims
12. Muslim	26. War
13. News	27. Security
14. Policy	

Table 3.2: Noun Keyword List

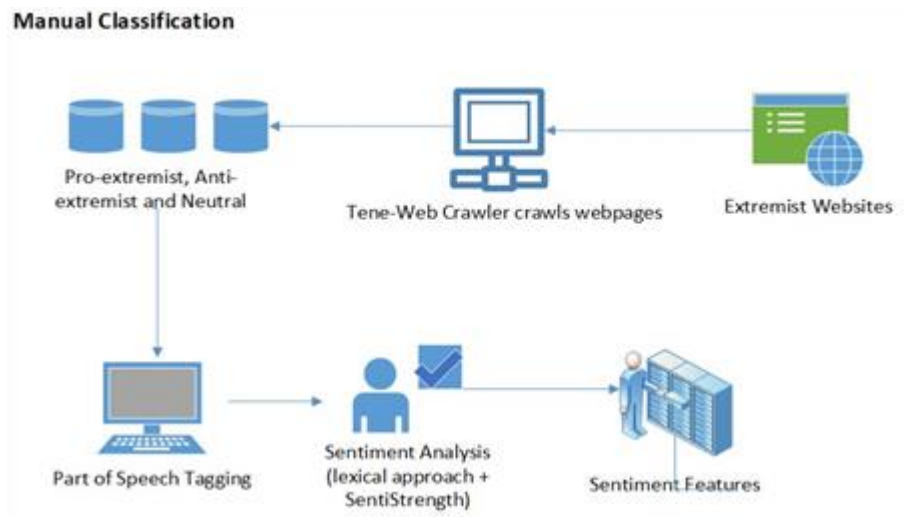


Figure 3.8: Sentiment Feature Extraction Process

The technique considered in this sentiment analysis approach uses top-k noun keywords to obtain the sentiment around each Webpage. However, reducing the number of keywords to top k-noun keywords poses a challenge because there were some of the Webpages in each class have few or none of the selected keywords thereby leading to data sparseness and non-capture of sentiment of such webpage(s). This set of Webpages that have no associated sentiment value(s) are sometimes encoded as blank or NaNs (missing values) which cannot be denoted by 0 as zero represents neutral sentiment in this experiment.

Consequently, we got a data missing completely at random (MCAR). Figure 3.9 displays the data frame of the dataset. Missing data can cause an disparity in the dataset, leading to poor model analysis, regardless of the type of the missingness either (the data is missing at random (MAR) or missing completely at random (MCAR) or missing not at random (MNAR). Missing not at random (MNAR) refers to a circumstance in which the missingness cannot be explained by the observed variables. Missing completely at random (MCAR) describes a condition in which the missing values are unrelated to any other values, whereas data missing at random (MAR) describes the opposite [17].

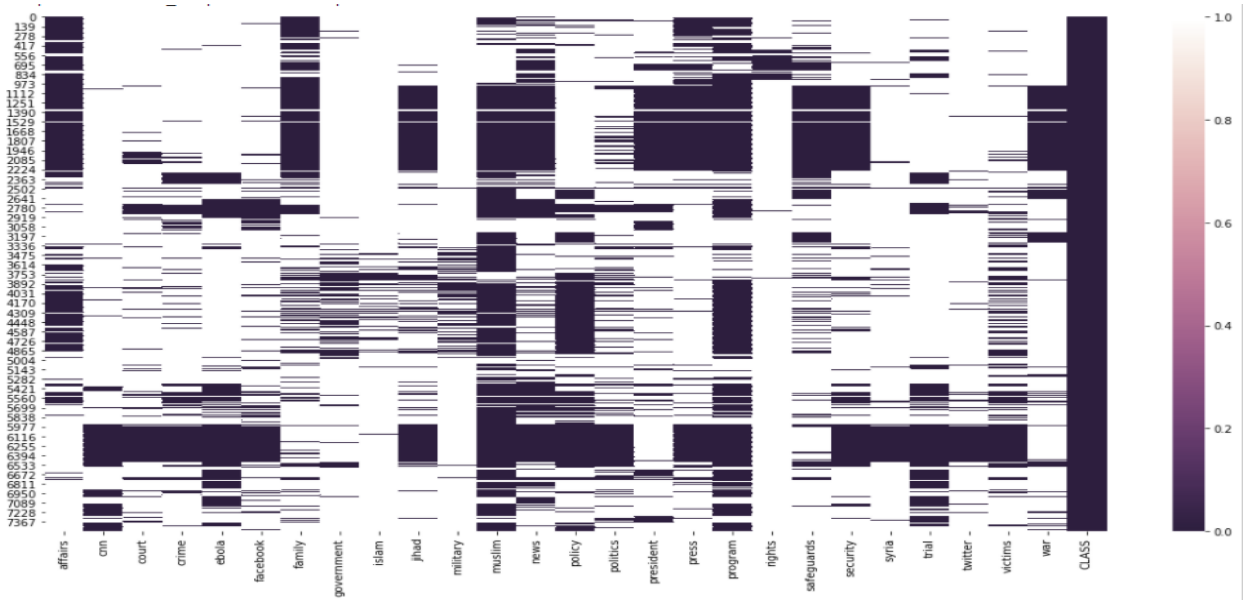


Figure 3.9: Data Frame of the Dataset

The easiest approach to a missing data problem would have been dropping the missing data but this is dangerous because the deleted data can be informative [60]. A better approach to the missing data is the imputation method. Imputation is a method for replacing missing data with an approximation based on other available data [17]. In the process of the imputation, a value according to the accessible data is calculated and later preceded into the substitution process [17].

There is no ideal or accurate technique to make up for missing values in a dataset. For some datasets and missing data types, each strategy may perform better, but for others, it may perform substantially worse [57]. However, how to calculate value from the accessible data led to the different imputation methods explored in this thesis to handle missing data. Machine learning imputation-based techniques are adopted because of their better performance compared to statistical-based imputation [59],[65],[67]. Eventually, the implementation of imputations methods on a sentiment-based approach (the procedure that utilises top-k noun keywords to obtain sentiment values from text corpus

before being fed into machine learning) generates different three versions of ICCRC datasets namely; KNN, MissForest and MICE dataset. Each process is described below.

3.5.2 Imputation Methods

K-Nearest Neighbor Imputation

The missing values in the ICCRC dataset were filled in using the scikit-learn class `KNNImputer`. The approach employs the core KNN algorithm which is more beneficial than the oversimplified approach of replacing all values with the mean or median. In the experiment, the K parameter, also referred to as the distance from the missing data is supplied. The mean of the neighbors was used to predict the missing number. The `KNNImputer()` library actualizes this and takes the following arguments:

`n_neighbors`: this refers to the number of data points that should be included that are closest to the missing value.

`Metric`: By default, `nan_euclidean` is used as the distance metric when finding values

`Weights`: by default `uniform` is used to evaluate the basis on which neighboring values should be handled, values such as `{uniform, distance, callable}`.

Multivariate Imputation by Chained Equation (MICE)

One of the effective method of addressing missing data in a set of data is multiple imputations by chained equations. In this study, the following steps were actualised to obtain the MICE imputation based dataset:

1. All features are imputed using a simple type of imputation, such as Mean Imputation.
2. A feature's values are reverted to missing.

3. a regression analysis is implemented on the seen values from the target variable in the above step of the approach by exploring other variables in the imputation model
 4. The missing values in the column were replaced by regression model predictions (imputations).
 5. For each variable for which there are missing data, steps 2-4 are repeated.
 6. steps 2-4 were conducted simultaneously while updating the imputed values each time.
- 5 The cycles were performed for the experiment where the optimum performance was reached (the coefficients in the regression models converged hence the model became stable). The final imputations are kept at the end of these cycles, creating a single imputed dataset.

Missforest Imputation

In the MissForest version of the dataset, the study explored the mean to impute all missing data, then fits a random forest on the seen portion and forecasts the missing part for each variable with missing values (i.e. the training set is the observed observations, while the prediction set is the missing values). This training and prediction approach is iterated until a stopping criterion is met or a user-specified maximum number of iterations is reached. Once all variables with missing data are filled in, one imputation cycle is completed. Consequently, in this experiment, the imputation process is repeated several times. The reason for the numerous iterations is that, beginning with iteration 2, the random forests that perform the imputation are trained on higher and better quality data that has been predictively imputed. Consequently, the optimum performance was achieved after 4 iterations.

3.6 Posit Experimental Set-up

A Posit API was developed and employed in the experiment so that, when applied to the dataset, Posit produced data on word-level features. The Posit API is an extension of the actual Posit system, built using Django, Python Shell and the AWS Elastic Beanstalk framework. The Posit API has two endpoints: `api_posit` and `result_name`, where `result_name` is a unique id auto-generated for a particular Posit call. The `api_posit` endpoint receives an http POST request from any services with a zipped input file of key "file_input". The request call triggers an inner function that performs the Posit analysis and returns a zipped output of the result. The `result_name` endpoint receives a GET request to download the result of a particular `api_posit` call. When it receives a request call, it triggers an internal function that searches the AWS Linux file system for the result of the `<result_name>`. Once it finds it, it returns the zipped file back to the request call. The API has some language binding in Unix Shell, Ruby, Python and Java. Details of the implementation are presented in Appendice B1.

3.6.1 Posit Textual Analysis (Word-Level Feature)

This section describes the second text feature representation framework. In the following, the syntactic feature extraction process using Posit textual analytic tool is discussed.

The Posit textual analysis toolset is a program written mainly in UNIX scripts and is capable of generating a detailed frequency-based syntactic analysis of a textual corpus [14]. Recently, Posit was implemented in an integrated full-featured Posit-API version [79]. The Posit-API version provides the full scope of the Posit application in the analysis of text data sets.

When the Posit API was applied to the dataset, it generates quantitative data from any text. The output provides word-level features and associated values. The features include word count, number of characters and sentences, number of token and types, n-gram

frequencies and statistics based upon parts-of-speech (POS) [14]. By default, Posit produces data on 27 features. Figure 3.10 and 3.11 show the procedural role of Posit. The resultant feature set from Posit can be fed into a classifier for Web page classification.

The output from Posit analysis produces three different levels of detail, a summary level, the intermediate (aggregate) part-of-speech analysis and the detailed word types together with the part-of-speech analysis. The summary level includes the total number of verbs, nouns, adverbs, etc. In addition, frequency data is produced in the intermediate level for the contents of the text analysed in terms of particular parts of speech. For example, it generates analysis of different forms of verb such as, the base type of verbs, the gerund, the past tense, the past participle, the 3rd person present, the present tense (non-3rd person) form and the 3 modal auxiliary forms. In the fine detail level, frequency data for each word in terms of part-of-speech type is provided, such as the number of occurrences of every word that are in the past participle form, etc. The three different levels of Posit analysis details are shown in Appendix.

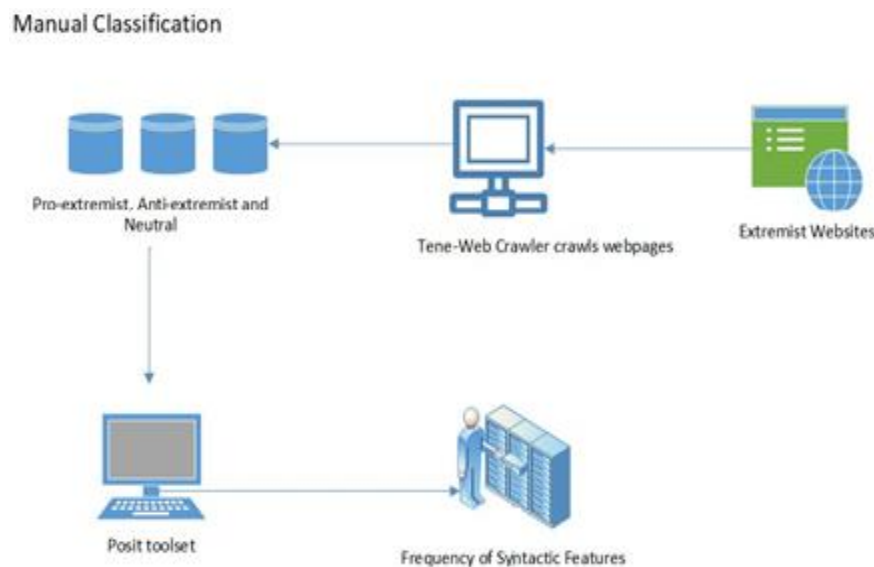


Figure 3.10 Word-level Feature Extraction Process using Posit Analysis

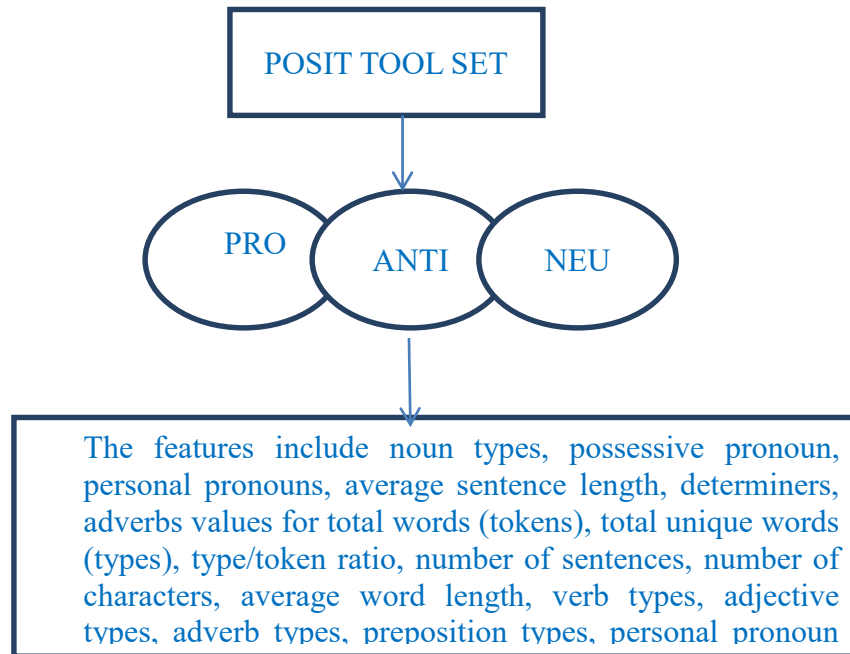


Figure 3.11 Processed Features using Posit

Posit word-level features, together with the manual classification, produced 28 features to be fed into a classifier for Web page classification. On the basis of chosen classifiers and the target manual classification, this provided a measure of how many of the pages were successfully classified on the basis of the Posit features.

3.6.2 Extended-Posit Analysis

For this third text feature representation framework, Posit was improved to include character-level content. This upgraded the conventional word-level statistics to provide an additional 44 character-level features for each instance of text data. The extension of Posit to adopt character-level as well as word-level data preserves the domain-neutral nature of Posit analysis. This extended-Posit technique was implemented on the extremist

Web pages through the API facility, to output both word-level and character-level features. The character-level features contain quantitative information on individual alphanumeric characters, and a subset of special characters, questions marks, exclamation marks, asterisks, periods and dollar signs. Following this analysis, each data item of the extremism Webpage is represented by a set of 72 features – 27 word-level, 44 character-level features and the manual classification. Thereafter, this list of page features comprises 72-Posit features, including the manual classification for a direct entry into a classifier for Webpage classification.

3.7 The Composite Analysis

This section describes the fourth text feature representation framework (the proposed framework).

This framework is designed to utilise the combination of sentiment and syntactic features in textual content as a basis for text features which are fed as input into machine learning algorithms to build a classification system. The proposed composite framework operates through a custom-written Python script that merges together sentiment features derived from a lexical approach in sentiment analysis and the frequency of syntactic word-level features obtained from Posit. The rationale behind the hybrid features in the composite approach is to apply the richer features of the textual corpora that could be fed into the classification model. Both sentiment and syntactic features have proven to be significant and useful input in developing a classification model [5,13,14]. Figure 3.12 below illustrates the composite feature extraction process. The generated output data comprises 54 features including the manual classification for a direct entry into classifiers, this is to generate a measure of how well each page can be classified into their appropriate classes.

.

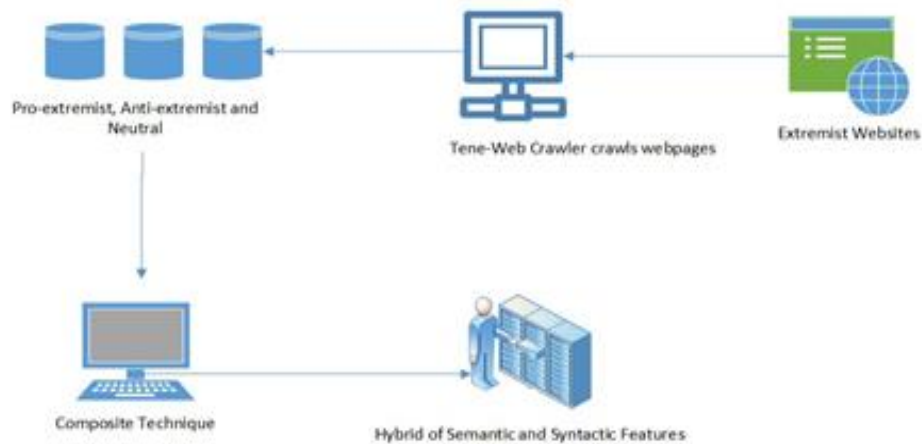


Figure 3.12: Composite Feature Extraction Process

The details of the classification models explored in this thesis are further explained in the next chapters of which include, Machine Learning algorithms (such as, J48, Random Forest (RF) and KNN)) and Neural Networks, (Multilayer Perceptron (MLP) and Recurrent Neural Network (RNN)) which are implemented using both Scikit-learn and TensorFlow API respectively.

Chapter Four: Machine Learning

This chapter describes the implementations of the Machine Learning Models and the results obtained on different classification frameworks explored in this thesis. As a reminder, the classification frameworks in question, are Sentiment (MICE, Missforest and KNN imputation), Posit (features on the basis of word-level information), and the Extended Posit (features on the basis of both word and character-level information) and the proposed framework, Composite-based classification method.

4.1 Machine Learning

Machine learning is a branch of Artificial Intelligence that provides systems the capacity to automatically learn from data and explore knowledge from the experience for future predictions without being programmed for each specific case. An algorithm builds a model using example input and applies the model to make decisions or predictions [20]. The aim is to develop models that learn without any help or human intervention. Machine learning builds algorithms that can learn from data in contrast to static programming that instructs a computer what to do in the case of specific data. In machine learning, the computer derives its model based on the data available. There are two categories of machine learning: supervised learning and unsupervised learning. The supervised learning algorithm learns from a function that converts input to output depending on the sample input-output sets [80]. It deduces a function from pre-classified (labelled) training data comprising a set of training instances. In supervised learning, a pre-classified dataset is involved. Examples of supervised machine learning algorithms are Support Vector Machine, Neural Network, J48, Random Forest algorithm, etc. Unsupervised learning is a form of machine learning that searches for previously unnoticed patterns in a data set without pre-classified labels and with no supervision such as a k-means clustering algorithm, apriori algorithm, etc. [80].

4.2. ALGORITHMS

The classification algorithms employed in this thesis are J48, Random Forest Decision Tree algorithms, and K-Nearest Neighbours. The details are further explained below:

i. J48 Decision Tree

The J48 decision tree is a predictive machine-learning model that creates a classification or regression model in a tree-shaped structure on the attribute values of the available training data with the purpose of classifying a new item [80]. Whenever J48 comes across a training set, it spots the attributes that distinguish various instances distinctly (i.e. the features with the highest information gain) within the available values of these features, if there is no confusion, then that branch is terminated, and the target value obtained is allocated to it. J48 operates by determining the dependent variable, that is, the target value of a new sample using the various attribute values in a given data set. The branches between the nodes of the decision trees show the possible values of the attributes in a given sample; the internal nodes indicate the different attributes and the terminal nodes produce the final value, (i.e. the classification of the dependent variable) [80]. The dependent variable is the attribute to be predicted while other attributes that aid in predicting the worth of the dependent variable are referred to as independent variables in the dataset.

The different types of decision tree include ID3, (CART) and C4.5 [80]. The J48 decision tree algorithm is adopted because it gives a better understanding of how the algorithm makes decisions. In addition, it contains an algorithm that enhances text classification and a rule-building process [80].

ii. K-Nearest Neighbor

One of the most basic machine learning algorithms is the K-Nearest Neighbour algorithm, which is a supervised learning method. The K-NN approach assumes that new

data and current instances are comparable and assigns the new instance to the category that is closest to the existing categories [81]. The K-NN method stores all accessible data and classifies a new data point based on its resemblance to the existing data [81]. Both regression and classification problems can be solved with the K-NN approach, but are commonly used for classification tasks. A kind of non-parametric algorithm is the K-NN algorithm, this implies that it doesn't assume anything about the data. K-NN algorithm is regarded as a weak learner algorithm since it doesn't instinctively learn from the training set; alternatively, it reserves the dataset and during the classification, it acts on the training set [81]. During the training stage, the KNN algorithm simply keeps the information, and when it receives new data, it classifies it into a category that is quite similar to the new data [81].

KNN method starts with determining the number of neighbors; there is no specific way to discover the ideal value for "K," therefore we must fine-tune the parameters to get the best results. It estimates the Euclidean distance between K neighbors. If the input variables are similar in Euclidean, Euclidean is an appropriate distance metric to utilize. Euclidean distance is estimated by using the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input characteristics j.

$$\text{Distance}(x, x_i) = \sqrt{\sum_j (x_j - x_{ij})^2}$$

Thereafter, the approach takes the K nearest neighbors based on the Euclidean distance obtained. Then, compute the number of data points in each class among these k neighbors. The algorithm eventually allocates the new data points to the class with the greatest number of neighbors [81]. KNN algorithm is easy to set up and resistant to noisy training data.

iii. Random Forest Algorithm

This algorithm starts by selecting random samples from a given dataset and then forms a decision tree which the algorithm then separates/ split each class in the tree using their features, a new random sample of features is selected for every single split, and the algorithm only chooses one of the random samples as the prediction. It uses entropy and information gain for calculation [82]. It's also worthy of note that the random forest is a more advanced application of a decision tree, based on its voting and weighting functions between multiple decision trees, which makes it act as a stacked ensemble [82]. It employs ensemble learning, a method for addressing complex problems by merging many classifiers.

4.3 MODEL METRICS

The metrics described in this section were used consistently across all classification models.

4.3.1 Validation

Model validation refers to a process where a trained model is gauged with testing set in machine learning. The testing data set is a subset of data that is not part of the training set. The main function of the testing set is to determine the generalisation power of a training model [83].

Training a model on a subset of data and testing it on the remaining samples is a primary approach to validating a learning model. Dividing the data into two raises a challenge of what proportion should test and training set be chosen. It is substantiated that 80:20 or 70:30 are acceptable ratios [83], while [84] suggested 75:25 as a common choice for some particular classification problems. However, it is necessary for both subsets to be represented well with a sufficient amount of data. Otherwise, the model will not have adequate information about a category, and testing the model using the testing set might not yield a good result [84]. The train set is used to make machines learn the pattern

and create a model for future prediction. The test dataset is used to test model performance such that it considers this data as unseen data. Another better option is to explore cross-validation. When we use cross-validation, even the train set is divided into N partitions to make sure that our model is not overfitting [84].

Cross-validation estimates how the outcome of a statistical analysis will behave on a new data set. The most popular among the cross-validation type is the K-fold Cross-Validation. Other types of cross validation include Leave-One-Out and Leave-p-Out but they are computationally expensive [48]. K-fold Cross-Validation is mostly used in machine learning for a given predictive modelling problem because it is easy to comprehend, and produces a result with a lower bias than other methods [85]. The method has one parameter known as k which describes the number of categories into which a given data sample is to be divided. K-fold Cross-Validation entails randomly splitting the set of samples into k categories, or folds of the same size [85]. The first fold is used as a validation set, and the method is fitted on the remaining $k - 1$ folds. A specific value for k can be chosen, such as $k=5$ or 10 .

4.3.2 Evaluation Metrics

A machine learning algorithm's performance on a dataset could be evaluated using various metrics. Such metrics include root-mean-square or mean absolute error, true positive (TP), false negative (FN), false positive (FP), true negative (TN), accuracy, recall, precision, f-measure, confusion matrix etc. These metrics will be explained in more detail later in this chapter. In addition, there are two cases in classification which include the binary and multi-class categories. Binary classification deals with two definite categories: one positive and one negative towards an objective while the multi-class category deals with classifying instances into one of three or more classes. Many performance measures can be drawn from a confusion matrix. The row of confusion indicates the actual classes while the columns show the predicted classes. One class is

indicated to be a positive class (yes) while the other is the negative class (no) for the other. Table 4.1 below shows a 2x2 matrix.

	Predicted	
	Yes	No
Yes Class	True Positive (TP)	False Negative (FN)
No	False Positive (FP)	True Negative (TN)

Table 4.1: Confusion Matrix 2x2

False negative means an instance predicted to be negative which is positive and false positive vice versa. True Positive is an instance predicted to be yes and the actual outcome is also yes. True Negative is an instance predicted to be no and the actual outcome is also no. The accuracy is defined as the proportion of the number of correct predictions to the total number of input samples.

Accuracy = (True Positive + True Negative) / (True Positive + True Negative + False Positive + False Negative).

Recall reveals a number of true positive entities recognised by the classifier out of all entities identified as positive while precision shows the degree of the accuracy (i.e. The algorithm returns most of the relevant items).

Precision = True Positive / (True Positive + False Positive).

The recall is the ratio of the total amount of relevant occurrences that were retrieved.

Recall = True Positive / (True Positive + False Negative).

In a perfect classification, precision and recall have a value of 1. Specificity evaluates the number of times the negative class is actually classified as negative [86].

$$\text{Specificity} = \text{True Negative} / (\text{True Negative} + \text{False Positive})$$

The F-measure, also known as F1, is defined as the harmonic mean of recall and precision [86].
$$\text{F-measure} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$$

4.4 Machine Learning Set-Up

The feature sets considered in this section are (i) the ‘default’ 27 Posit features, (ii) the extension of Posit to include 44 character features (referred to as extended-Posit features), (iii) sentiment features (KNN, MF and MICE imputation) and Composite features (a mix of sentiment and syntactic features derived from the textual data). All of the features in these different sets were extracted from the three predefined categories of extremist Websites (with 2500 Webpages in each category). Each feature set was employed to determine its degree of effectiveness in classification, via three algorithms, the J48 decision tree, Random Forest and KNN. After the data preprocessing and preparation each machine learning algorithm was implemented on each feature set. In addition, each model explored was tweaked to obtain optimal performance. Again, feature selection techniques such as embedded and wrapper methods were applied to obtain useful features for excellent performance. In this case, J48 and Random Forest algorithms were explored for feature selection operations. Further explanations are detailed below.

4.4.1. Hyper parameter Turning

The task of selecting a set of ideal hyperparameters for a machine learning algorithm is regarded as hyperparameter tuning. A model argument known as a hyperparameter has a

value established before the learning process even begins [87]. Hyperparameter tweaking is the cornerstone of machine learning algorithms [87]. Data is used to learn model parameters, and hyper-parameters are tweaked to achieve the best fit. A decision tree, for example, has hyperparameters such as maximum depth and the minimum number of observations in the leaf, whereas a KNN model has hyperparameters such as weight, n_neighbour and leaf size. Because finding the ideal hyper-parameter can be complex, search algorithms such as grid search and random search are often used. Grid search is a task in the model selection package of Scikit-learn. This method is useful for looping over predetermined hyperparameters and fitting the estimator (model) to the training data, then the best parameters are chosen from the hyperparameters presented [87].

Grid search selects a grid of hyperparameter values and compares them all. The min and max values for each hyperparameter must be specified by guesswork (an assumption gauged as a result of the behaviour of the initially unturned baseline model) [87]. Having examined its behaviour, one could be informed on what feasible range the model could begin to better generalise. RandomsearchCV is efficient and saves a lot of time but this thesis, on the other hand, adopted GridSearchCV. GridSearchCV is preferred over RandomSearchCV because it ensures the best model results within the test values by testing each and every one of the variables supplied, as opposed to Randomized Search, which chooses combinations at random. GridsearchCV's processing time increases as the number of combinations increases. As a result, only a few hyperparameters were employed in the GridsearchCV implementation in order to improve processing speed. Each of the model's parameters is explained in Table 4.2 below.

Model	HyperParameters	Range
J48 Decision Tree	Criterion	entropy
	max_depth	1 – 21
	min_samples_leaf	2-5

Random Forest	Criterion	gini,entropy
	max_depth	1 – 21
	min_samples_leaf	2-5
KNN	Leaf_size	1- 10
	'n_neighbors	1, 3, 5, 7, 9, 11, 13
	Weight	Uniform

Table 4.2: The Model Parameters

Hyperparameters regulate the model's over-fitting and under-fitting. Different datasets have different optimal hyperparameters. The following steps are taken to obtain the best hyperparameters:

- We pick the models to be used; we check the model's parameters; we select the techniques for searching the hyperparameter; and finally, we instantiate the GridSearchCV method.
- An evaluation criterion for scoring the model is defined. Here we made use of the accuracy score
- The concept of nested cross validation was used. 5-fold cross-validation was used for both the outer and inner loops.
- Fit the search to the data (X train and y train) and run it.

GridSearch CV uses the Cross-Validation method to test all possible combinations of the values supplied in the dictionary and assesses the model for each one. As a result, after employing this function, we can obtain the accuracy for any combination of hyperparameters and select the one that performs the best.

Decision Tree Algorithms (J48 and Random Forest) Parameters

The criterion

The **criterion** for measuring or evaluating the quality of each decision tree split is the ‘entropy’. The entropy simply means that if a sample (row) is randomly selected from a split, what is the possibility that it would be incorrect [88]. The goal of the entropy is to ensure that there is near 1 entropy (i.e. near-balance of each class so that if a certain target variable class is picked, the probability of selecting the right class and ensuring the purity of that split is high).

A minimum sample leaf

A minimum sample leaf specifies the minimum number of samples that must be present at a leaf node before a split occurs [88]. The least number of samples necessary to be at a leaf node is denoted by the minimum sample leaf. In the minimum sample leaf parameter, a split point at any depth will only be evaluated if it leaves at least the minimal amount of samples in each of the left and right branches for training samples [88].

Max_depth

This parameter denotes the maximum depth of the tree the model should support. It accepts an integer as a parameter but defaults to none, which enables the nodes to increase until all leaves are pure or contain less than the minimum number of samples required [88].

K-Nearest Neighbor Algorithms Parameters

The parameters explored are leaf_size, 'n_neighbors and weight. The leaf size is a parameter in a KD tree or KD ball tree algorithm in KNN that helps to partition, allocate or organize data points in a multi-dimension (multi-feature) space by calculating the

distances between each data point. So, the larger the leaf_size, the slower the classification of these data points and vice versa [89]. The n_neighbors are the total number of data points closest to a selected data point [89]. The weight function is utilised in predicting likely values. It is set to a default value uniform, this enables all points in each neighborhood to be weighted equally [89].

4.4.2 Feature Selection

To increase classifier accuracy and save runtime in high-dimensional datasets, features must be reduced to an acceptable subset [48]. Choosing a subset of important features for use in model creation, feature selection improves accuracy and run-time. The fundamental goal of feature selection is to improve predictors. Filters, wrappers, and embedding methods are the three types of feature selection methods. Wrappers and embedded methods are explored in this thesis because the filter method only provides a ranking of relevant features using univariate statistics and no training is involved for filter method while the wrapper and embedded method provide subset of feature after training [90]. The two methods explored are discussed below.

Wrapper Method:

Feature selection can impact a machine learning model's performance by defining a significant feature subset for increasing the performance and identifying the variability [90]. The wrapper method evaluates the "usefulness" of features subject to the performance of the classifier [90]. The wrapper techniques evaluate a set of features using a machine learning algorithm that uses a systematic review to scan over the range of possible feature subsets, rating each subset based on the strength of the algorithm's performance. This algorithm is known as a greedy algorithm because it tries to discover the finest feasible combination of features that results in the best performance model [90]. Examples of the Wrapper method include Recursive Feature Elimination, sequential feature selection algorithms, forward and backward elimination passes, best-first search, etc. Recursive Feature Elimination would be explored in this thesis and its description is detailed below.

Recursive Feature Elimination (RFE): This algorithm is effective for determining which features in a training set are essential in predicting the target variable. RFE generates a rating of features as well as candidate subsets, as well as the related accuracy. The subsets with the most accuracy are often used as the final subset [91]. RFE works using the supplied machine learning method, prioritizing features by relevance, deleting the least important features, and fitting the model again. This procedure is done until only a certain amount of features are left [91].

Embedded method: Embedded method investigates the connection of features in the same way that wrapper methods do. It built the search for the best subset of features into the classifier construction [92]. To begin, this technique is used to train a machine learning model and then use it to calculate feature importance, a measure of how relevant a feature is when generating a prediction. Finally, it uses the derived feature importance to delete non-important characteristics [92]. The relevance features tells us which factors are more significant in predicting the target class accurately. The difference between embedded and wrapper approaches is that during learning, an internal model building metric is applied [92].

4.5 Classification Result

After executing all experiments and the analysis on the available feature sets, this section presents the result of the experiments for each framework. Thereby, a clear position of each research question can be considered. The results interpreted in this section are the performance of the overall Webpage classification; pro-extremist, anti-extremist and neutral class. Precision, recall, f-measure, and accuracy are the metrics used for the performance evaluation of the model, in the graph presented, y-axis is the f1-score and the x-axis is the parameter value and 5-fold cross validation was explored to provide a degree of validation.

4.5.1 Random Forest Classification Result

Each feature set from Sentiment, Posit, Extended-Posit and Composite analysis were deployed into Scitlean API, where Random Forest was applied with GridsearchCV for optimum performance. The process aims to generate measures that show how the system assigns each page to its appropriate classes. Table 4.3 below described the parameters used in the Random Forest model. The details of each classification framework results are detailed below:

	Parameter	Parameter Values
Random Forest	Criterion	Entropy
	max_depth	1 – 21
	min_samples_leaf	2-5

Table 4.3: The Random Forest Model Parameters

4.5.2 Sentiment-Based Framework using 27 features (MF Imputation)

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation. The details are shown in Table 4.4.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	19	2
2	Entropy	17	2
3	Entropy	19	2
4	Entropy	18	2
5	Entropy	18	2

Table 4.4: Grid Search Best Parameters for MF Imputation

From the above parameters, we can see that our sentiment analysis framework (MF imputation) demands a maximum depth not less than 19 in order to attain the best performance. Max depth is simply the longest path that each tree has from its root right down to its last leaf. A minimum sample leaf specifies the minimum number of samples that must be present at a leaf node before a split occurs. A min_sample_leaf as low as 2 means that we have simpler and less complicated tree structures within the forest, trees comprising of just 2 branches before arriving at a decision hence, lesser computation time and lesser hardware utilization are achieved.

Below is the graph showing the performance of all the 5 folds in relationship to its maximum depth in Figure 4.1. All the training scores for each fold converged at above 80%, and all the cross-validation scores likewise also did converge above the 80% mark but a notable gap could be noticed between all the training scores and their cross-validation counterparts, but the overfitting is minimal in the model. From above Figure 4.1, we can clearly see that as the max depth of the decision tree increases, the performance of the model over the training set increases continuously. From the graph, the y-axis is the f1-score and the x-axis is the parameter value. Fold 5 finished highest in overall accuracy. MF Imputation model gave 86% of extremist Webpages overall classification. Coming down to the final evaluation in the confusion matrix in Figure 4.2, the pro-extremist class had the highest cases at 94%. The results of other categories are shown in Table 4.5.

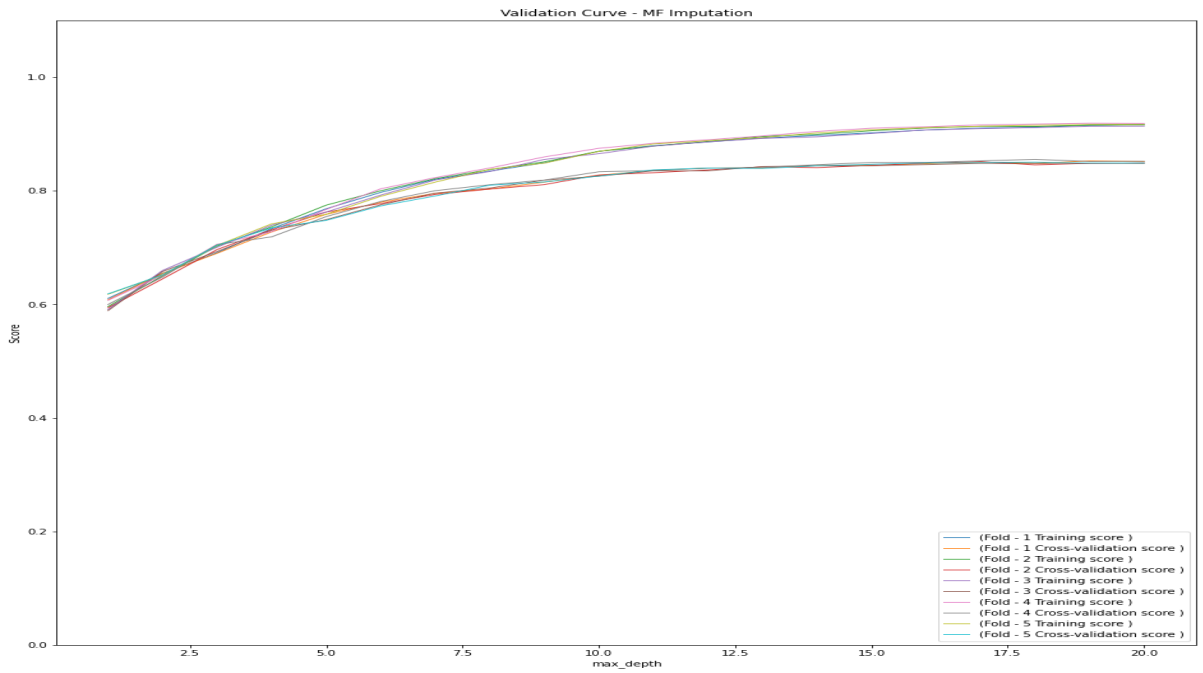


Figure 4.1: Validation Curve-MF Imputation

FP Rate	Precision	Recall	F-score	
0.056	0.894	0.946	0.92	Pro-extremist
0.113	0.8	0.903	0.848	Anti-extremist
0.036	0.912	0.741	0.818	Neutral

Table 4.5: MF Classification Result using Random Forest

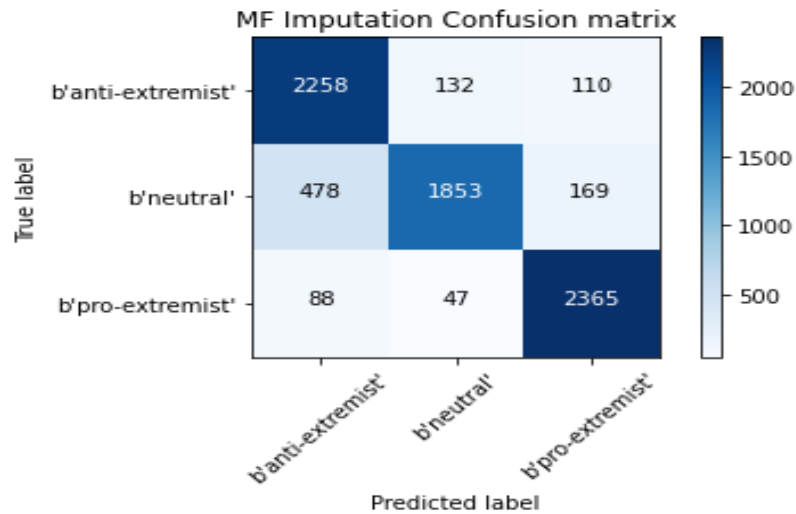


Figure 4.2: MF Imputation Confusion Matrix

4.5.3 Sentiment-Based Framework using 27 features (KNN Imputation)

Below are the best parameters after grid search on 5 folds within the KNN imputation data and the details are described in Table 4.6.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	20	2
2	Entropy	17	2
3	Entropy	20	2
4	Entropy	20	2
5	Entropy	19	2

Table 4.6: Gridsearch Best Parameters for KNN Imputation

For the KNN data, the above parameters showed that the model required a 20 max depth levels to accomplish the best performance, with a constant minimum sample leaf of 2.

The variance between the training and validation scores in KNN imputation is lesser as compared to MF as shown in Figure 4.3. The classification model in KNN imputation is well fitted than what is obtainable in MF imputation but the accuracy is lower than the MF imputation. The model produced an overall classification of 85%. In KNN imputation, Figure 4.4 displayed the confusion matrix and Table 4.7 detailed the classification results of the three categories. From the confusion matrix, the pro-extremist category was the most correctly identified case at 87%.

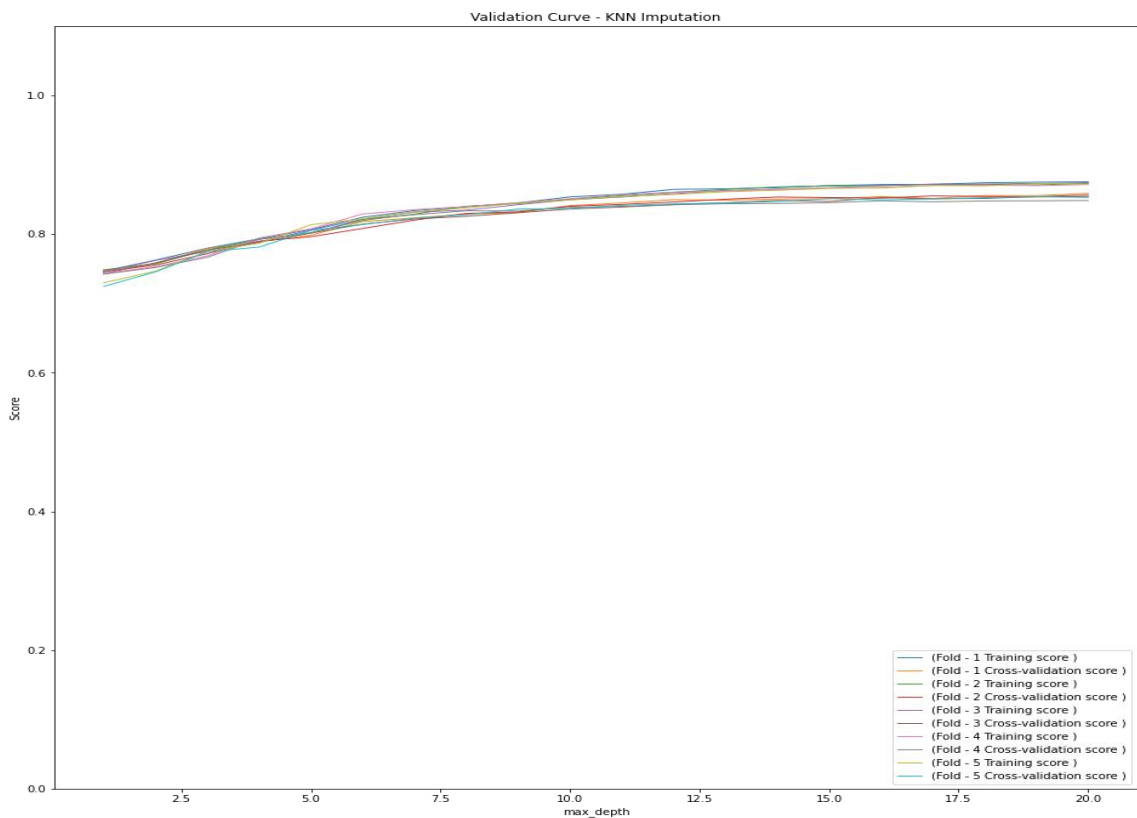


Figure 4.3: Validation Curve-KNN Imputation

FP Rate	Precision	Recall	F-score	Class
0.077	0.849	0.872	0.86	Pro-extremist
0.048	0.898	0.842	0.87	Anti-extremist
0.09	0.826	0.856	0.84	Neutral

Table 4.7: KNN Classification Result using Random Forest

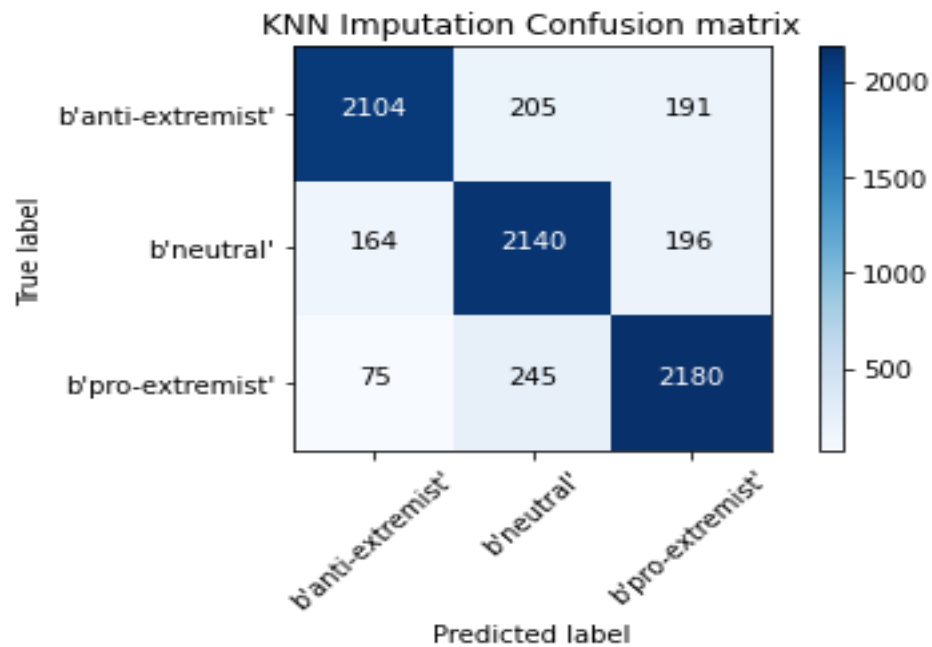


Figure 4.4: KNN Imputation Confusion Matrix

4.5.4 Sentiment-Based Framework using 27 features (Mice Imputation)

Below are the best parameters after grid search on 5 folds within the MICE imputation data and the details are shown in Table 4.8.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	20	2
2	Entropy	18	2
3	Entropy	18	2
4	Entropy	19	2
5	Entropy	20	2

Table 4.8: Gridsearch Best Parameters for MICE Imputation

The best parameters for max_depth are still between 18-20, with fold 1 and fold 5 having the best curve as they not only turned out to be the highest but the both(training curves) converged at the end of the max_depth iterations. Still, we can also notice the same positive relationship between the score and the max_depth meaning that as its value is increased, the accuracy increased. Nonetheless, in the validation curve in Figure 4.5, the variance (difference in both the validation and training set scores) is smaller compared to MF Imputation but a little bigger than the KNN. None of the training or validation sets decreases rapidly and hence the model is a well-fit model. Random Forest gave 88% in classifying overall Webpages into their respective classes.

The MICE imputation data has a much higher overall accuracy compared to MF and KNN imputation. The anti-extremist group still maintained a higher accuracy, F1-score and precision just like in the KNN imputation. The pro-extremist class also maintained the highest recall score as well. For precision, this means that for the MICE data, the random forest easily and more accurately predicts positively the anti-extremist group much more than the other classes. While for recall, in the MICE data, the model is more sensitive to the pro-extremist class. But coming down to the F1 score which is a

combination of both recall and precision, we can see that the anti-extremist group still has the best performance in the MICE data. This can be seen in the classification result and confusion matrix in Table 4.9 and Figure 4.6 respectively.

FP Rate	Precision	Recall	F-score	Class
0.069	0.864	0.881	0.87	Pro-extremist
0.037	0.921	0.877	0.89	Anti-extremist
0.075	0.855	0.879	0.86	Neutral

Table 4.9: Mice Imputation Classification Result using Random Forest

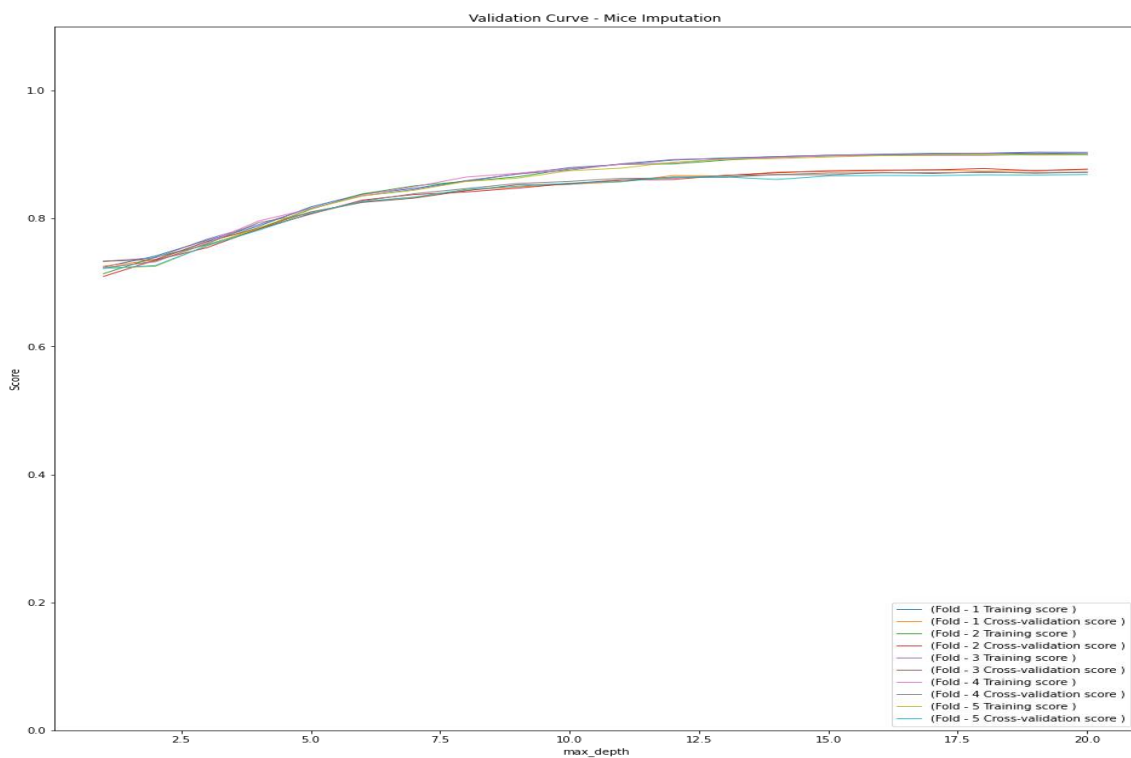


Figure 4.5: Validation Curve-MICE Imputation

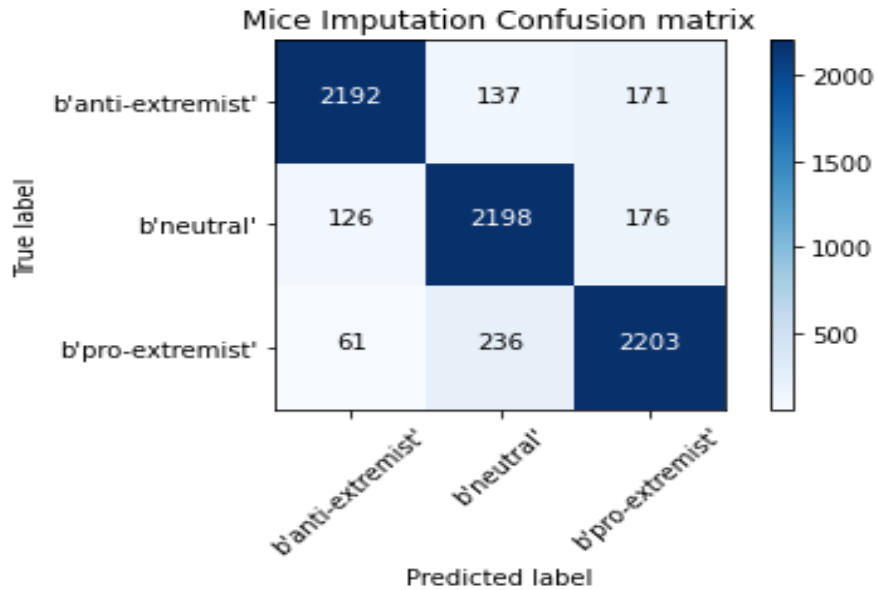


Figure 4.6: Mice Imputation Confusion Matrix

4.5.5 Feature Selection: Mice Imputation Features

For the MICE Imputation, the wrapper method was applied and we noticed that as the features were applied in measured percentages through time, we noticed that the performance of the model improved as the features were added meaning and the best performance so far was achieved when the whole features were applied. Hence, this means that every feature has its own level of importance within the data and must be included to get the best performance of the model. The best accuracy is 88.2% obtained at the 100 percentile of the feature subset at runtime of 3.2sec. Below is the graphic representation of the wrapper method in Figure 4.7. The embedded method was also applied and this time, it performed poorly compared to the wrapper method. So, it gave 79.8% at the 75th percentile of the subset of relevant features but reduced to 79.4% when 100% of the features subsets were applied. See the graphic representation below in Figure 4.8

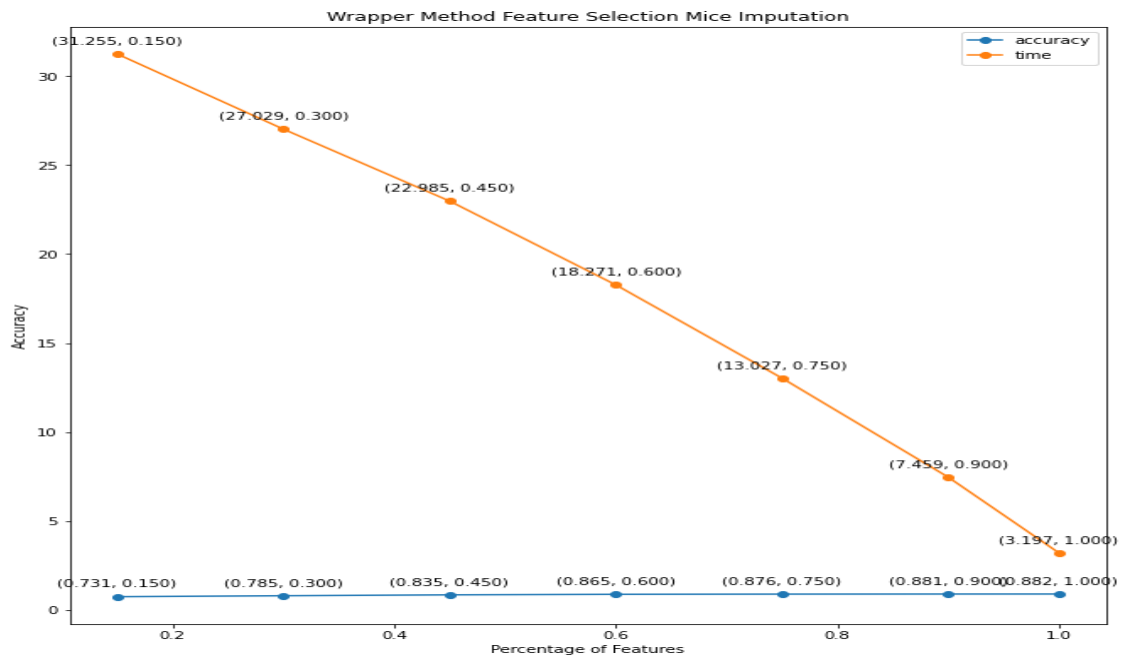


Figure 4.7: The Wrapper Method in MICE Imputation using Random Forest

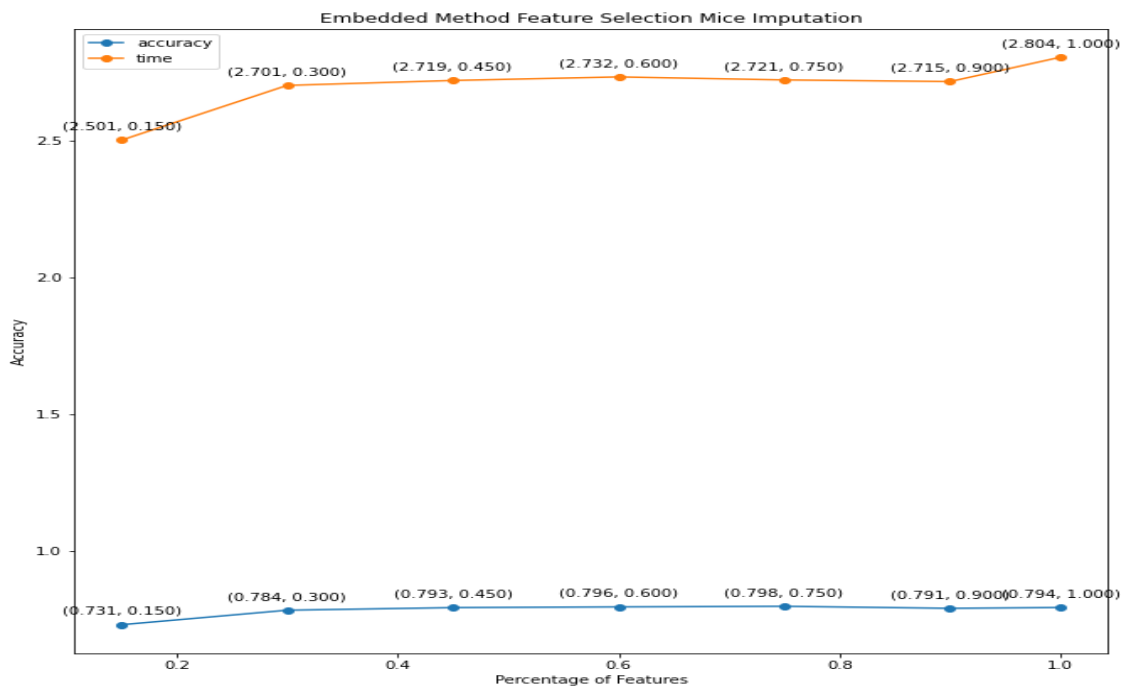


Figure 4.8: The Embedded Method in MICE Imputation using Random Forest

4.5.6 Posit-Based Classification Framework

Below are the best parameters after grid search on 5 folds within the Posit data and the details are shown in Table 4.10.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	16	2
2	Entropy	15	2
3	Entropy	18	2
4	Entropy	15	2
5	Entropy	20	2

Table 4.10: Gridsearch Best Parameters for Posit Data

The grid search results for Posit indicated that the model required a range of 15-20 max depth levels in order to accomplish best performance, with a constant minimum sample leaf of 2. Figure 4.9 shows the training and validation curves. The model's over-fitness is minimal as the variance in both training and the validation scores is low. Random Forest gave overall accuracy of 93%. From the confusion matrix in Figure 4.10, pro-extremist cases were mostly identified with a higher precision rate compared with other categories at the rate of 94% and 96% respectively. Table 4.11 details the classification results.

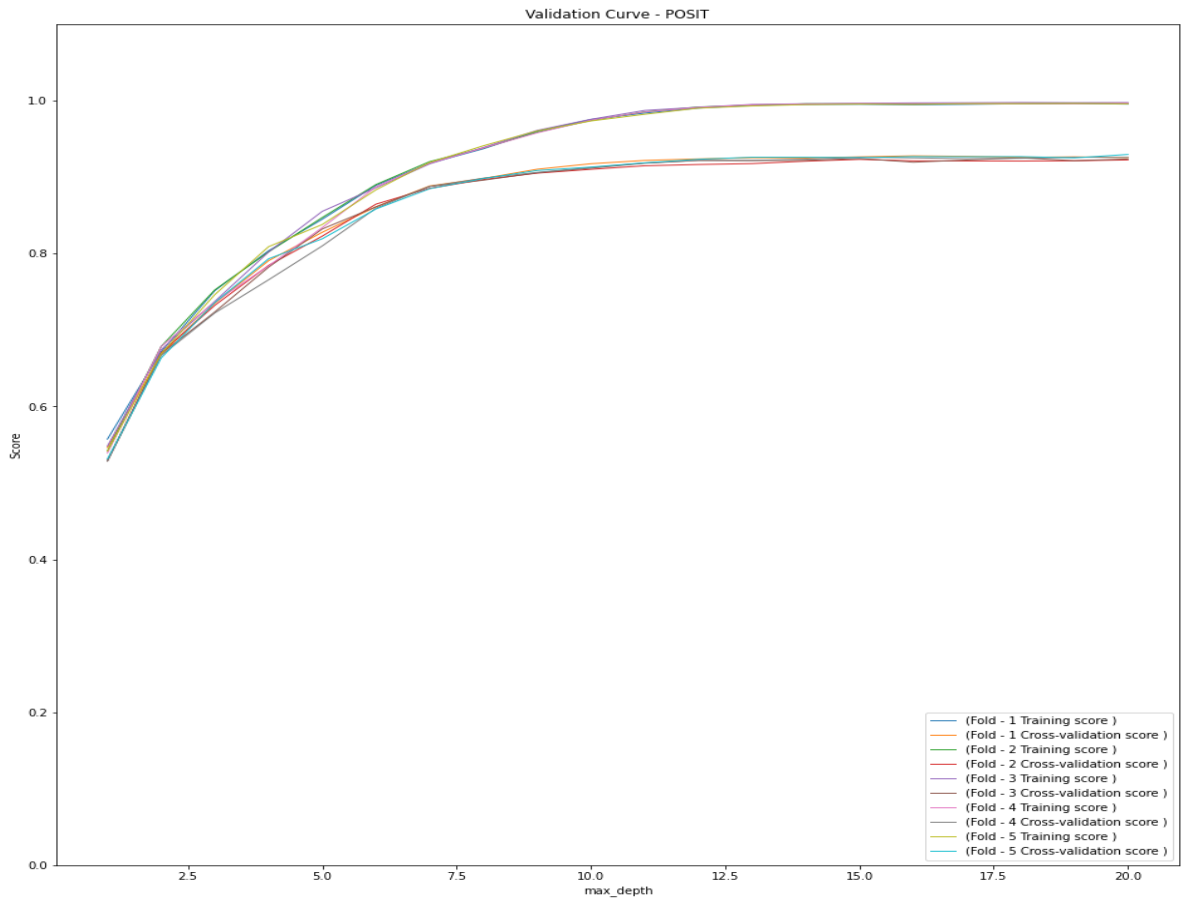


Figure 4.9: Validation Curve- Posit

FP Rate	Precision	Recall	F-score	Class
0.018	0.96	0.94	0.95	Pro-extremist
0.05	0.90	0.91	0.92	Anti-extremist
0.03	0.938	0.94	0.92	Neutral

Table 4.11: Posit Classification Result using Random Forest

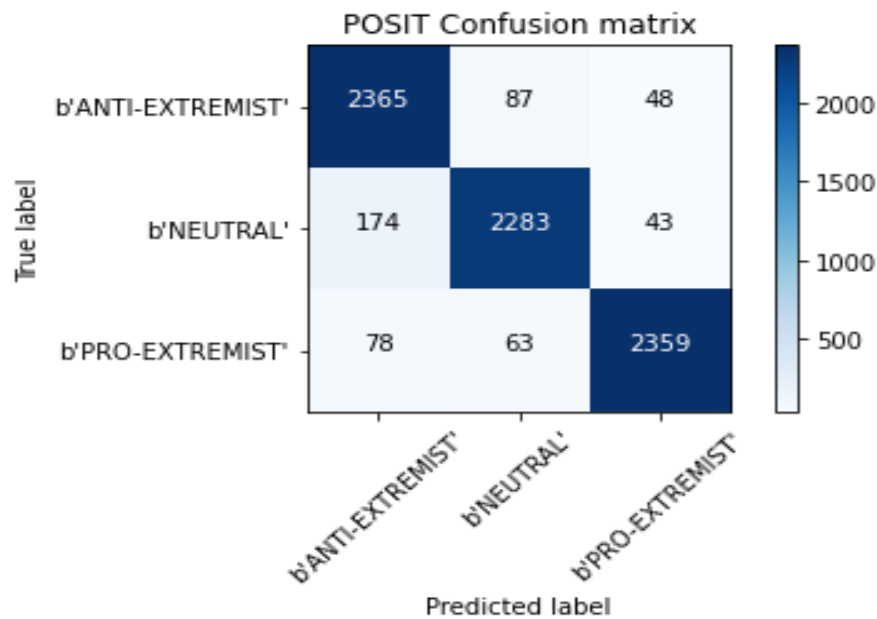


Figure 4.10: Posit Confusion Matrix

4.5.7 Feature Selection: Posit Features

As part of the model construction, wrapper and embedded are the two different filter algorithms employed for the subset evaluation of standard default 27-Posit features. The results from the model indicated that each feature of Posit data has its own level of importance within the data to attain better classification in the wrapper method. The wrapper method gave the best accuracy, at 93.9% when 100 percentile of the feature subset were utilized. The algorithm attained a processing speed of 9.6sec against the embedded method that gave 93.3% at 8.8sec. The embedded method produced its optimum performance at the best 45 percentile of the features. Below is the graphic representation of the wrapper method and embedded method in Figures 4.11 and 4.12 respectively.

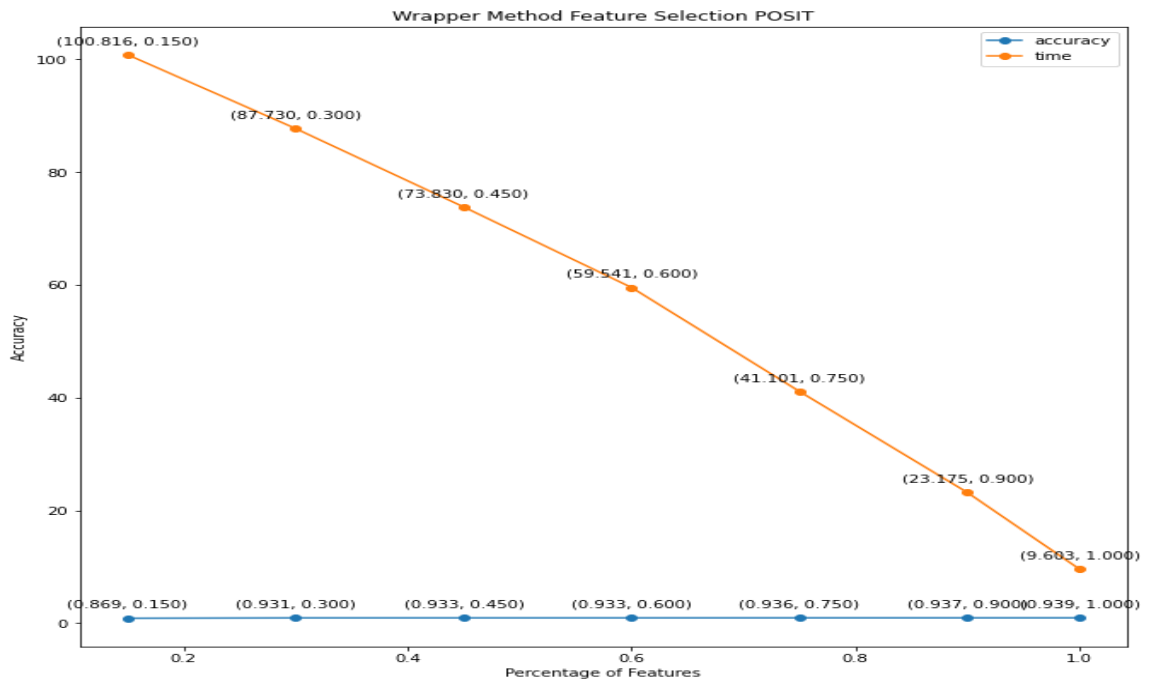


Figure 4.11: The Wrapper Method in Posit using Random Forest

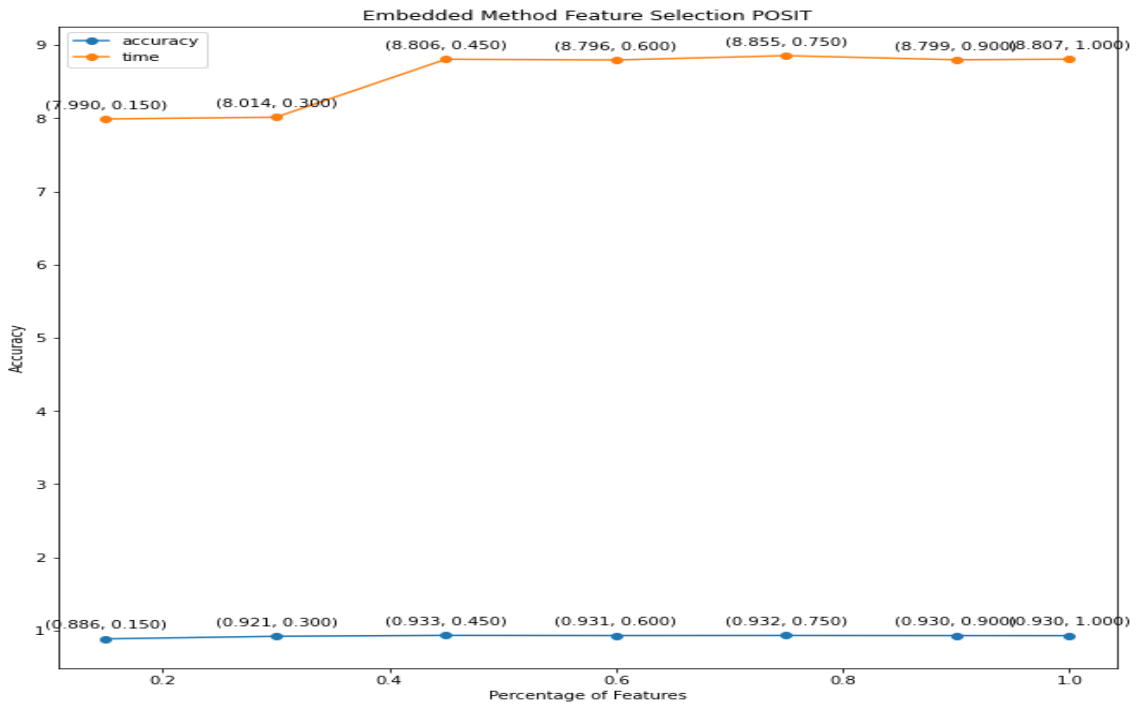


Figure 4.12: The Embedded Method in Posit using Random Forest

4.5.8 Extended-Posit Feature Set (71 features) Classification Framework

Below are the best parameters after grid search on 5 folds within the Extended-Posit data and the details are shown in Table 4.12.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	18	2
2	Entropy	19	2
3	Entropy	17	2
4	Entropy	19	2
5	Entropy	19	2

Table 4.12: Gridsearch Best Parameters for Extended-Posit Data

For the Extended-Posit data, the above parameters showed that the model required a range of 17-19 max depth levels in order to accomplish the best performance, with a constant minimum sample leaf of 2. Random Forest classified the total extremist Webpages at the rate of 95%. This is an improved result over what was obtainable in Posit. The pro-extremist category had the most identified cases and highest precision when compared with other categories. This is an indication that the framework is effective in discerning pro-extremist category. Tables 4.13 and Figure 4.14 detail the results. The model is healthy with a minimal overfitting rate considering the variance between both training and validation scores. The validation curve is displayed in Figure 4.13

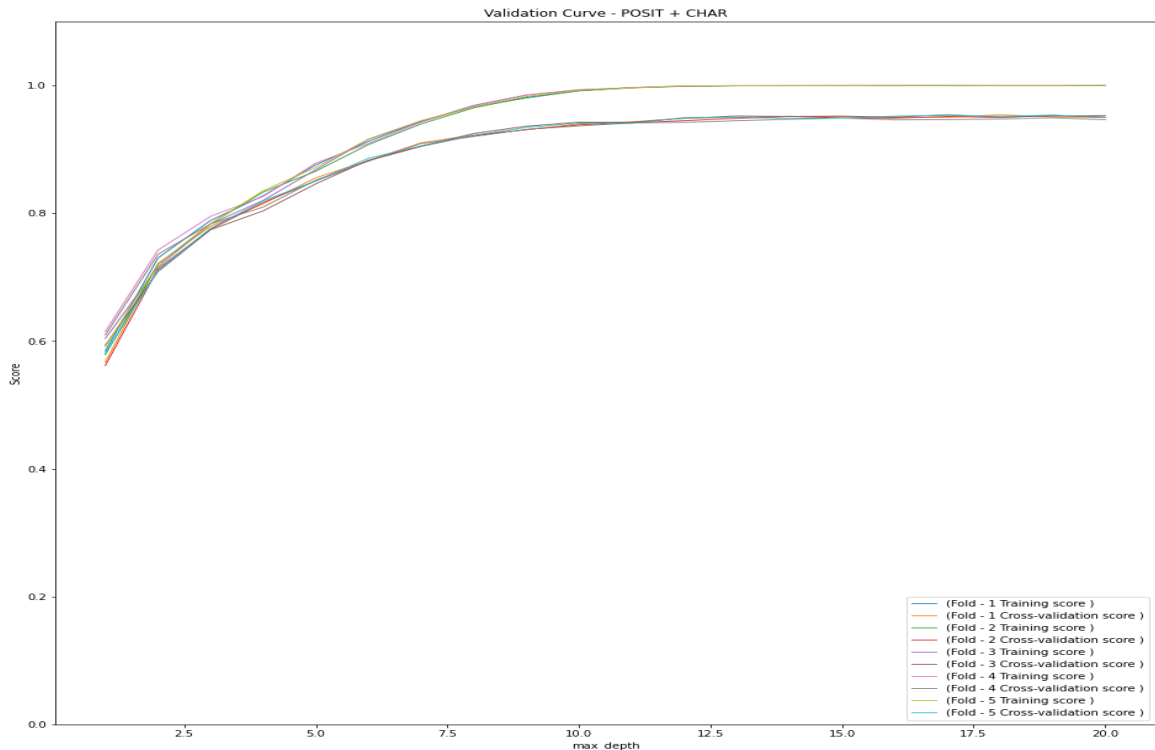


Figure 4.13: Validation Curve-Extended Posit

FP Rate	Precision	Recall	F-score	Class
0.007	0.98	0.97	0.97	Pro-extremist
0.041	0.92	0.97	0.94	Anti-extremist
0.018	0.96	0.92	0.94	Neutral

Table 4.13: Extended- Posit Classification Result using Random Forest

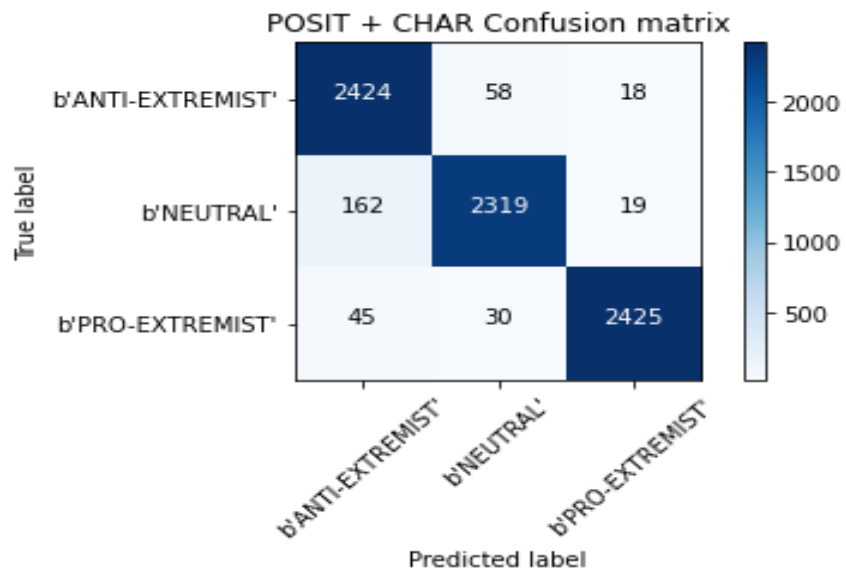


Figure 4.14: Extended Posit Confusion Matrix

4.5.9 Feature Selection: Extended-Posit Features

For the extended-Posit, both the wrapper and embedded methods were applied to 71 extended-Posit features. As the features were applied in measured percentages through time in the wrapper method, we noticed the model improved best when the 75 percentile of best features were applied. The method gave a 96.1% degree of accuracy. The execution time was 165sec. Below is the graphic representation of the wrapper method in Figure 4.15. The embedded feature selection was also applied, it underperformed when compared to the wrapper method. So, it had 95.4% at the 75th percentile of the features but reduced to 95.1% when 100% of the features were applied. The execution time was 14.5sec. See the graphic representation below in Figure 4.16.

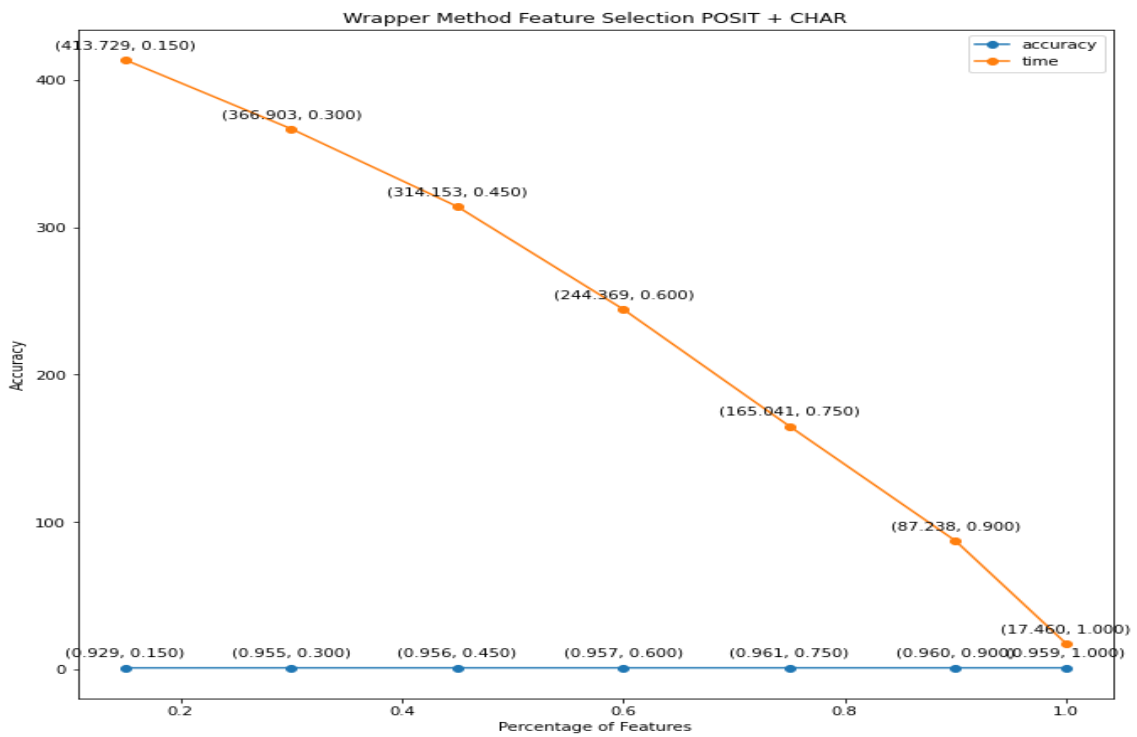


Figure 4.15: The Wrapper Method in Extended-Posit using Random Forest

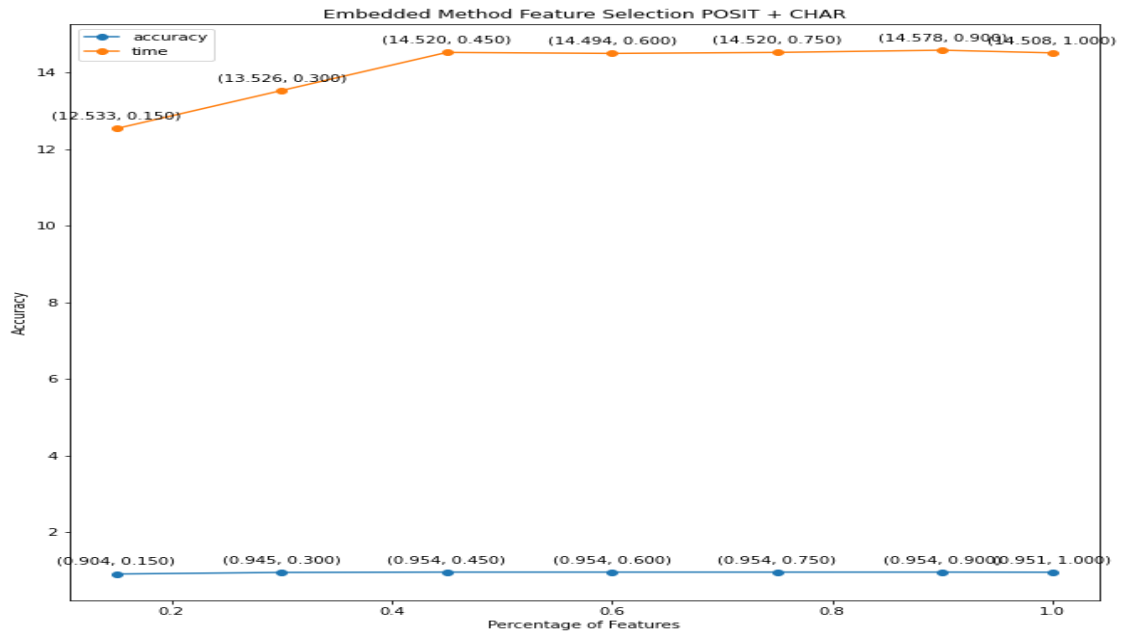


Figure 4.16: The Embedded Method in Extended-Posit using Random Forest

4.5.10 Composite Classification Framework

Below are the best parameters after grid search on 5 folds within the Composite data and the details are presented in Table 4.14.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	19	2
2	Entropy	16	2
3	Entropy	18	2
4	Entropy	18	2
5	Entropy	17	2

Table 4.14: Gridsearch Best Parameters for Composite Data

The grid search results for Composite indicated that the model required a range of 16-19 max depth levels in order to accomplish the best performance, with a constant minimum

sample leaf of 2. Random Forest classified the total extremist Webpages at the rate of 95.8%. This is an improved result over what was obtainable in all frameworks. The pro-extremist category had the most identified cases and highest precision when compared with other categories and all frameworks. This is an indication that the framework is the most effective in classifying the pro-extremist category. Table 4.15 and the confusion matrix in Figure 4.18 detail the results. The model has a minimal overfitting rate considering the variance between both training and validation scores. The validation curve is displayed in Figure 4.17.

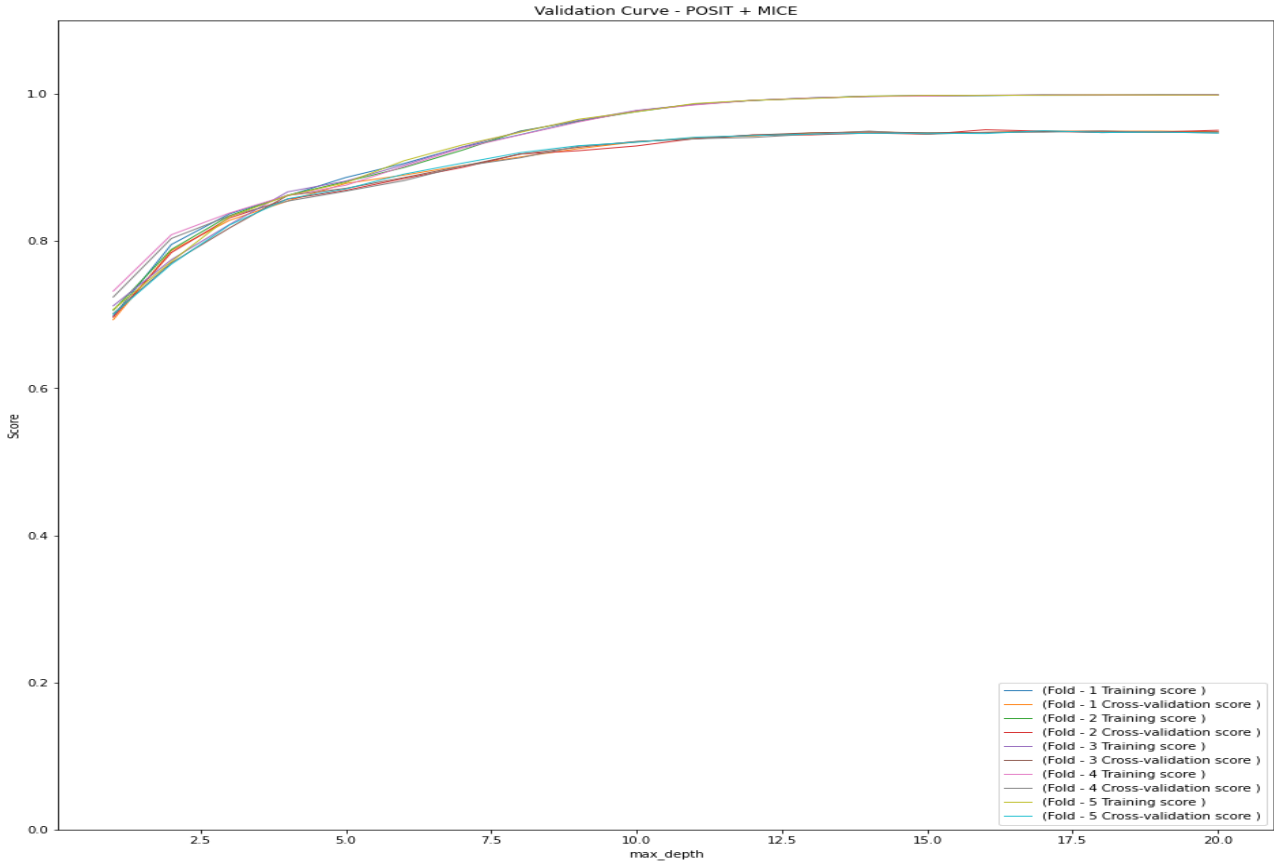


Figure 4.17: Validation Curve- Composite

FP Rate	Precision	Recall	F-score	Class
0.011	0.97	0.97	0.97	Pro-extremist
0.032	0.937	0.94	0.95	Anti-extremist
0.02	0.96	0.96	0.95	Neutral

Table 4.15: Composite-Based Classification Result using Random Forest

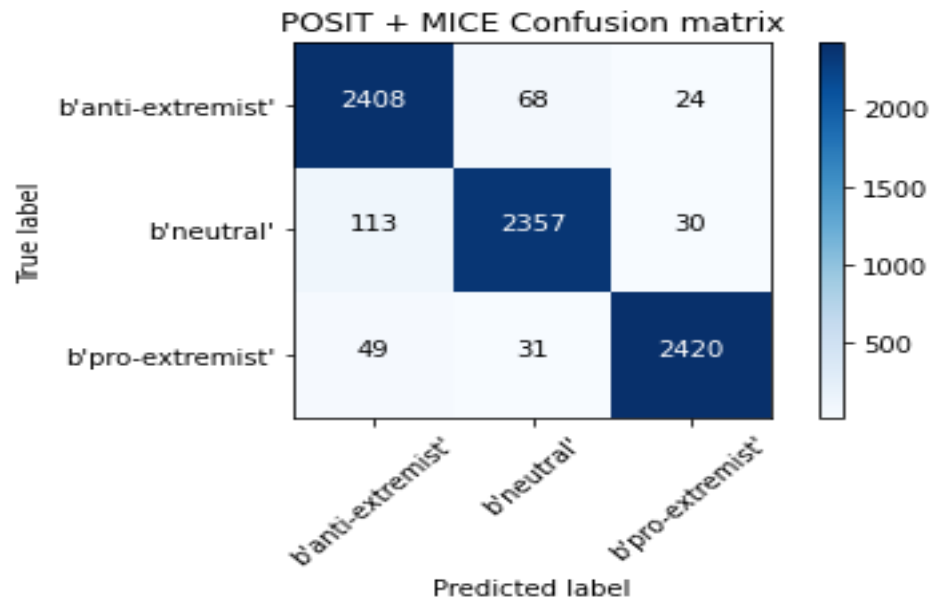


Figure 4.18: Composite Confusion Matrix

4.5.11 Feature Selection: Composite Features

The feature selection algorithms explored are embedded and wrapper algorithms. The wrapper method outperformed the embedded method in terms of classification accuracy

but the embedded did well better in terms of processing speed as both almost arrived at almost the same accuracy. The result from the wrapper method indicated that the model yielded significant performance in terms of classification accuracy, at 95.9% when all the subset features were used with a processing time of 9.08sec. Optimum performance is noticed at the 45 percentile of the subset features to produce 95.1% at 8.76sec speed time. Both embedded and wrapper method results are detailed in Figures 4.19 and 4.20 respectively.

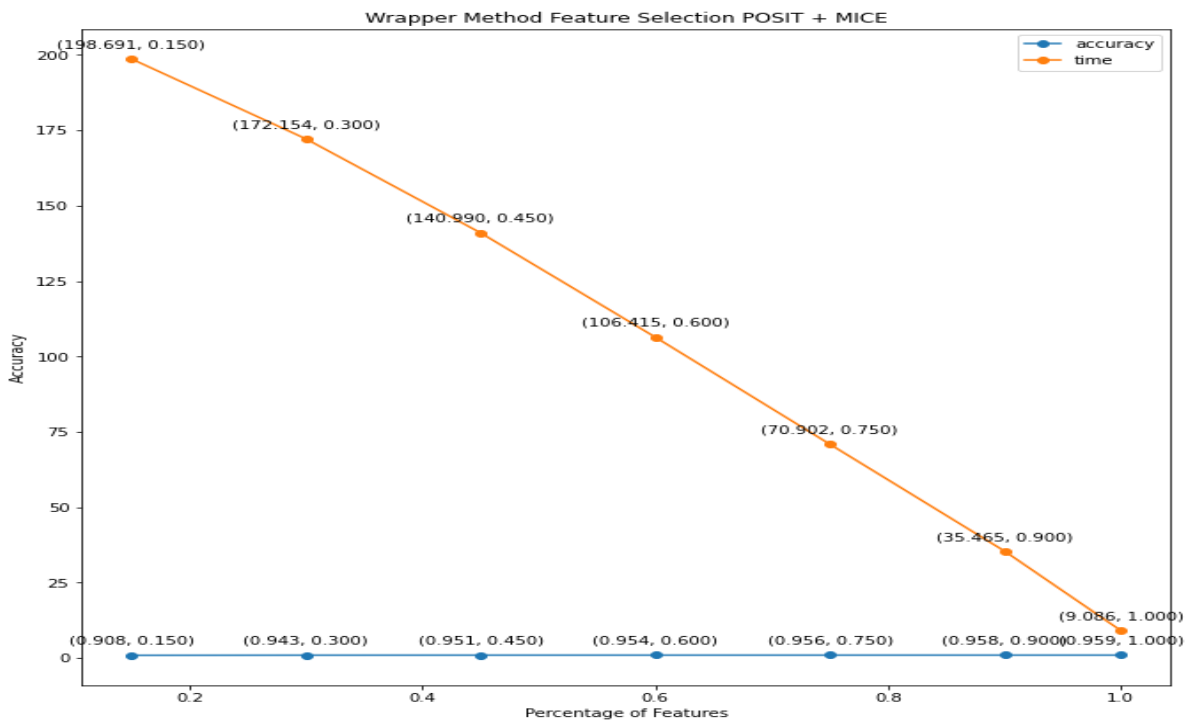


Figure 4.19: The Wrapper Method in Composite using Random Forest

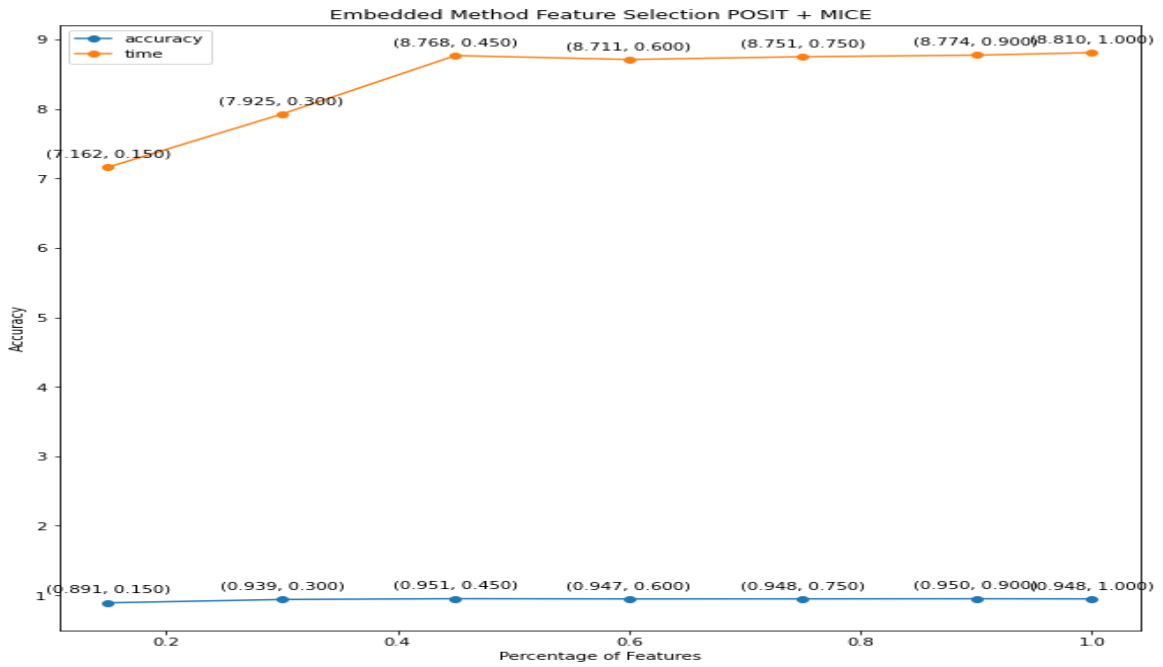


Figure 4.20: The Embedded Method in Composite using Random Forest

4.6 J48 Decision Tree Classification Results

Again, each feature set from Sentiment, Posit, Extended-Posit and Composite analysis were deployed into Scitlean API, where the J48 decision tree algorithm was applied with GridsearchCV for optimum performance. We implemented our classifier, the J48 decision tree algorithm to create a rule-building process for the automated classification system. The aim of this process is to generate measures that show how the system assigns each page to its appropriate classes. Table 4.16 below presented the parameters used in the J48 model. The details of each classification framework results are detailed below:

	Parameter	Parameter Values
Random Forest	Criterion	Entropy
	max_depth	1 – 21
	min_samples_leaf	2-5

Table 4.16: The J48 Model Parameters

4.6.1 Sentiment-Based Framework using 27 features (MF Imputation)

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for MF in Table 4.17.

Folds	criterion	max_depth'	min_samples_leaf
1	entropy	19	2
2	entropy	17	3
3	entropy	19	3
4	entropy	20	2
5	entropy	20	2

Table 4.17: Gridsearch Best Parameters for MF Imputation

The max depth for each of the 5 folds is between 17-20, and the min_sample_leaf has a value of 2 and 3 throughout the fold. The implication is that it takes just a minimum of 2 samples for a leaf node to split. Though small, it's usually preferable for a small dataset. An average Max depth of 19 means that the decision tree model took up to 19 splits to achieve best the performance and it was at the 19th split that the final class classification decision was taken. Hence, it took the model 19 splits to perfectly distinguish each class in pure splits with low impurity. So, in general, the hyperparameter demands a quite low

for an overall accuracy score of 83%. . Pro-extremist category is the most identified case when compared with other categories at the rate of 92.3%. The results of other categories are presented in Table 4.18 and Figure 4.22 respectively. The data plot of both the validation and training scores in each of the five folds in respect to their max depth is presented in Figure 4.21.

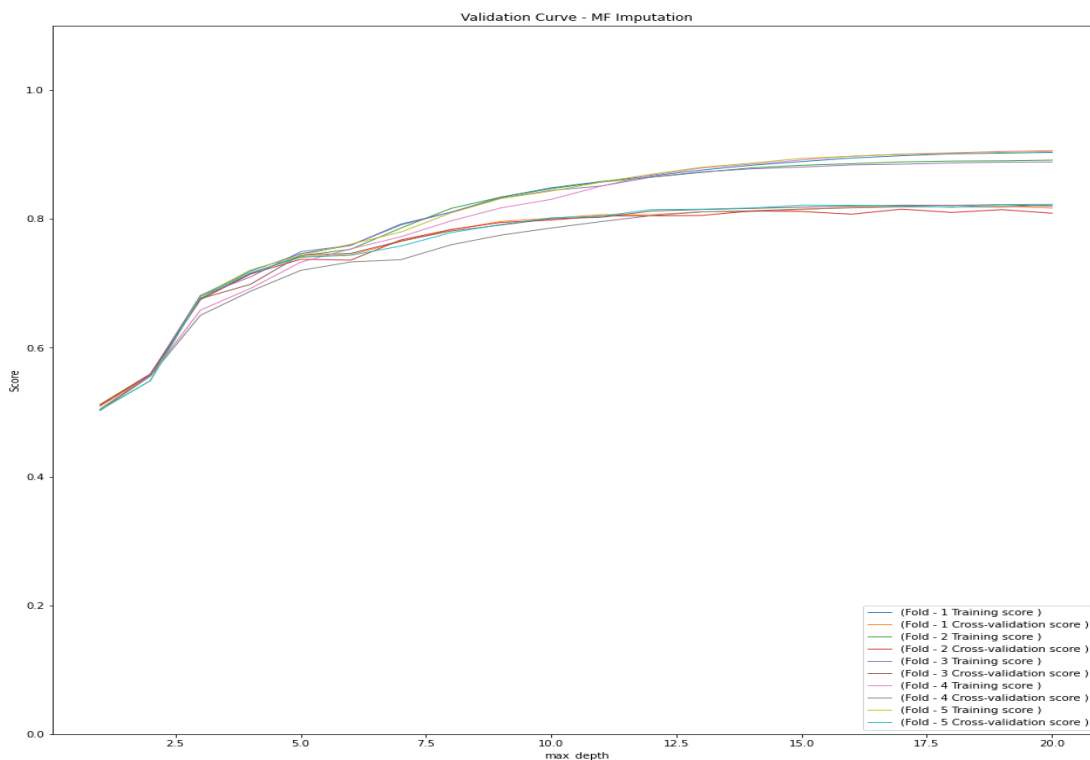


Figure 4.21: Validation Curve-MF Imputation

False Positives Rate	Recall	Precision	F1	Class
0.127	0.842	0.768	0.804	Anti-Extremist
0.060	0.729	0.858	0.788	Neutral
0.066	0.923	0.875	0.898	Pro-Extremist

Table 4.18: MF Imputation Classification Result

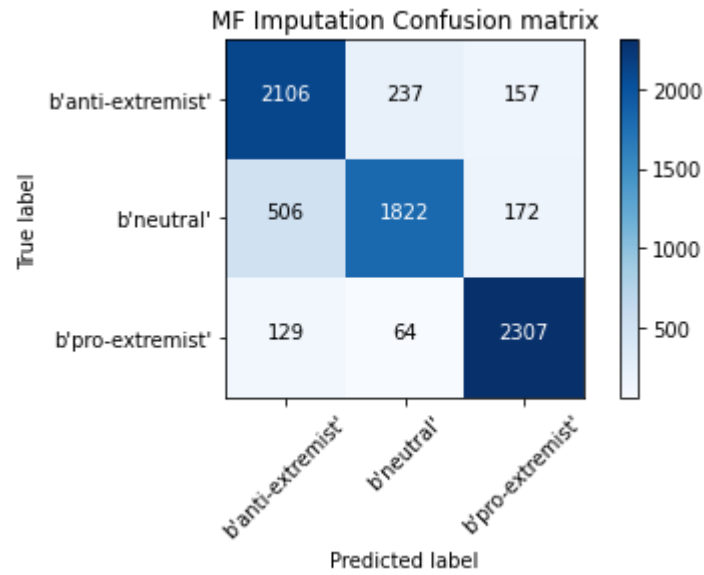


Figure 4.22: MF Imputation Confusion Matrix

4.6.2 Sentiment-Based Framework using 27 features (KNN Imputation)

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for KNN in Table 4.19.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	20	2
2	Entropy	20	2
3	Entropy	20	2
4	Entropy	20	2
5	Entropy	19	3

Table 4.19: Gridsearch Best Parameters for KNN Imputation

KNN imputation has a general max_depth range between 19-20 which is higher than the MF range. Here, the min_sample_leaf also has a range of 2-3. On average, the max depth demand 20 branches before the best score is achieved on the KNN imputation. For this fact, this increase has a positive effect on the overall classification accuracy to a credit of 84.3%. The model correctly identified pro-extremist cases at 88.6%, more than other categories. The results are detailed in Table 4.20 and the confusion matrix in Figure 4.24. The variance between each validation and training score indicates minimal overfitting. The data plot showing the performance of all the 5 folds about their maximum depth is presented in Figure 4.23.

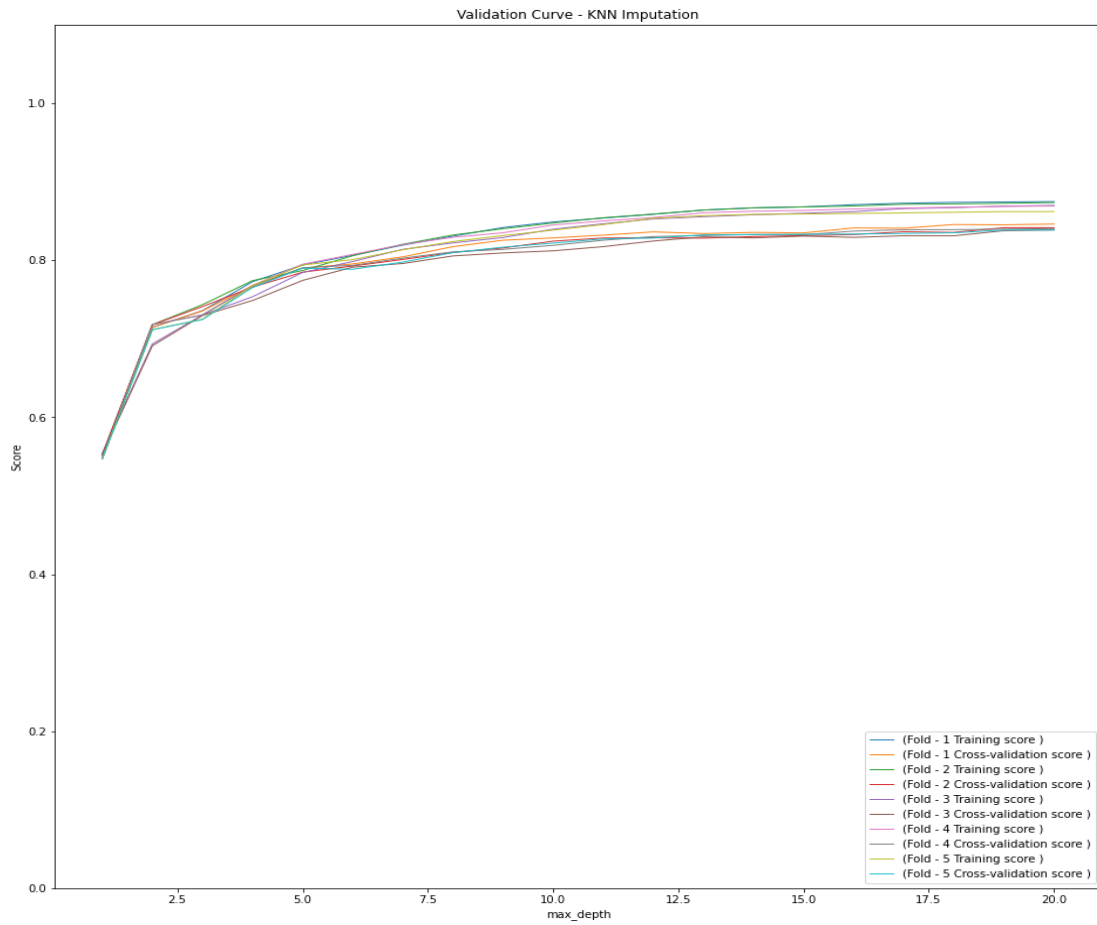


Figure 4.23: Validation Curve-KNN Imputation

False Positives Rate	Recall	Precision	F1	
0.045	0.824	0.901	0.861	Anti-Extremist
0.818	0.818	0.841	0.829	Neutral
0.113	0.8861	0.797	0.839	Pro-Extremist

Table 4.20: KNN Imputation Classification Result

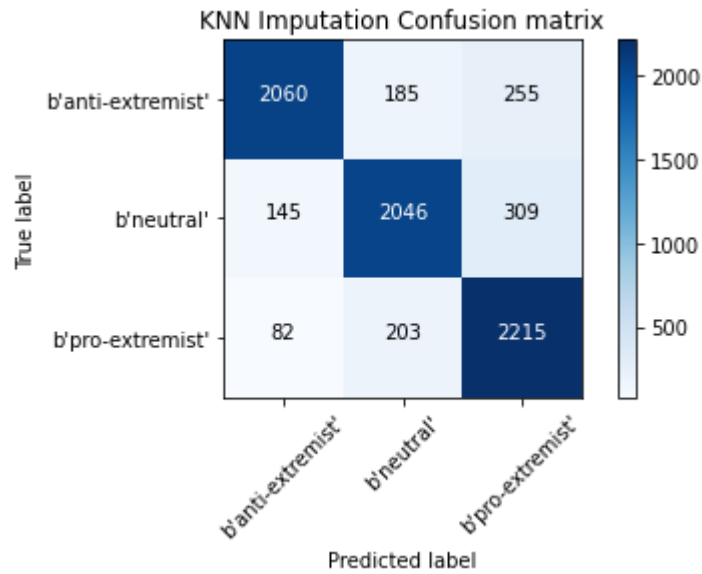


Figure 4.24: KNN Imputation Confusion Matrix

4.6.3 Sentiment-Based Framework using 27 features (Mice Imputation)

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for Mice in Table 4.21.

Folds	criterion	max_depth'	min_samples_leaf
1	entropy	19	2
2	entropy	19	2
3	entropy	19	2
4	entropy	19	2
5	entropy	20	2

Table 4.21: Gridsearch Best Parameters for Mice Imputation

For the mice, the max depth demand increased again with the same range as KNN having 19-20 but a lower average of 19. The min_sample_leaf remained at a constant of

2. The J48 gave an overall accuracy of 86.5%. The pro-extremist category had the most correctly classified cases at 86.5%. The classification results were detailed in Table 4.22 and the confusion matrix in Figure 4.26 The training and validation plot for the model is presented in Figure 4.25. The low variance between the curves indicated that the model does not overfit.

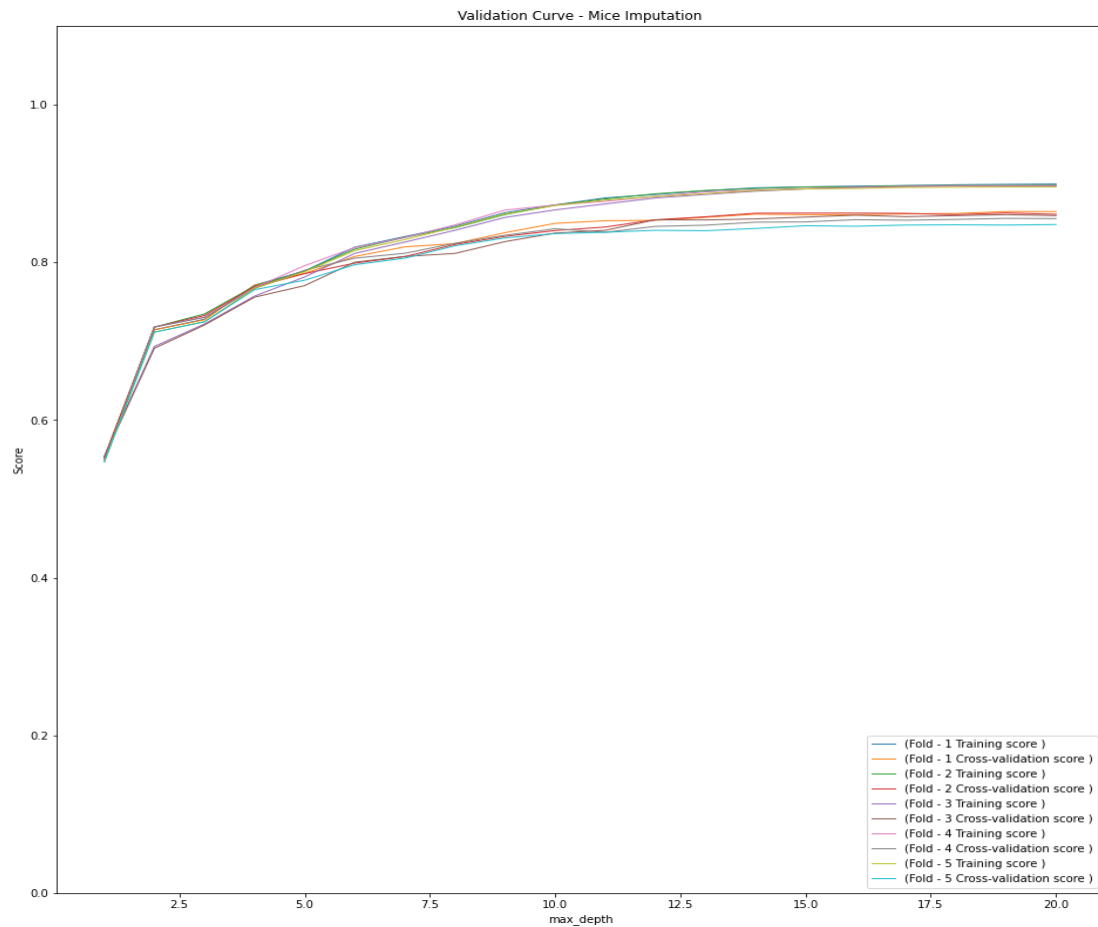


Figure 4.25: Validation Curve-Mice Imputation

False Positives Rate	Recall	Precision	F1	Class
0.051	0.865	0.895	0.88	Anti-Extremist
0.081	0.864	0.842	0.853	Neutral
0.071	0.865	0.858	0.862	Pro-Extremist

Table 4.22: Mice Imputation Classification Result

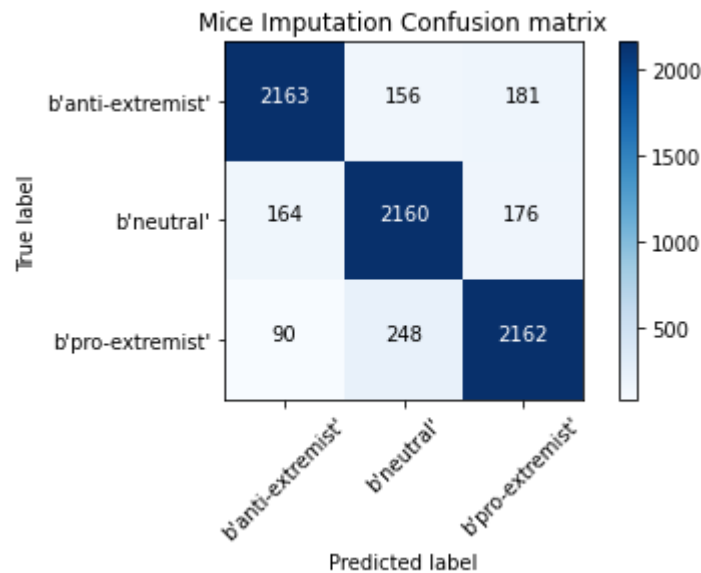


Figure 4.26: Mice Imputation Confusion Matrix

4.6.4 Feature Selection: Mice Imputation Features

In the classification model, the wrapper method and the embedded algorithms were applied. The embedded method underperformed the wrapper method, it had an early climax where it achieved its best score of 75.8% with just 30% of the features at a speed of 0.21sec while the wrapper method achieved its best score using all 90% of features

with an accuracy of 87.0% at 0.34 sec runtime. These results are detailed in Figures 4.27 and 4.28 respectively.

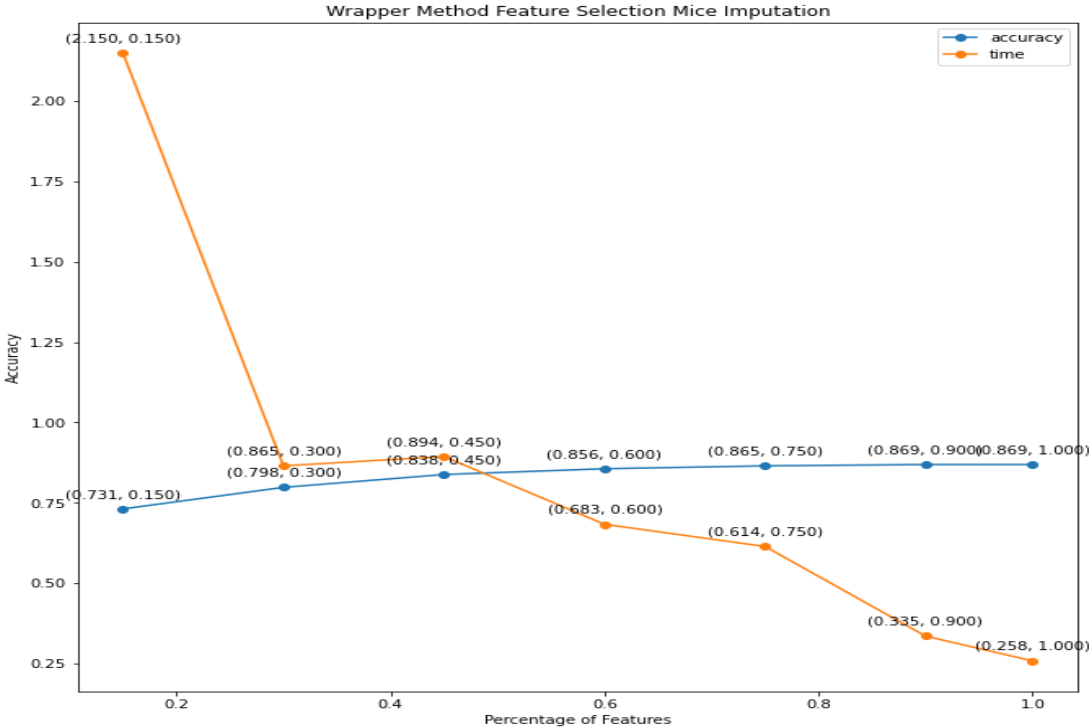


Figure 4.27: Wrapper Feature Selection Method –Mice Imputation

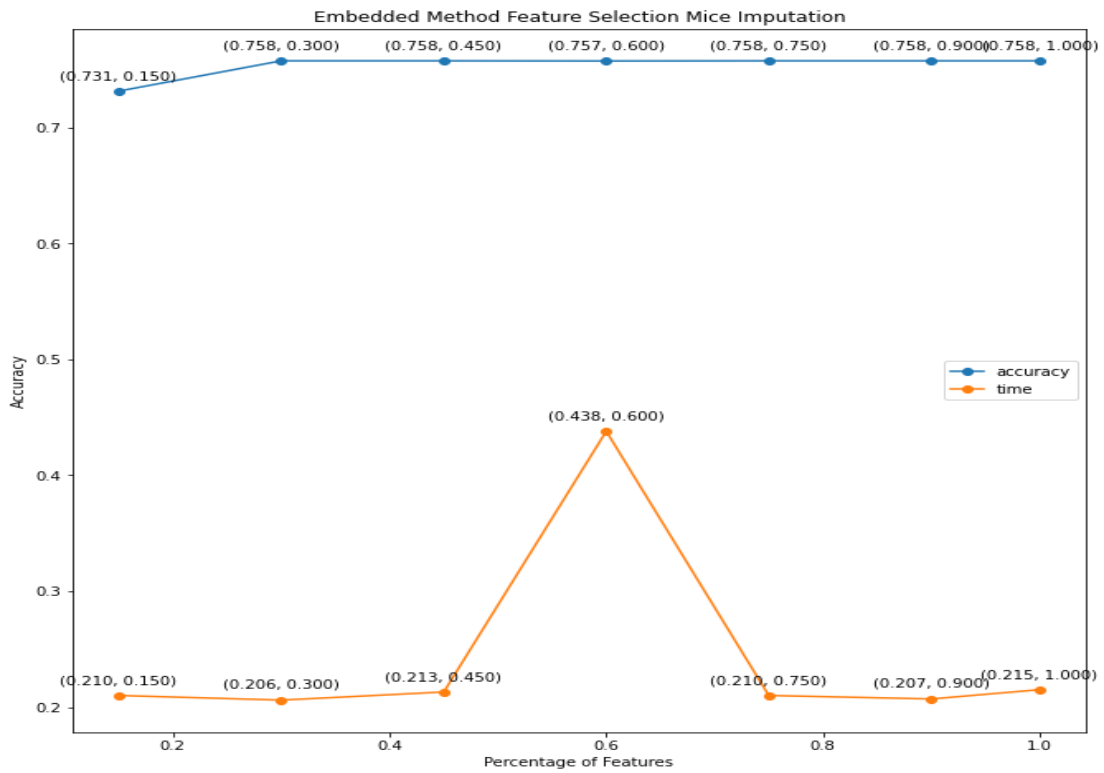


Figure 4.28: Embedded Method Feature selection Mice Imputation

4.6.5 Extended Posit

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for Extended Posit in Table 4.23 below.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	15	2
2	Entropy	18	4
3	Entropy	20	2
4	Entropy	17	4
5	Entropy	16	3

Table 4.23: Gridsearch Best Parameters for Extended Posit

In the Extended Posit case, we noticed a drastic reduction in the max depth demands to a range of 15-20, with an average of 17. The min_sample_leaf increased to a range of 2-4 and an average of 3. It's okay to say that the Extended Posit dataset places lower max_depth demands on the decision tree but generally increases the min_sample_leaf for the best performance to be achieved. Eventually, J48 gave 89.6% of correctly classified instances of overall Webpages. The pro-extremist category had the most correctly identified pages, at 92.9%. The detailed results are shown in Table 4.24 and the confusion matrix in Figure 4.30. Below is the graph showing the performance of all the 5 folds in relationship to their maximum depth in Figure 4.29.

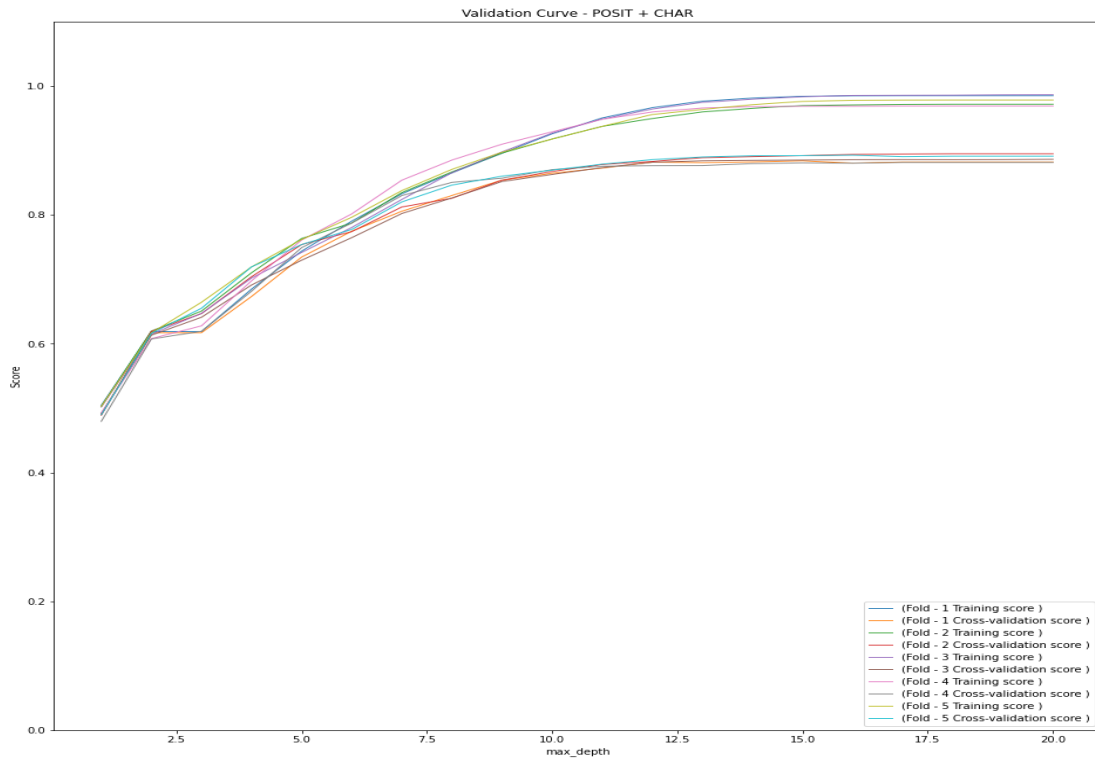


Figure 4.29: Validation Curve-Extended POSIT Imputation

False Positives Rate	Recall	Precision	F1	Class
0.065	0.88	0.872	0.876	Anti-Extremist
0.062	0.879	0.876	0.877	Neutral
0.029	0.929	0.942	0.936	Pro-Extremist

Table 4.24: Extended POSIT Imputation Classification Result

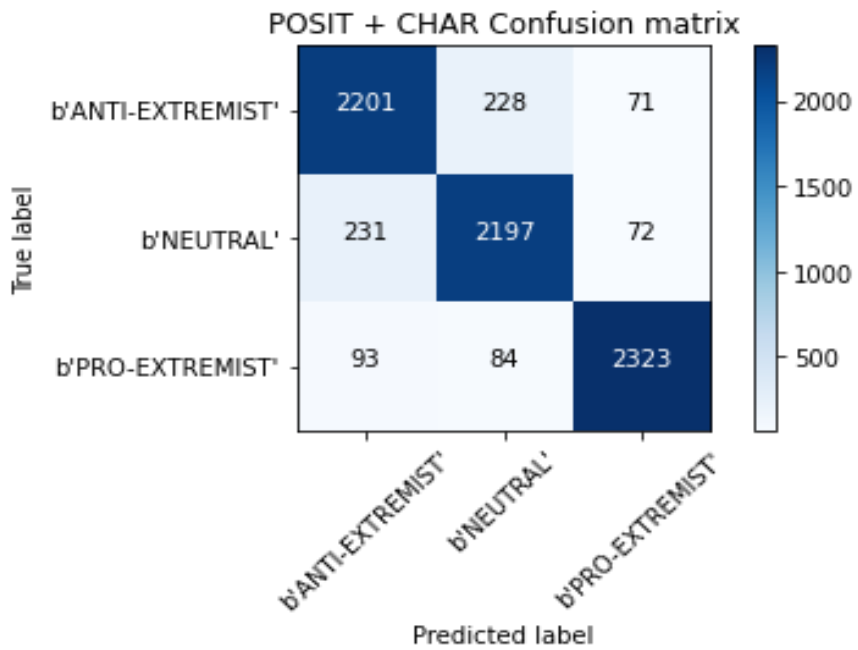


Figure 4.30: Extended POSIT Imputation Confusion Matrix

4.6.6 Feature Selection: Extended-Posit

Again, feature selection algorithms were implemented on Extended Posit of which the wrapper method slightly outperformed the embedded method by achieving a higher accuracy with just 45% of its feature subset and achieved an accuracy of 90% at a processing speed of 9.3sec while the embedded method achieved its highest accuracy of 89.9% at 1.2sec with just 30% of its feature subsets. Unarguably based on performance the wrapper method had a higher accuracy but as far as cost is concerned, the embedded method achieved nearly the same accuracy with just 30% of its features. The results are presented in Figures 4.31 and 4.32 respectively.

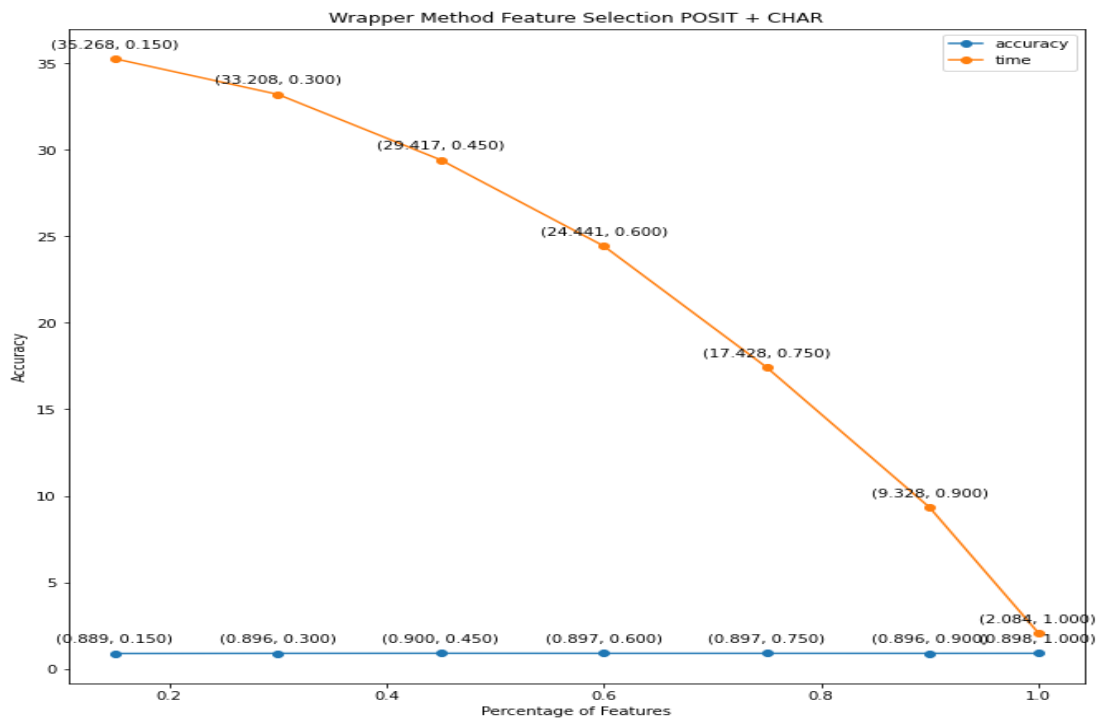


Figure 4.31: Wrapper Method–Extended Posit

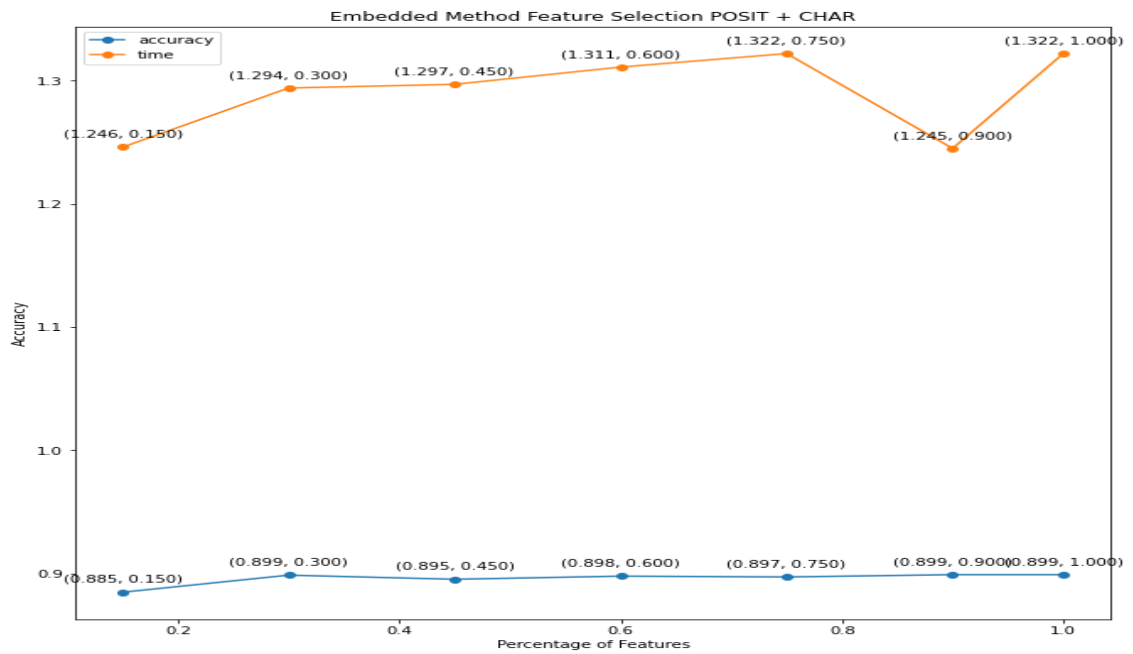


Figure 4.32: Embedded Method for Extended-POSIT

4.6.7 Posit

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for Posit in Table 4.25 below

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	15	2
2	Entropy	15	3
3	Entropy	13	2
4	Entropy	14	2
5	Entropy	14	3

Table 4.25: Gridsearch Best Parameters for Posit

Here in the posit, we have a lower max depth demand of 13-15 with an average of 14, and a min_sample_leaf of 2-3 with an average of 2. The decision tree finished with an overall classification accuracy of 89.4% with the pro-extremist category classified at 91%, higher than the other three categories. The results are presented in Table 4.26 and the confusion matrix in Figure 4.34. Again, the validation and training curves displayed in Figure 4.33 indicates a low level variance and hence the model does not overfit.

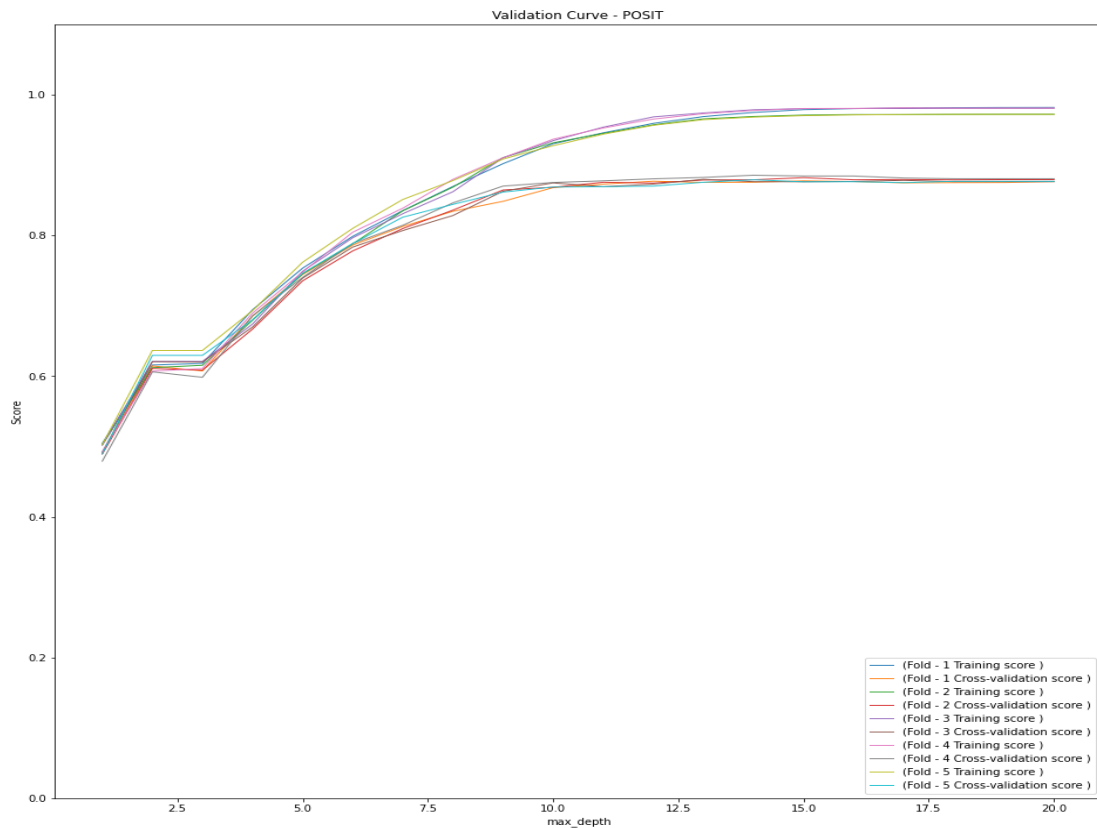


Figure 4.33: Validation Curve Posit Imputation

F P Rate	Recall	Precision	F1	Class
0.066	0.898	0.872	0.885	Anti-Extremist
0.874	0.874	0.886	0.88	Neutral
0.037	0.911	0.925	0.918	Pro-Extremist

Table 4.26: Posit-Based Classification Result

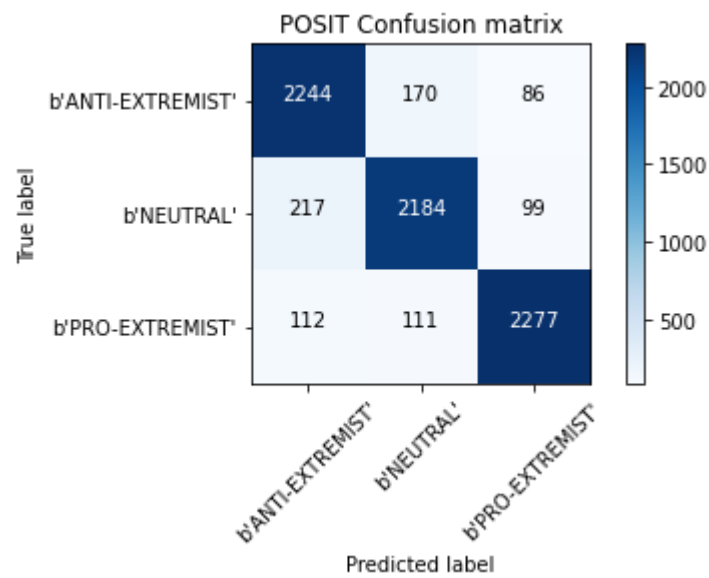


Figure 4.34: Posit Imputation Confusion Matrix

4.6.8 Feature Selection: Posit

In the Posit classification framework, the wrapper method outperforms the embedded with a better result of 89.6% with just 60% of the feature subsets after which the performance of the model began to deplete as more features were added. This is achieved at a speed of 9.33sec. The results of both wrapper and embedded results are presented in Figures 4.35 and 4.36 respectively.

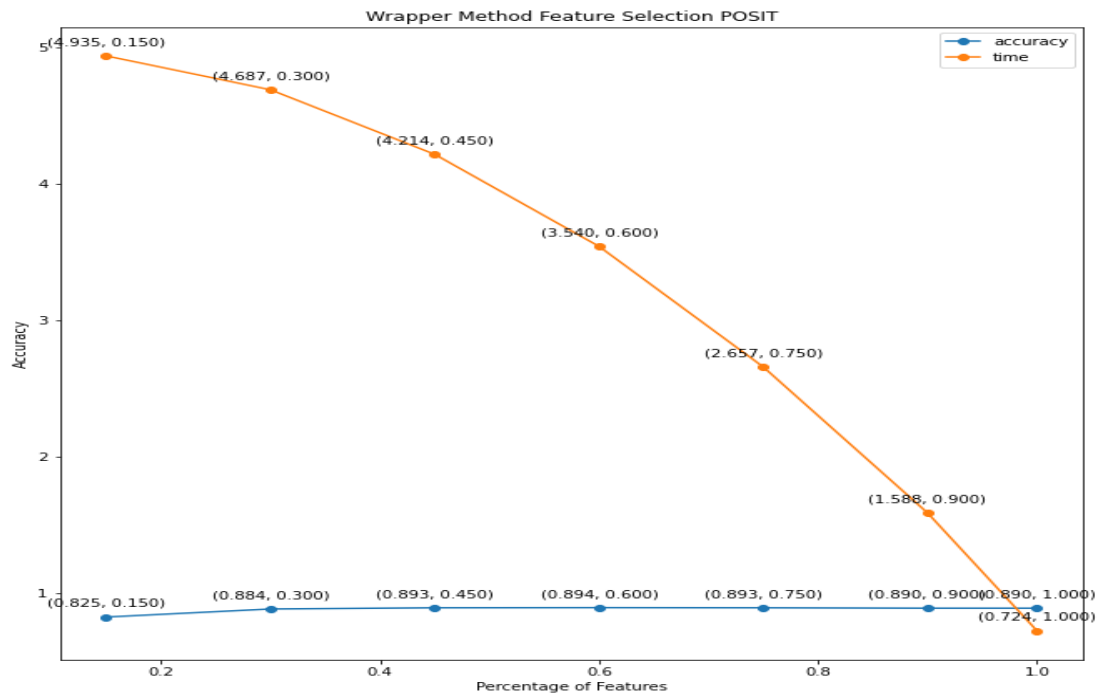


Figure 4.35: Wrapper Method Feature Selection – Posit

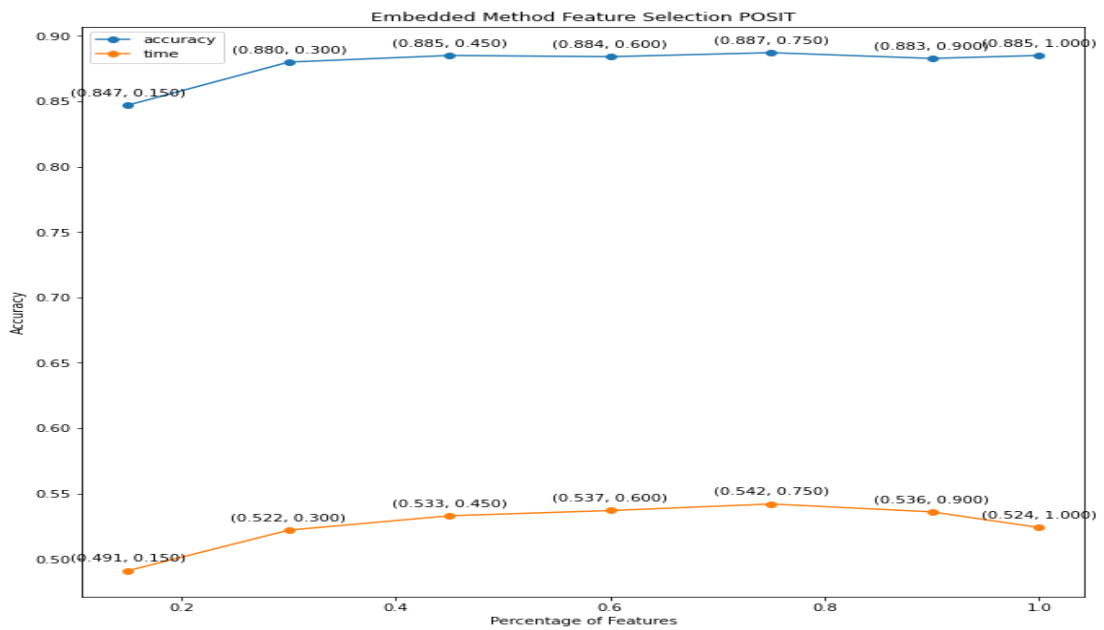


Figure 4.36: Embedded Method Feature selection Posit

4.6.9 Composite Based Classification Framework

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation for Composite in Table 4.27 below.

Folds	Criterion	max_depth'	min_samples_leaf
1	Entropy	17	3
2	Entropy	14	4
3	Entropy	17	2
4	Entropy	15	2
5	Entropy	12	2

Table 4.27: Gridsearch Best Parameters for Composite

The Composite demands a max_depth range of 12-17, with an average of 15, which is a unit higher than the previous Posit from Table 4.34 above. The J48 decision tree maintained min_sample_leaf between 2-4 with an average of 3. These achieved an overall accuracy of 91.5% with excellent individual performances of each class with accuracies over 91.5%. The pro-extremist category had the highest level of correctly identified pages, at 93.2%. The classification result and the confusion matrix are presented in Tables 4.28 and Figure 4.38 below. Figure 4.37 shows the validation and training curves which indicate the model is well fitted.

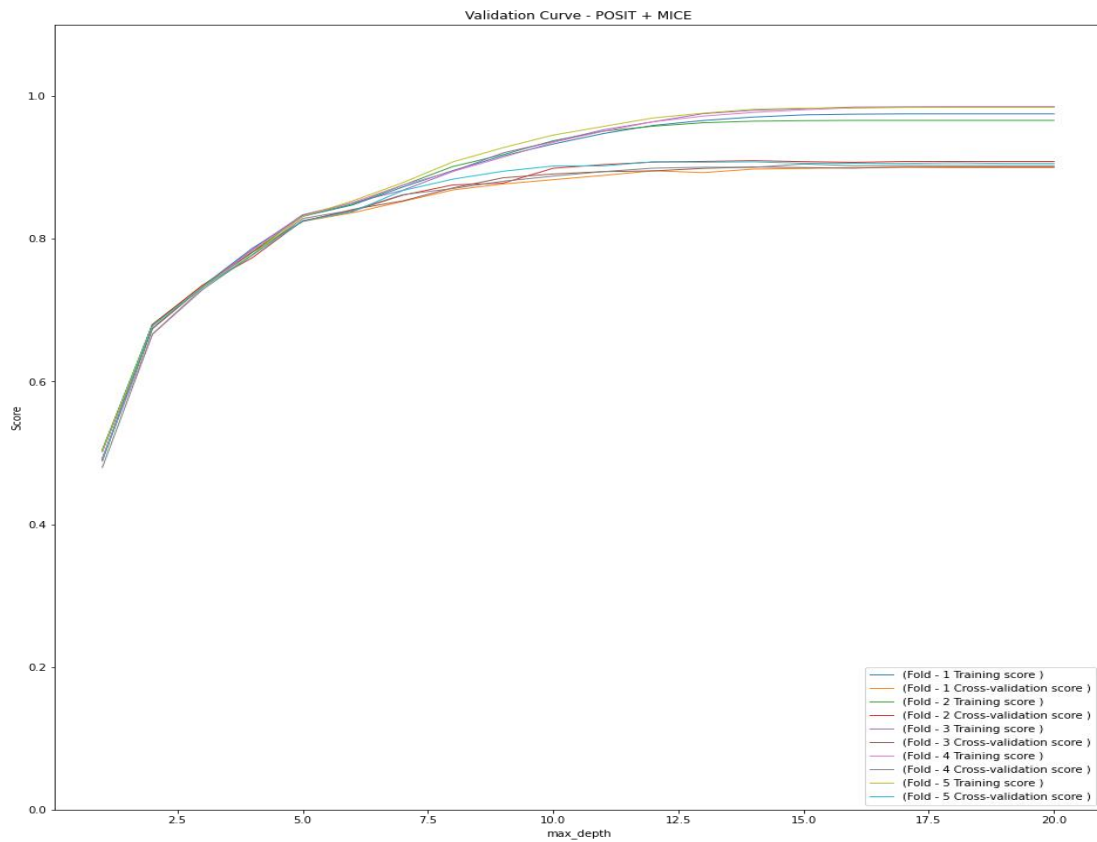


Figure 4.37: Validation Curve Posit Mice Imputation

F P Rate	Recall	Precision	F1	Class
0.052	0.909	0.898	0.903	Anti-Extremist
0.047	0.906	0.906	0.906	Neutral
0.028	0.932	0.943	0.938	Pro-Extremist

Table 4.28: Composite-Based Classification Result

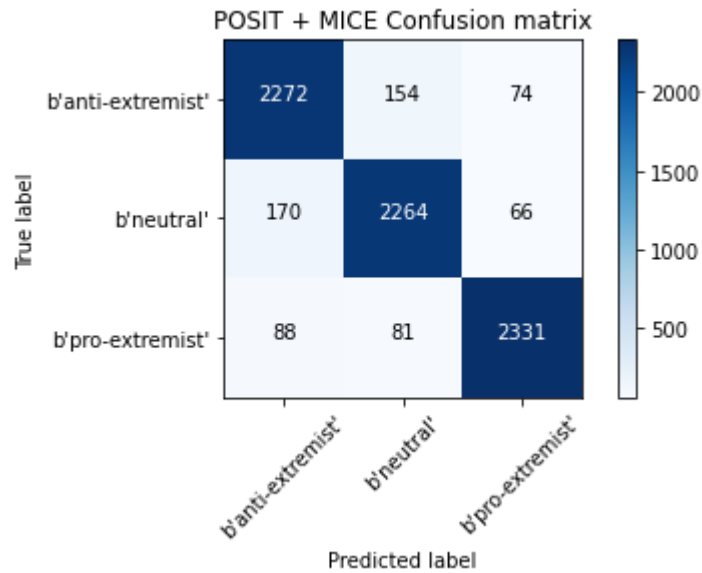


Figure 4.38: Composite-Based Confusion Matrix

4.6.10 Feature Selection: Composite-based classification framework

In the composite based classification framework, the wrapper method outperforms the embedded method with just 60% of its features utilized which achieved a high accuracy of 91.3% at 8.04sec while the embedded method achieved its highest of 89.8% with all 90% of its feature hence making the wrapper method more cost effective. The results are detailed in Figures 4.39 and 4.40 respectively.

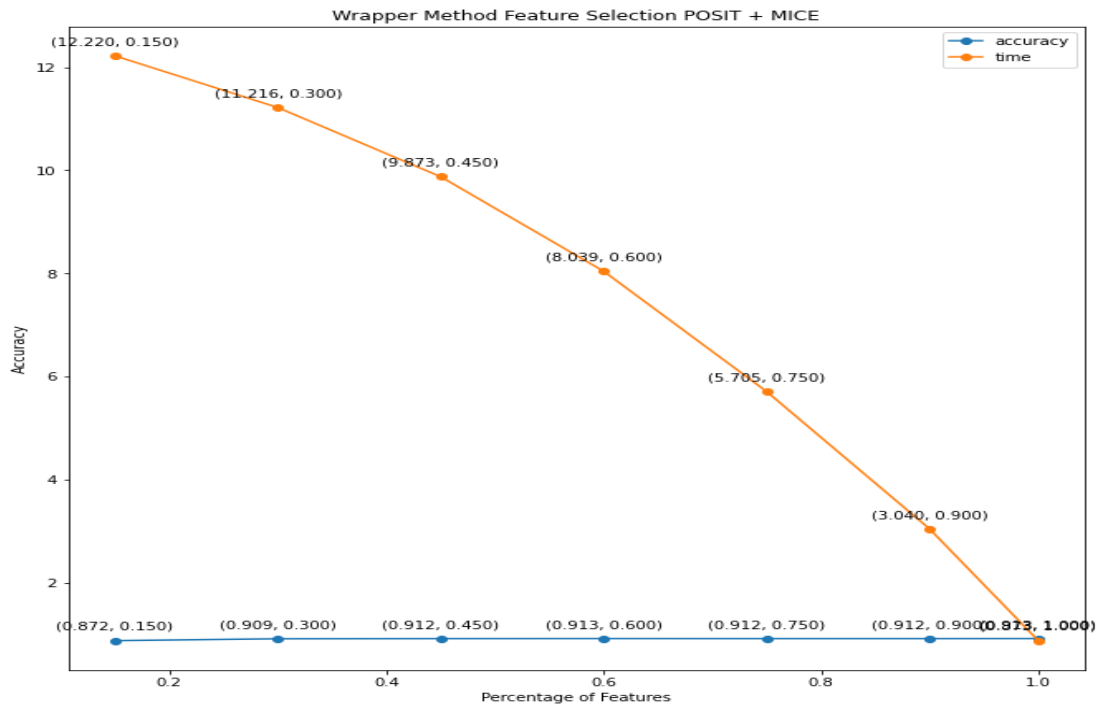


Figure 4.39: Wrapper Method for Composite

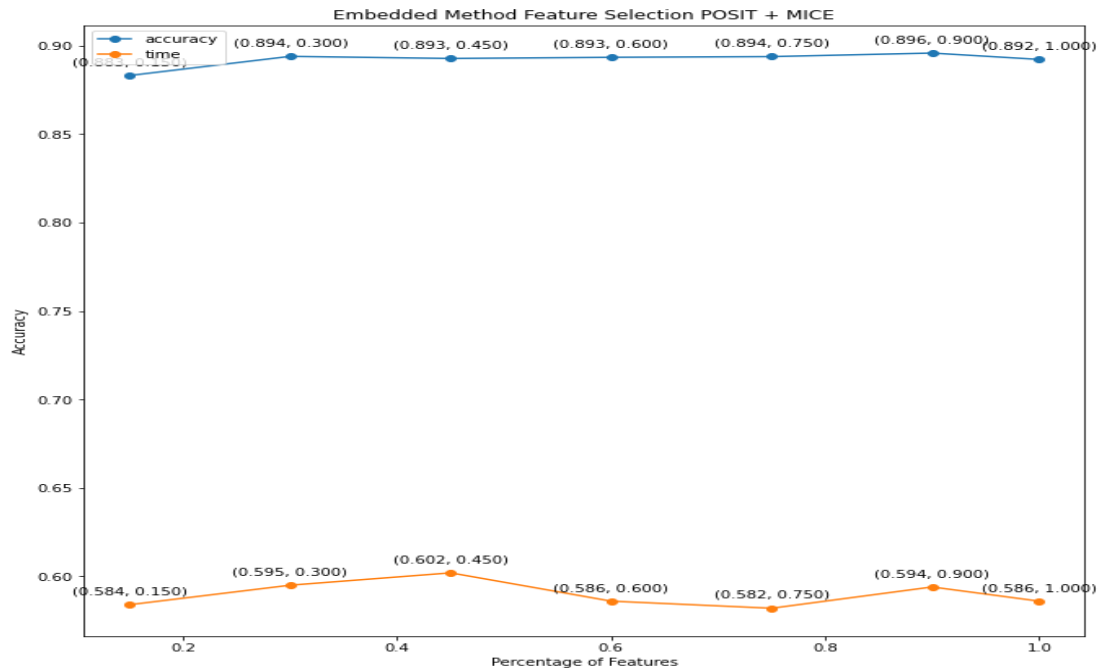


Figure 4.40: Embedded Method for Composite

4.7 KNN CLASSIFICATION RESULTS

This section presents the result of each framework using KNN. Table 4.29 below presented the parameters used in the KNN model across the frameworks.

KNN	leaf_size,	1- 10
	'n_neighbors	1, 3, 5, 7, 9, 11, 13
	'weights	Uniform

Table 4.29: The KNN Model Parameters

4.7.1 MF Imputation

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of MF Imputation using KNN in Table 4.30 below.

Folds	leaf_size	n_neighbors	weights
1	1	1	uniform
2	1	3	uniform
3	4	5	uniform
4	1	3	uniform
5	4	3	uniform

Table 4.30: Gridsearch Best Parameters for MF Imputation

After the application of grid search through 5-fold cross-validation, the leaf_size parameter which was initially set between the range of 1-10, could only iterate between 1-4, no fold exceeded a leaf_size of 4 while the least leaf_size a fold attained is 1. This tells us that in general, our model is expected to perform best and best fit the MF imputation data if our leaf_sizes are set between 1-4. It's also observed that for the MF imputation data, our best performance was found within the n_neighbors range of 1-5 against an initial range of 1-13. The leaf size is a parameter in a KD tree or KD ball tree algorithm in KNN that helps to partition, allocate or organize data points in a multi-dimension (multi-feature) space by calculating the distances between each data point. So, the larger the leaf_size, the slower the classification of these data points and vice versa. The n_neighbors are the total number of data points closest to a selected data point. We observed that the MF only needed an average of 3 n_neighbors to attain the best performance. Another key observation is how the n_neighbors keeps increasing each time the leaf_size increases as we go down the 5 folds except for the 5th fold. This means that the n_neighbors is very sensitive to the leaf_size and must be more than the leaf_size and with a range 1-5 if the best performance must be attained within the MF imputation.

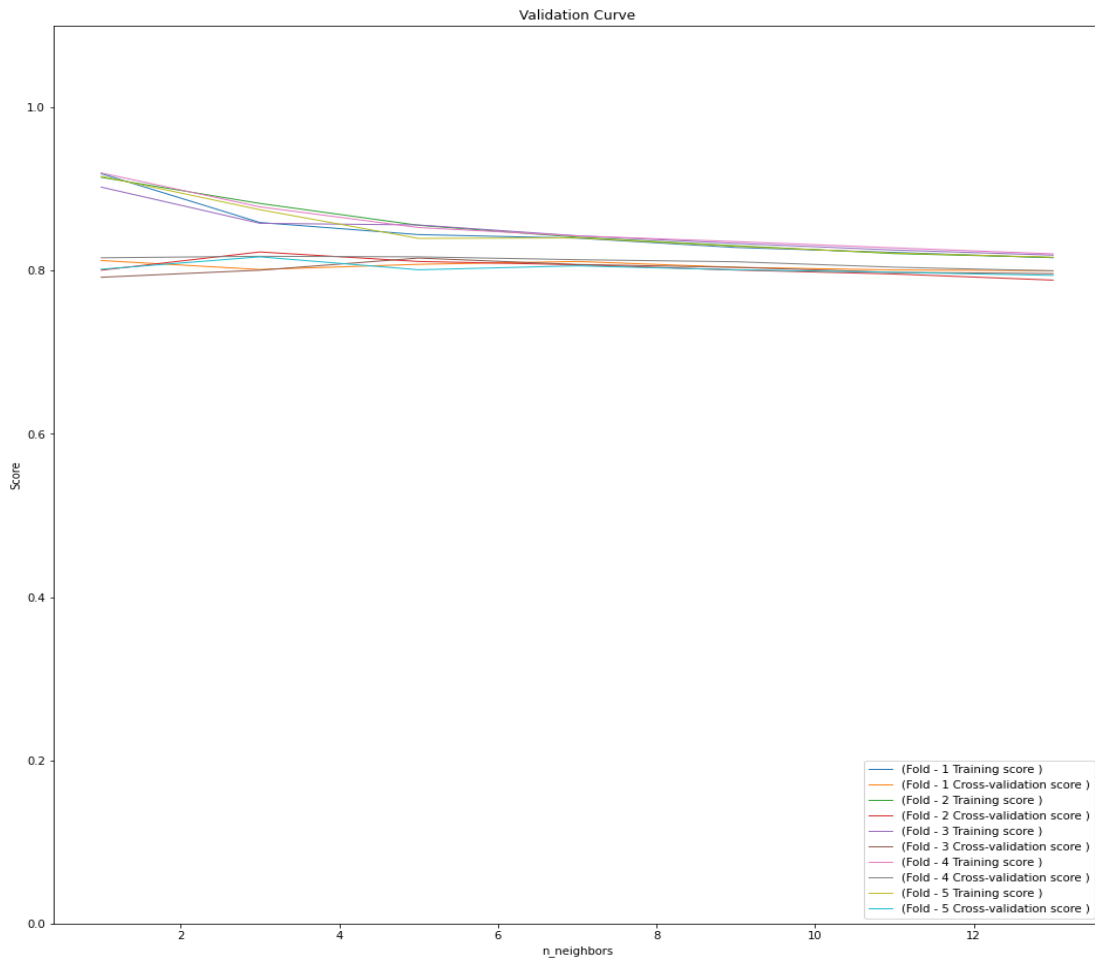


Figure 4.41: Validation Curve-MF Imputation

From Figure 4.41 above showing the training and test curves, a behavior is noticed that is consistent with both curves is the fact that the performance of the model reduces every time the `n_neighbors` increase. This confirms our initial findings within the grid search folds iterations that optimal performance for the MF imputation can only be found at `n_neighbors` ranges between 1-5. If we are to be a bit specific, the optimal performance can be found at an `n_neighbor` value of 3 because that is where most of the convergences between each folds occurred. In actuality, the over-fitting tendencies get to reduce (because both training and test folds get to converge better) as we increase the `n_neighbors` but we'd have to sacrifice performance to reduce over-fitting tendencies.

KNN gave an overall classification of 82.1%. The results of other categories are shown in Table 4.31. The model correctly identified pro-extremist cases more than other categories in the confusion matrix presented in Figure 4.42.

False Positives Rate	Recall	Precision	F1	Class
0.118	0.827	0.777	0.801	Anti-Extremist
0.051	0.715	0.875	0.787	Neutral
0.098	0.922	0.824	0.87	Pro-Extremist

Table 4.31: MF Imputation Classification Result using KNN

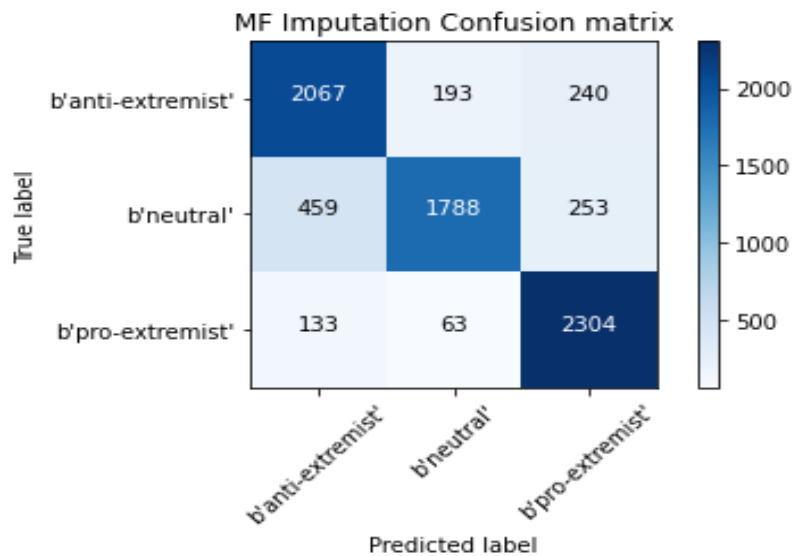


Figure 4.42: MF Imputation Confusion Matrix

4.7.2 KNN Imputation

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of MF Imputation using KNN in Table 4.32 below.

Folds	leaf_size	n_neighbors	Weights
1	2	9	Uniform
2	2	11	Uniform
3	2	7	Uniform
4	8	9	Uniform
5	1	9	Uniform

Table 4.32: Gridsearch Best Parameters for KNN Imputation

In the KNN imputation, we have a more complex scenario where the expected leaf_size range for optimal performance has increased to a range of 1-8 unlike MF which had 1-4, likewise the n_neighbors which also increased to a range of 7-11. On average, the optimal leaf_size is 3 and the optimal n_neighbors is 8 but this rather didn't reflect on the accuracy as we rather had a little lower performance, 81% against 82% we had in the MF case. Again, this boils down to the uniqueness of the dataset via the method of imputation. In the KNN imputation, the KNN algorithms needed a higher leaf_size for its processing as the KNN data may be a more complex and demanding dataset such that the KNN algorithm finds it difficult to establish a classification or decision boundaries within the multi-dimensional spatial regions hence, needing more leaf_size number and more n_neighbors.

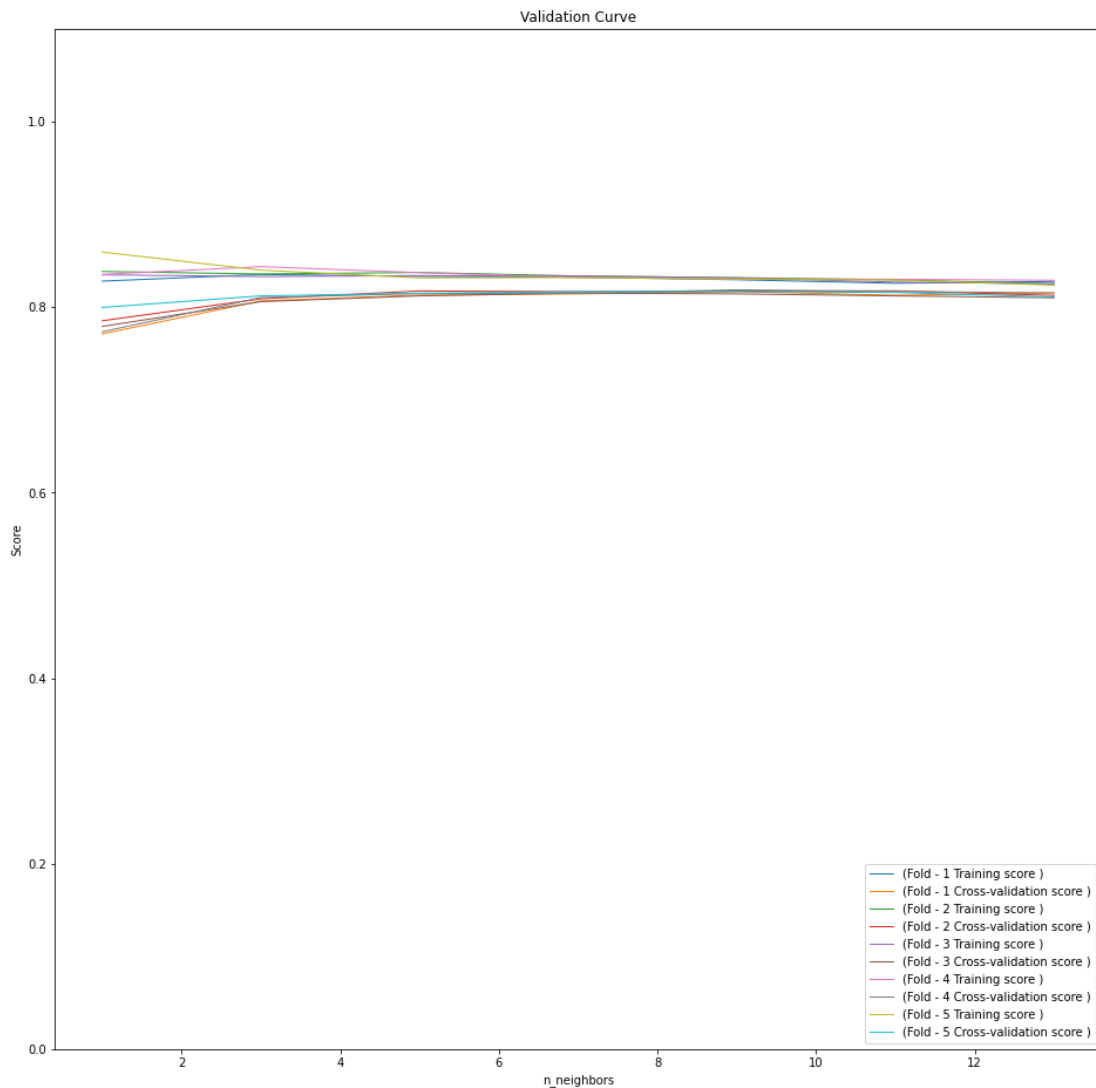


Figure 4.43: Validation Curve-KNN Imputation

From the training curves above in Figure 4.43, we can see that the model is a bit healthier than the previous MF such that there were little or no changes in accuracy after n_neighbors is above 4 in the graph except for the fact that there was increased convergence (reduced overfitting), which explains why it was necessary for the algorithm to increase the n_neighbors to as high as 9 and 11 to get the best performance, it was trying to curb overfitting. This places the KNN imputation as a better candidate model than the MF imputation. The results of other categories are

shown in Table 4.33. The KNN model correctly identified pro-extremist cases more than other categories in the confusion matrix presented in Figure 4.44.

False Positives Rate	Recall	Precision	F1	Class
0.059	0.821	0.873	0.846	Anti-Extremist
0.021	0.689	0.941	0.796	Neutral
0.189	0.949	0.715	0.815	Pro-Extremist

Table 4.33: KNN Imputation Classification Result using KNN

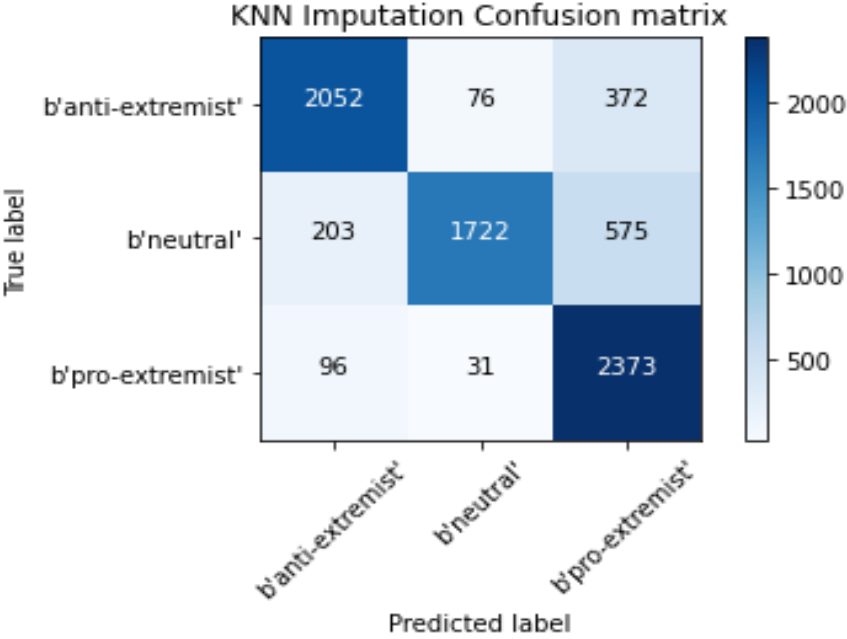


Figure 4.44: KNN Imputation Confusion Matrix

4.7.3 Mice Imputation

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of Mice Imputation using KNN in Table 4.34 below.

Folds	leaf_size	n_neighbors	weights
1	2	9	uniform
2	1	5	uniform
3	4	7	uniform
4	2	9	uniform
5	1	7	uniform

Table 4.34: Gridsearch Best Parameters for MICE Imputation

In the case of the Mice, we have less demanding parameters where the leaf_size has an average of 2 while the n_neighbors has an average of 7. Previously in the KNN, we had n_neighbors reaching a max of 11 whereas in MICE we have a max of 9.

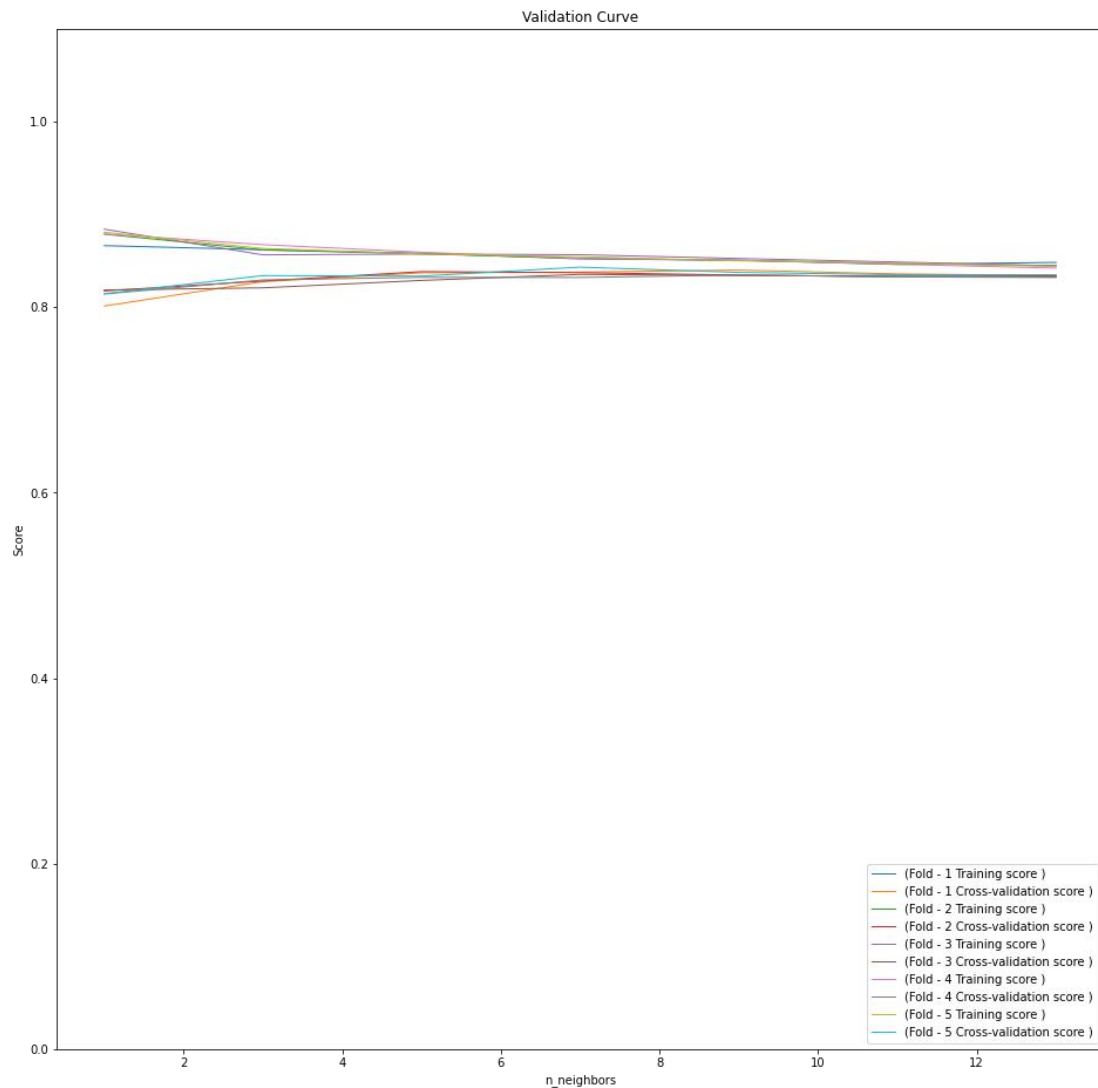


Figure 4.45: Validation Curve-Mice Imputation

Again, looking at the training curves above in Figure 4.45, just like KNN, the model accuracy stopped increasing when the `n_neighbors` attains 3 but after which it began to fight overfitting by increasing the `n_neighbors` to 9 where it hit its optimal performance. For the Mice, we even had an increased overall performance of 83% which turns out to have outperformed both MF and KNN. So, as soon as the model reached `n_neighbors` attains 9, it stopped converging and stopped improving hence making iterations of

n_neighbors>9 redundant to our experimentation. The remaining results of other categories are detailed in Table 4.35 and the confusion matrix is presented in Figure 4.46

False Positives Rate	Recall	Precision	F1	Class
0.062	0.858	0.872	0.865	Anti-Extremist
0.028	0.704	0.925	0.799	Neutral
0.152	0.951	0.758	0.844}	Pro-Extremist

Table 4.35: Mice Imputation Classification Result using KNN

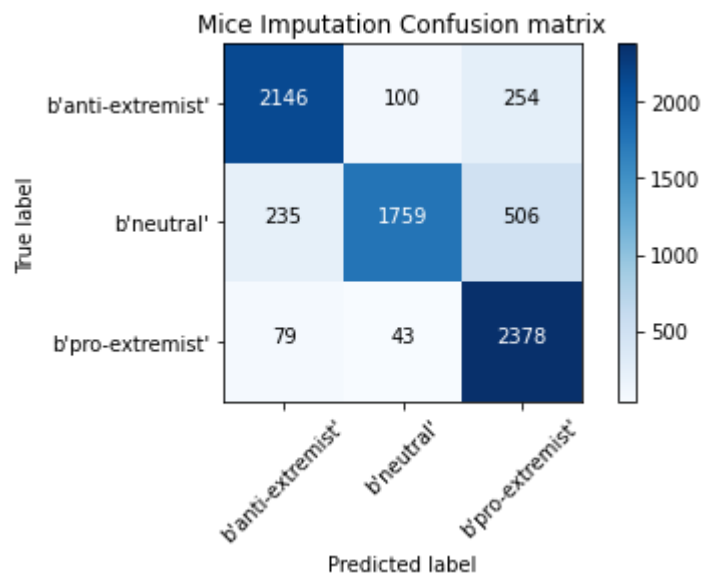


Figure 4.46: Mice Imputation Confusion Matrix

4.7.4 Extended- Posit Classification

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of Extended- Posit using KNN in Table 4.36 below.

Folds	leaf_size	n_neighbors	weights
1	1	3	uniform
2	1	5	uniform
3	1	3	uniform
4	1	5	uniform
5	1	7	uniform

Table 4.36: Gridsearch Best Parameters for Extended-Posit

For the Extended-Posit, we have a very interesting observation where on average leaf_size=1 and n_neighbors=5. This is a way less demanding set of parameters compared to the MF, MICE and KNN. For the Extended-Posit, the algorithm only needed a small leaf size to accurately establish a classification boundary within the spatial space. We observed that for each leaf_size=1, there is an increasing n_neighbors value from 3 to 7.

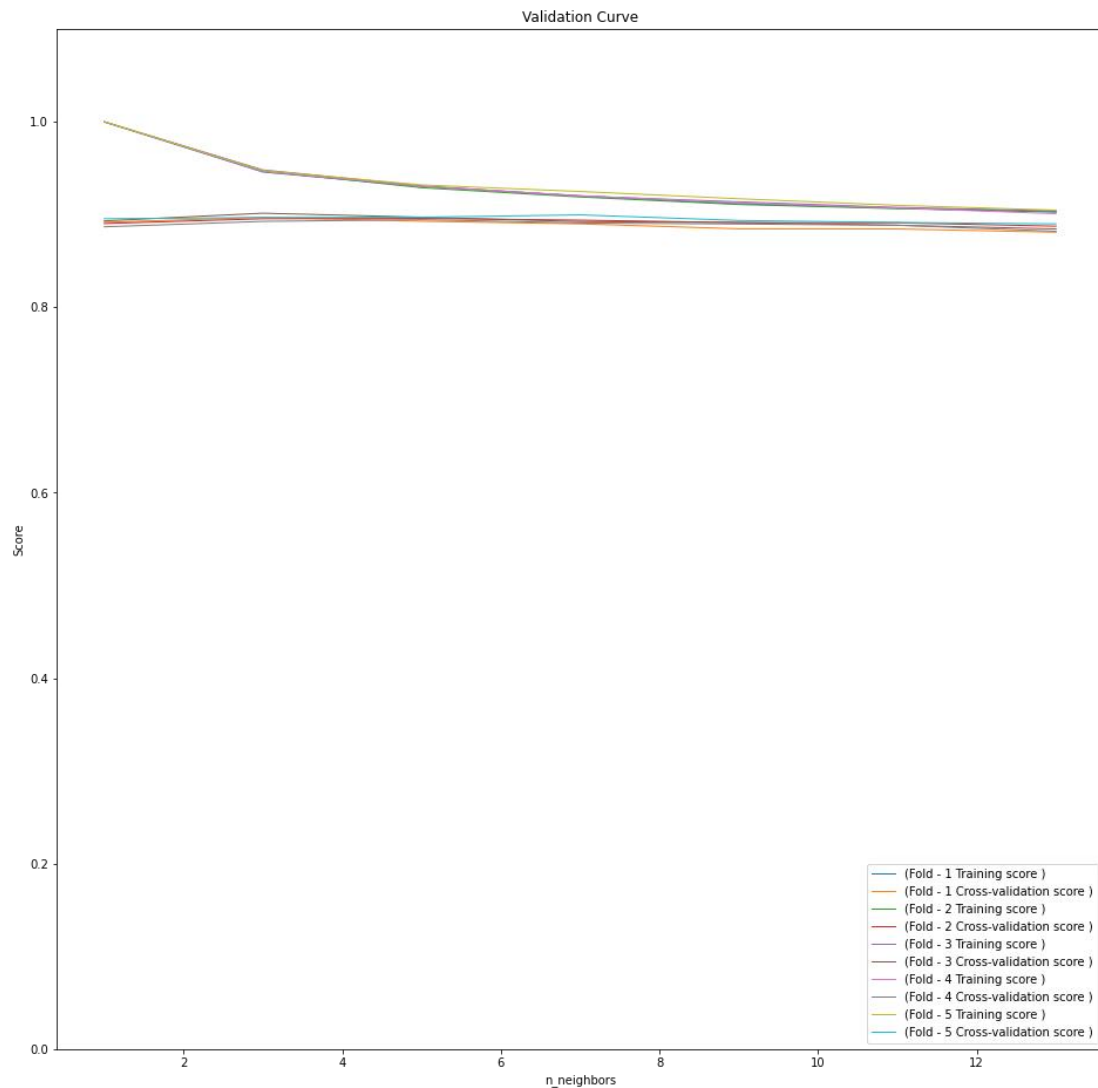


Figure 4.47: Validation Curve-Extended Posit

From the training and test curves above in Figure 4.47, you would notice that the same habit that occurred in mice and KNN imputations happened again, where the accuracy stalled after an early `n_neighbors` value (3 in this Extended-Posit case) and then continued to rather increase to higher values. Here, after `n_neighbors` attains 3 the accuracy stopped increasing hence the model stopped generalizing and afterward sought to deal with overfitting tendencies and increased the `n_neighbors` to 7 where it finished converging hence every other iteration afterward either didn't improve performance or

reduce overfitting. The Extended-Posit has outperformed every other framework explained above so far with very much less demanding parameters and computational resources, having an overall accuracy of 90.3%. The most correctly identified category is pro-extremist, at the rate of 97%, with the highest precision rate when compared with other categories in the confusion matrix table shown in Figure 4.48. The details of the results are presented in Table 4.37. The validation curve in Figure 4.47 indicates a very low variance between both validation and training scores and consequently the model is well fitted.

False Positives Rate	Recall	Precision	F1	Class
0.0717	0.906	0.863	0.884	Anti-Extremist
0.0377	0.848	0.918	0.881	Neutral
0.0348	0.958	0.932	0.945	Pro-Extremist

Table 4.37: Extended Posit Classification Result using KNN

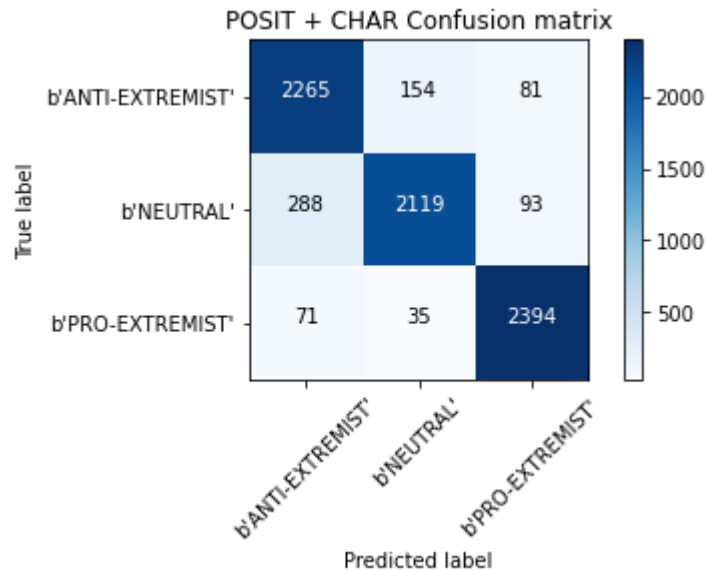


Figure 4.48: Extended Posit Confusion Matrix

4.7.5 Posit-Based Classification

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of Posit using KNN in Table 4.38 below.

Folds	leaf_size	n_neighbors	Weights
1	1	5	Uniform
2	1	5	Uniform
3	1	5	Uniform
4	1	3	Uniform
5	1	1	Uniform

Table 4.38: Gridsearch Best Parameters for Posit

The posit dataset also maintained a similar optimal range and average for the leaf_size and n_neighbors where leaf_size didn't exceed 1 but most importantly, the n_neighbors had a lesser range of 1-5 compared to Extended-Posit having 1-7. In general, the n_neighbors increased through each fold as leaf_size is 1. The small leaf_size is indicative of the fact that the dataset is less complicated and easily fits the KNN model allowing the classification boundaries to be easily established. So, if one looks at how the boundaries are formed, you would see a clean a contour neatly separating each class without much complexity.

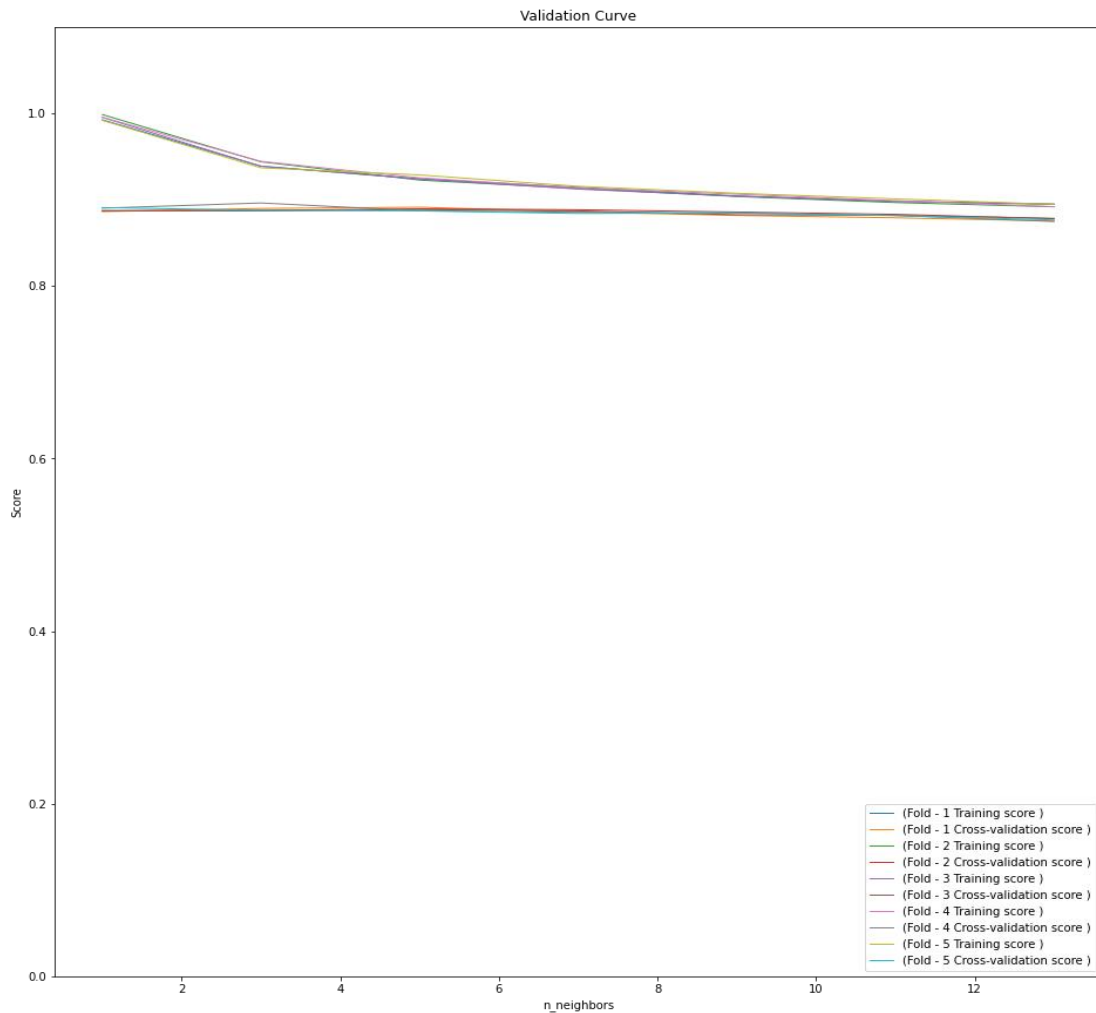


Figure 4.49: Validation Curve- Posit

The above curve in Figure 4.49 shows how the accuracy/performance of the training curve keeps reducing as the `n_neighbors` increases. The test curve maintained a steady performance and didn't change while the training curve went ahead to deal with some overfitting tendencies while it reduced to 5. After `n_neighbors` is equal to 5, the model stopped learning and reached its optimal performance. The overall performance accuracy turned out to be lower by a few decimals than the Extended-Posit, KNN gave 89%, leaving it as an underperforming model compared to the Extended-Posit but better than the KNN, mice and MF imputation data. The details of the results are shown in Table

4.39 and Figure 4.50. The validation curve in Figure 4.49 indicates a very low variance between both validation and training scores and consequently the model is well fitted

False Positives Rate	Recall	Precision	F1	Class
0.067	0.900	0.869	0.884	Anti-Extremist
0.054	0.870	0.889	0.880	Neutral
0.033	0.919	0.932	0.925	Pro-Extremist

Table 4.39: Posit Classification Result using KNN

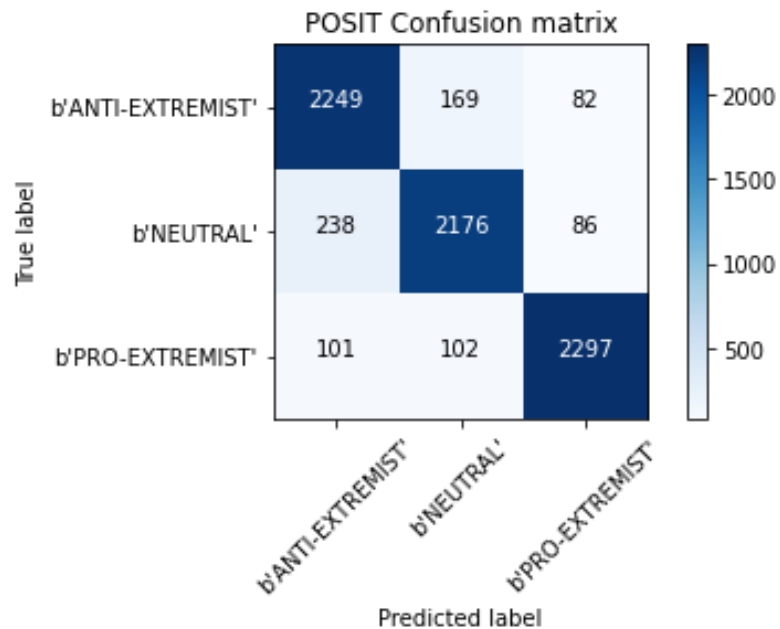


Figure 4.50: Posit Confusion Matrix

4.7.6 Composite Based Classification

The application of grid search yielded the following best parameter sets for 5 separate folds used for the validation of Composite using KNN in Table 4.40 below.

Folds	leaf_size	n_neighbors	Weights
1	1	3	Uniform
2	1	5	Uniform
3	1	5	Uniform
4	1	3	Uniform
5	1	5	Uniform

Table 4.40: Gridsearch Best Parameters for Composite

Here, the range of the leaf_size remained at 1, leaving our optimal n_neighbors range at 3-5 and an average of 4. We had our optimal value at n_neighbors is equal to 5, where some of the overfitting tendencies had been removed by the model after generalization stopped at n_neighbors is 4.

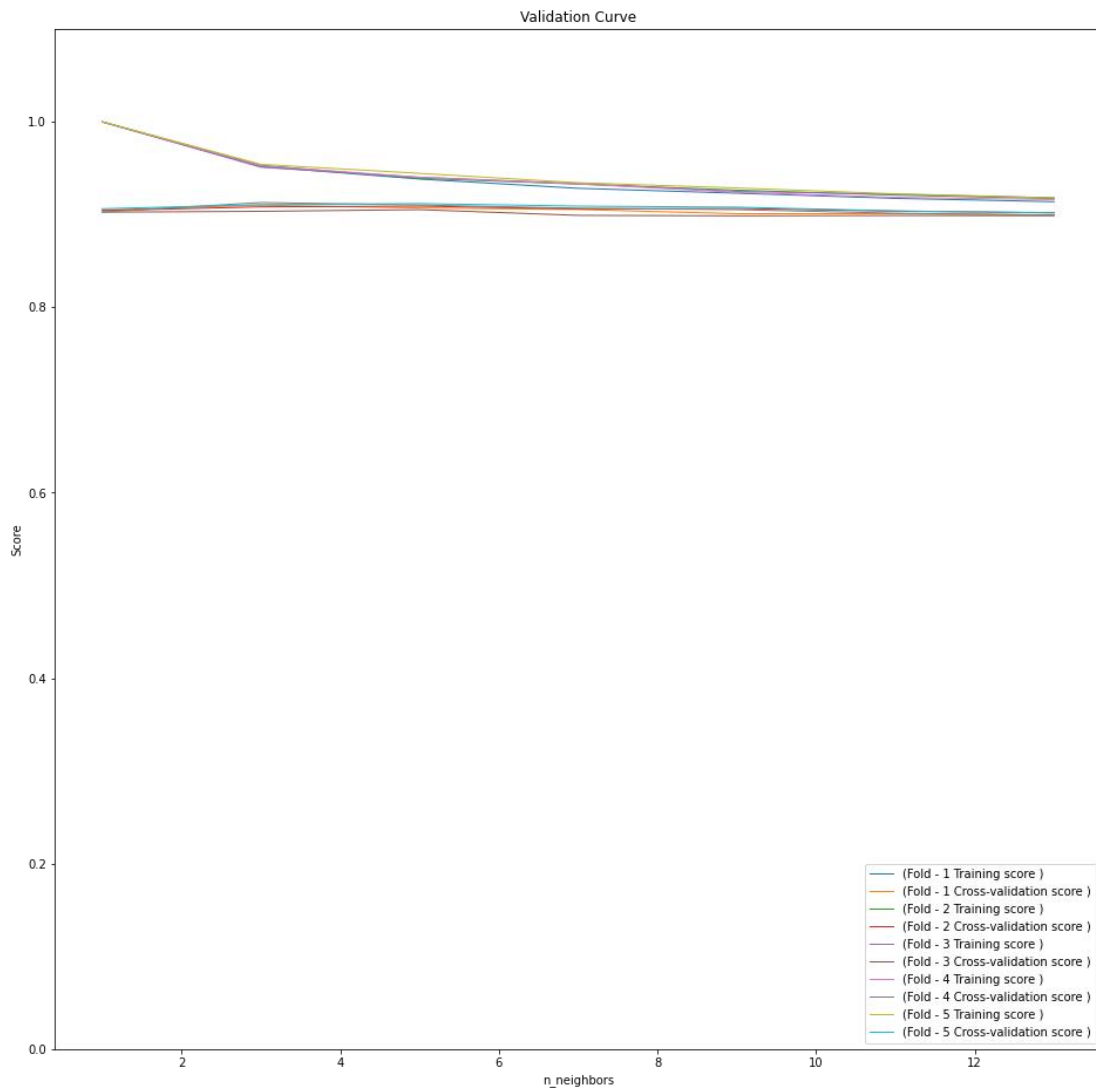


Figure 4.51: Validation Curve-Composite

The training and test curve above in Figure 4.51 rather confirms our initial observations, such that both curves seem to have different trajectories: The training curves show the decreasing accuracy and model performance as the `n_neighbors` increases while the test/validation accuracy shows an increasing model performance as the `n_neighbors` increases. But the scenario for the validation curve is only valid up to `n_neighbors` is 5 after which the model stops improving and remains steady. The Composite, having a little higher `n_neighbors` range compared to Posit and Extended-Posit has proven to

outperform every other model by reaching an overall performance of 92%. Details of the results are presented in Table 4.41 and Figure 4.52 respectively

False Positives Rate	Recall	Precision	F1	Class
0.056	0.916	0.891	0.904	Anti-Extremist
0.036	0.88	0.924	0.901	Neutral
0.031	0.956	0.938	0.947	Pro-Extremist

Table 4.41: Composite-Based Classification Result using KNN

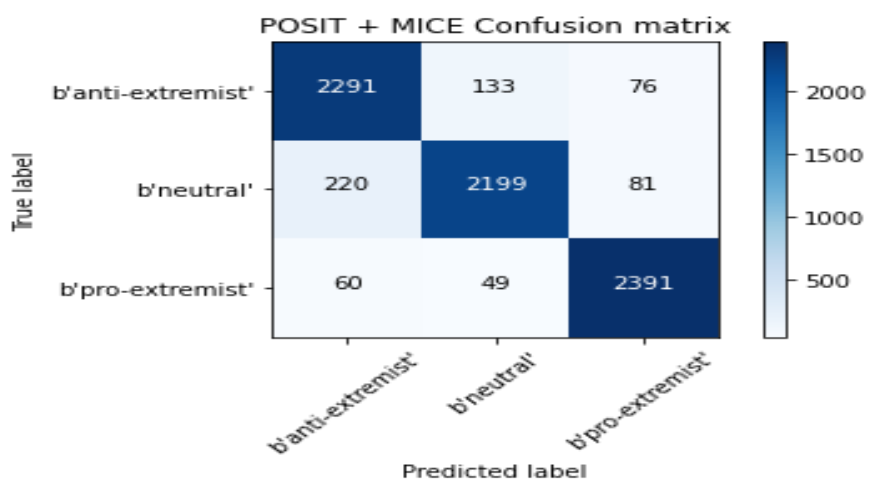


Figure 4.52: Composite Confusion Matrix

4.8 The Validation of Nigerian Extremism Dataset

Each of the textual analysis technique was applied on the Nigerian extremist Webpages, where Posit produces a 27- item feature list for each category of Webpage, Sentiment

analysis generates 26, Composite produces 53 and Extended Posit produces 71 - items features. We used the whole ICCRC dataset as a training dataset while the Nigeria dataset as a validation set. We utilised sklearn’s GridSearchCV to perform an exhaustive search over specified parameter values for an estimator. The estimators are the classifier algorithms. The CV option was set to 7 for the gridsearchCV. The grid parameters used are presented in Table 4.42 below:

Model	HyperParameters	Range
J48 Decision Tree	Criterion	Entropy
	max_depth	1 – 21
	min_samples_leaf	2-10
Random Forest	Criterion	Entropy
	max_depth	1 – 21
	min_samples_leaf	2-10

Table 4.42: The Model Parameters

ICCRC dataset was supplied as the training dataset for the fit method. The fit method is to perform model fitting using the set of parameters supplied. The Nigeria dataset was supplied for the prediction (Model validation). The prediction will use the best parameters from the gridsearchCV to predict the class. The classification results are presented below:

4.9 J48 Classification Results

4.9.1 Posit-Based classification Results

The application of grid search yielded the following best parameter sets used for the validation of Posit using J48 ('criterion': 'entropy', 'max_depth': 16, 'min_samples_leaf':

2). J48 gave overall classification accuracy of 48.5%. Pro-extremist Webpages were correctly identified at the rate of 31%. The results of other categories are detailed in Table 4.43 and Figure 4.54. The data plots of the training are displayed in Figure 4.53.

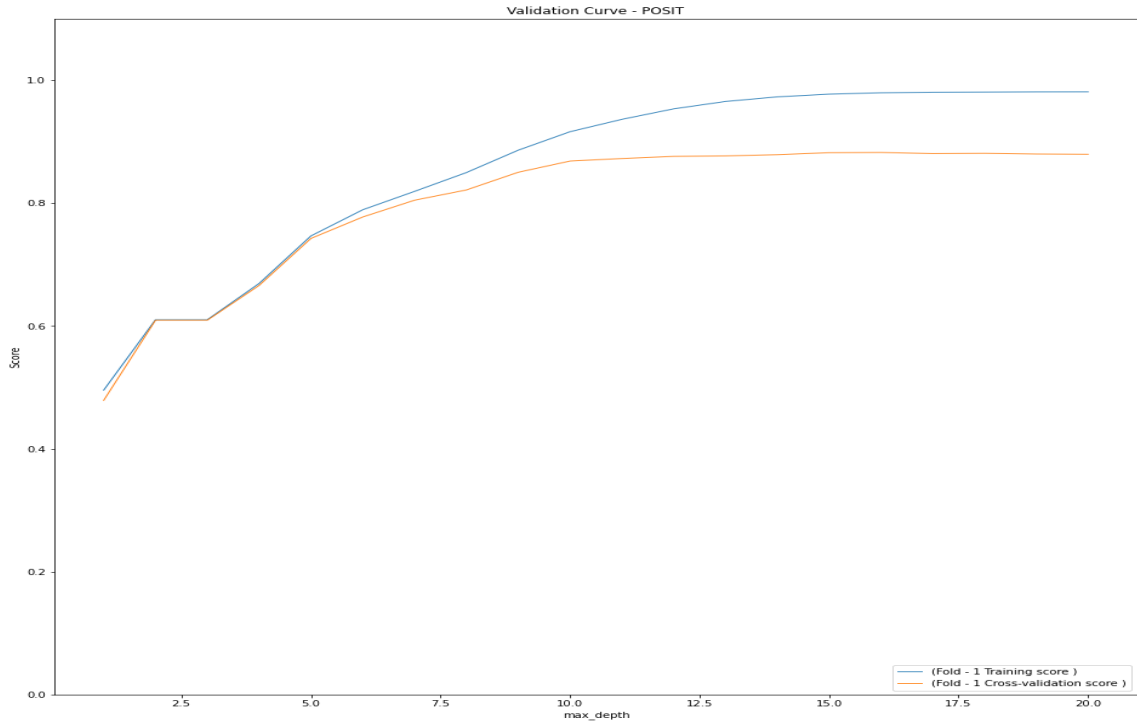


Figure 4.53: Validation Curve-Posit

False Positives Rate	Recall	Precision	F1	Class
0.371	0.529	0.416	0.465	Anti-Extremist
0.220	0.566	0.627	0.513	Neutral
0.183	0.316	0.391	0.350	Pro-Extremist

Table 4.43: Posit-Based Classification Result using J48

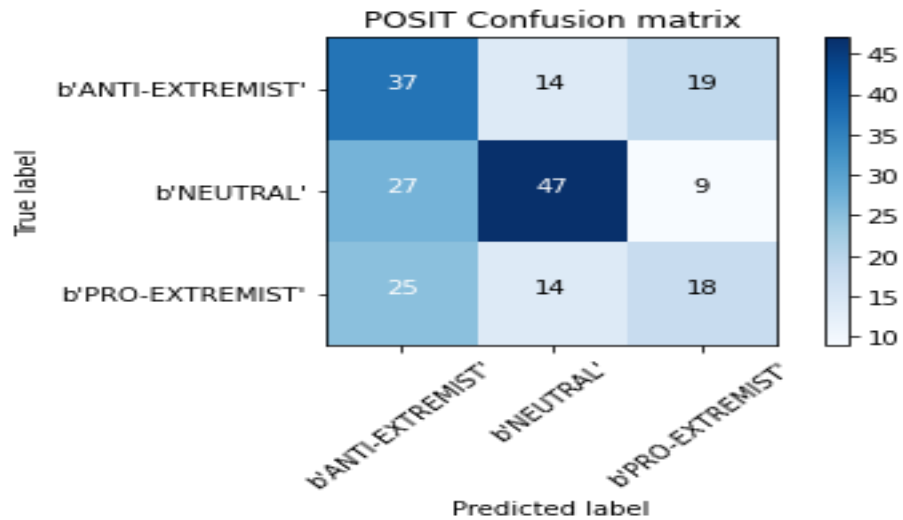


Figure 4.54: Posit Confusion Matrix

4.9.2 Extended-Posit Based Classification

The application of grid search yielded the following best parameter sets used for the validation of Posit using J48 (criterion: 'entropy', 'max_depth': 12, 'min_samples_leaf': 3). The J48 decision tree finished with an overall classification accuracy of 49% with the pro-extremist category classified at 41%. The validation curve is displayed in Figure 4.55. The results are detailed in Table 4.44 and Figure 4.56 respectively.

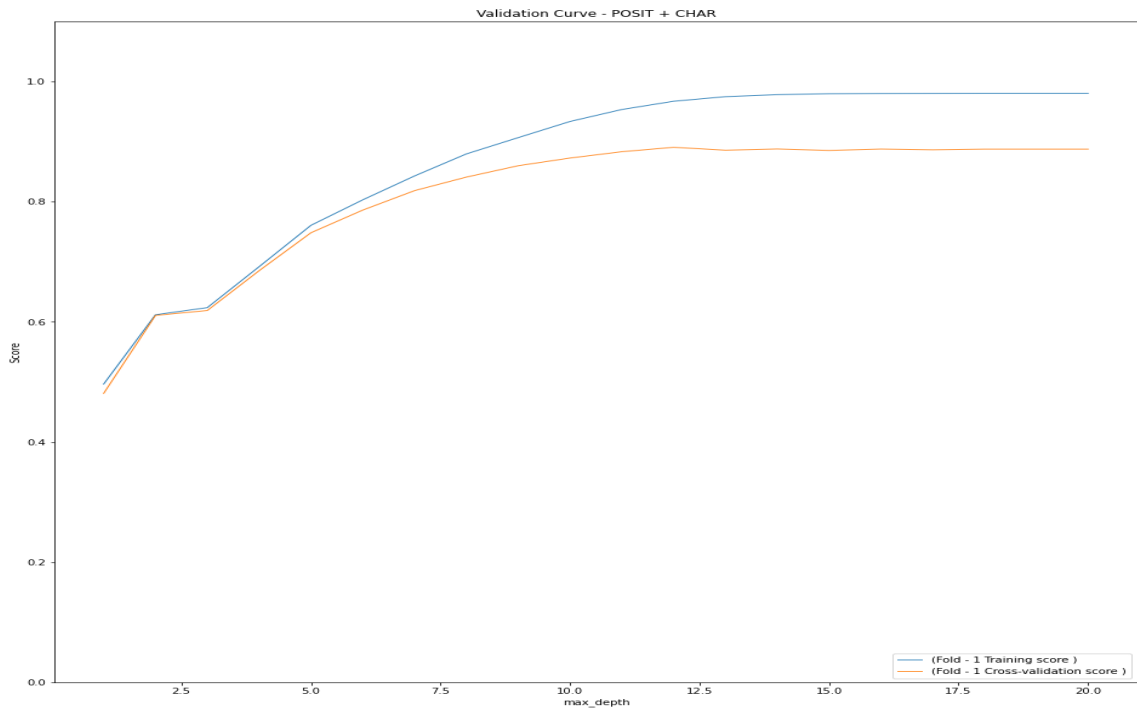


Figure 4.55: Validation Curve-Extended Posit

False Positives Rate	Recall	Precision	F1	Class
0.158	0.423	0.577	0.488	Anti-Extremist
0.410	0.652	0.422	0.512	Neutral
0.190	0.411	0.536	0.465	Pro-Extremist

Table 4.44: Extended Posit-Based Classification Result using J48

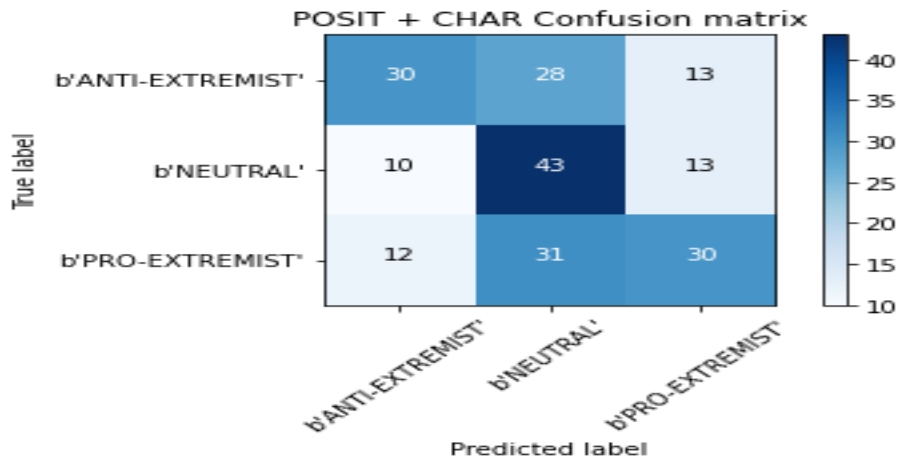


Figure 4.56: Extended-Posit Confusion Matrix

4.10 Random Forest Classification Results

4.10.1 Posit Classification Based Results

The optimal performance for the Posit can only be found at (criterion': 'entropy', 'max_depth': 17, 'min_samples_leaf': 2) with the application of gridsearchcv. Random forest gave 53% overall classification accuracy in validating Nigerian data. The model correctly identified pro-extremist cases at 41%. The results are detailed in Table 4.45 and the confusion matrix in Figure 4.58. The variance between each validation and training score indicates minimal overfitting. The data plot showing the performance is presented in Figure 4.57

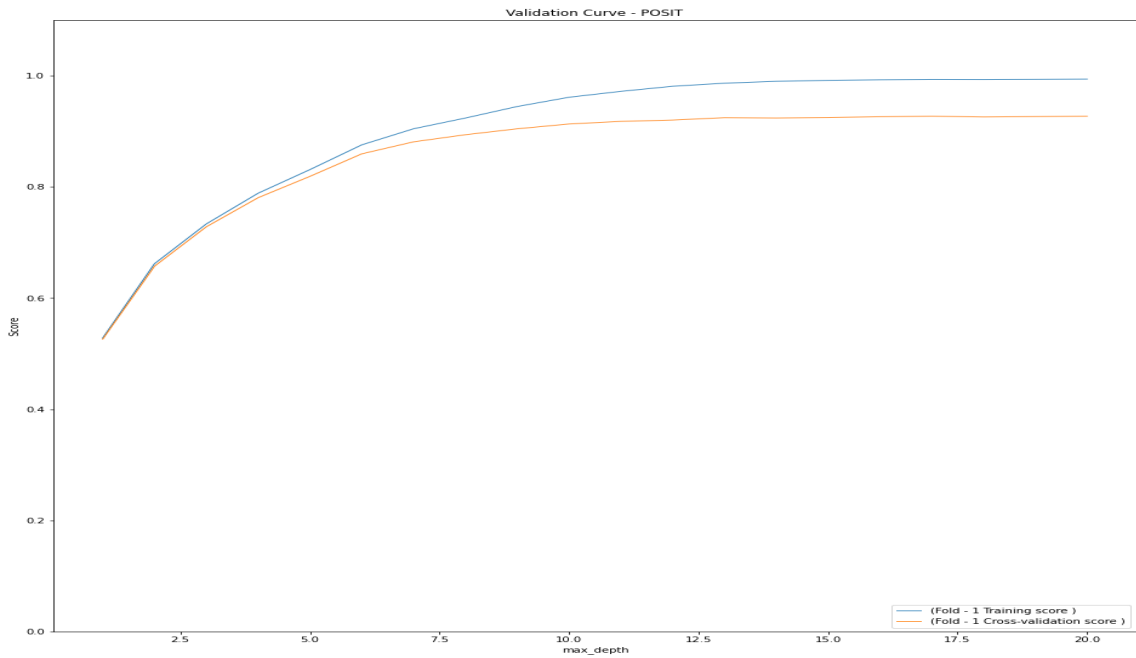


Figure 4.57: Validation Curve- Posit

False Positives Rate	Recall	Precision	F1	Class
0.199	0.338	0.481	0.397	Anti-Extremist
0.336	0.836	0.570	0.678	Neutral
0.170	0.413	0.510	0.456	Pro-Extremist

Table 4.45: Posit-Based Classification Result using Random Forest

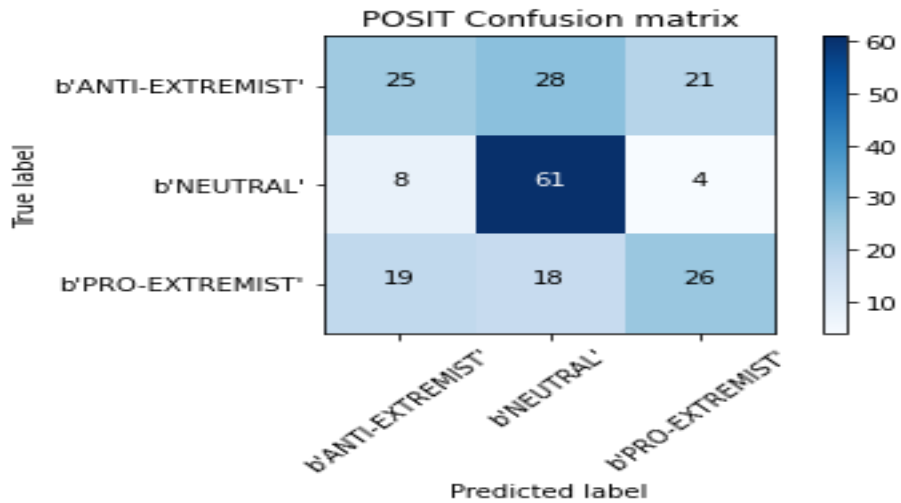


Figure 4.58: Posit Confusion Matrix

4.10.2 Extended Posit-Based Classification Result

The grid search results for Extended Posit-Based indicated that the model required a range of 17 max depth levels in order to accomplish the best performance, with a constant minimum sample leaf of 2. The Random forest gave 53% overall Webpages matching with Nigerian data. Pro-extremist cases were identified at the rate of 48.6%. The classification result and the confusion matrix are presented in Tables 4.46 and Figure 4.60 below. Figure 4.59 presented the validation and training curves which indicate the model is well fitted.

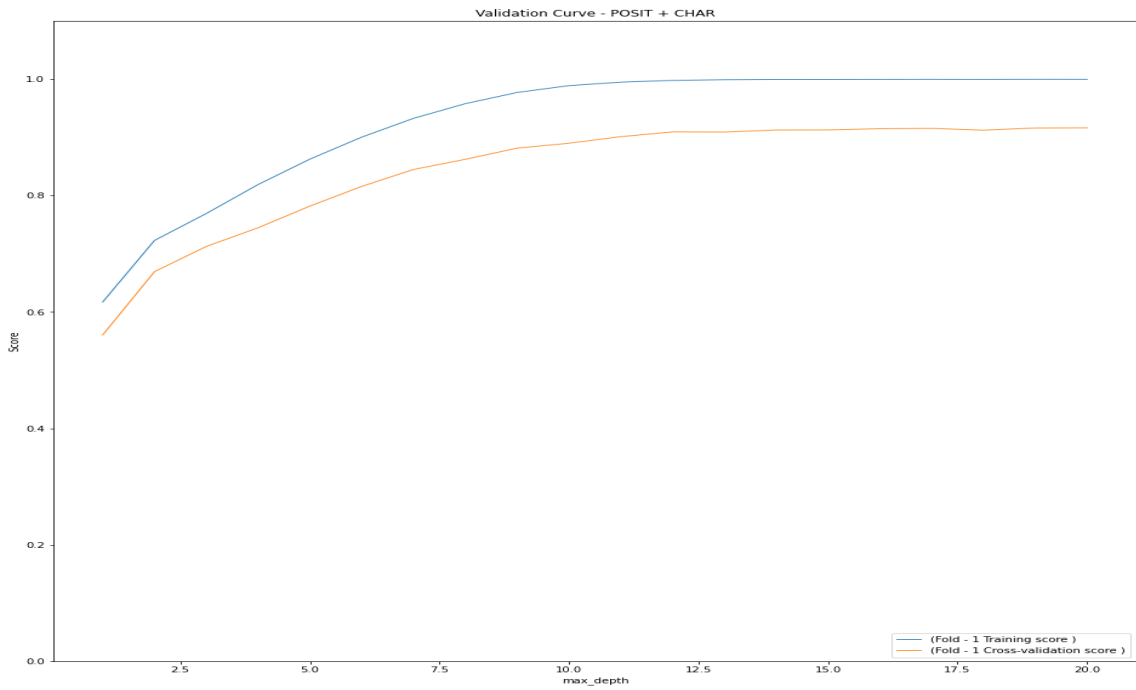


Figure 4.59: Validation Curve-Extended Posit

False Positives Rate	Recall	Precision	F1	Class
0.146	0.411	0.600	0.488	Anti-Extremist
0.448	0.716	0.429	0.536	Neutral
0.101	0.486	0.708	0.576	Pro-Extremist

Table 4.46: Extended Posit-Based Classification Result using J48

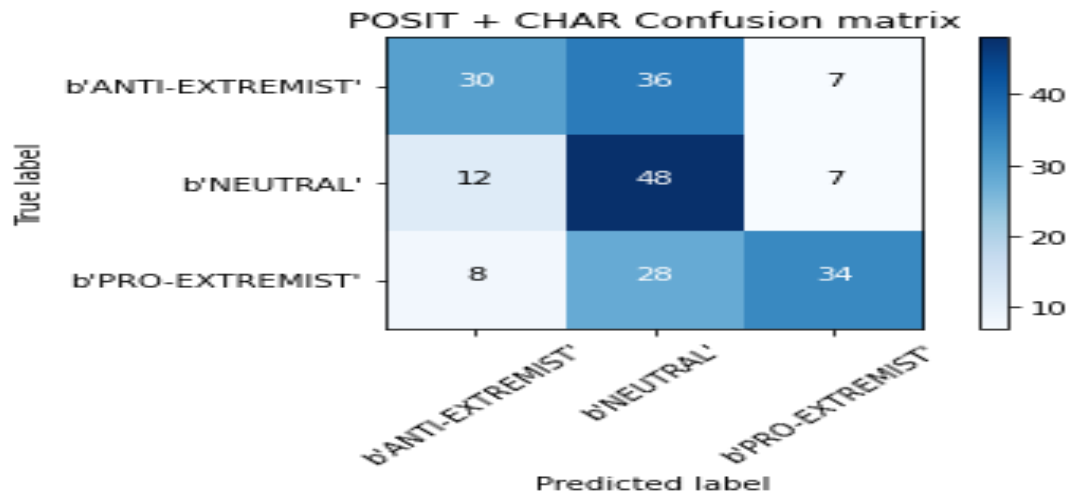


Figure 4.60: Extended-Posit Confusion Matrix

Chapter 5: Neural Networks

This chapter details the implementation of different Neural Network models explored in this thesis

5.1 Neural Network

Neural networks are a form of a machine learning algorithm that mimics how the brain operates, which is regarded as artificial neural networks. Learning can be semi-supervised, unsupervised or supervised [93]. Initially, neural networks were made up of three layers which consisted of input, output and hidden layers. However, once the hidden layer is greater than one, it becomes a deep-learning network unlike a traditional neural network such as the first perceptron comprising of one input and one output layer, and one hidden layer in between. This is a significant factor in distinguishing it from other single-hidden-layer neural networks [94]. Figure 5.1 shows a typical Neural Network. The algorithm is trained to learn and identify the pattern of features automatically by reconstructing samples from labelled data. In the process, the neural networks learn to identify correlations between specific relevant features and best results. It brings out relationships between feature signals and their representation. This is then applied to unstructured or unlabelled data, giving it access to use the same pattern on it for higher performance [93]. The neural network models considered in this work are Multilayer perceptron (MLP) and the Recurrent Neural Networks (RNN), they are explored due to their capabilities for complex classification tasks.

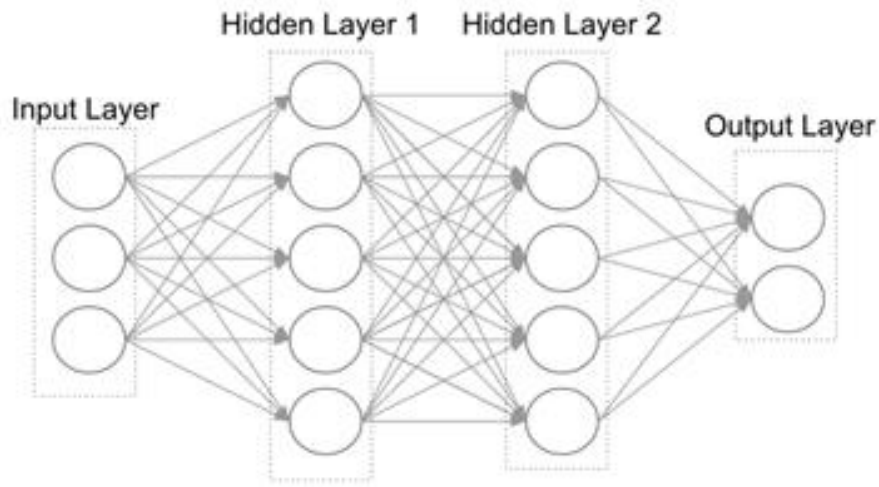


Figure 5.1: A Neural Network with 2 Hidden Layers

5.1.1 Multilayer perceptron (MLP)

The multilayer perceptron (MLP) consists of an input layer that receives the signal, an output layer that decides or predicts based on the input, and any number of hidden layers in between that act as the MLP's true computational engine [95]. Any continuous function can be approximated by MLPs with one hidden layer. In supervised learning problems, multilayer perceptrons are widely utilised. They have the ability to predict the correlations (or dependencies) between the inputs and outputs after being trained on a collection of input-output pairings [96]. In order to reduce error, the model's parameters, or weights and biases, are modified over the course of training. Backpropagation is utilized to perform the weight and bias modifications linked to the error [97]. To reduce the loss, inputs are multiplied by weights before being transmitted to the activation function, where they are modified in back propagation. The machine-learned values from neural networks are used as weights [96]. They self-adjust based on the difference between training inputs and projected outputs. After nonlinear activation functions, Softmax is used as an output layer activation function [98].

5.1.2 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are a form of neural network which simulate or forecast time series or sequence data [99]. RNN behavior is similar to human brain activity. Convolutional and feedforward neural networks (CNNs), like recurrent neural networks (RNNs), learn from training data [99]. Recurrent networks differ from other types of networks in that all of their layers share the same properties. Recurrent neural networks use the same weight parameter in each layer, as opposed to feedforward neural networks, which use different weights for each node [100]. To aid reinforcement learning, these weights are adjusted using gradient descent and backpropagation techniques [100]. Backpropagation through time (BPTT) is a technique used by recurrent neural networks to estimate gradients because it is specialized in sequence data, and it differs significantly from regular backpropagation [100].

Convectional backpropagation, a technique similar to BPTT [99], is used to train the model by computing errors from its output unit to its input layer. Using these computations, we can adjust and fit the model's parameters. Whereas feedforward networks do not, BPTT accumulates faults at each time step in contrast to the standard method. RNN frequently encounter exploding gradients and vanishing gradients during this process. The gradient, is the slope of the loss function along the error curve, determines the magnitude of these challenges. When the gradient becomes too small, it drops even further, updating the weight parameters until they are irrelevant or zero [100]. Hence, the algorithm then halts learning when this happens. Exploding gradients, which happen when a gradient is too large, lead to an unstable model. The model weights in this instance grow out of control and eventually take the form of NaN. One solution to address these issues is to reduce the number of hidden layers in the neural network [99]. When the gradient values are too small, vanishing gradients occur, which causes learning to stop or take too long. This was a major issue in the 1990s, and it was far more difficult to solve than the problem of exploding gradients.

Fortunately, the problem is resolved by utilising the LSTM idea [100]. The information flow differences between an RNN and a feed-forward neural network are shown in Figure 5.2 below.

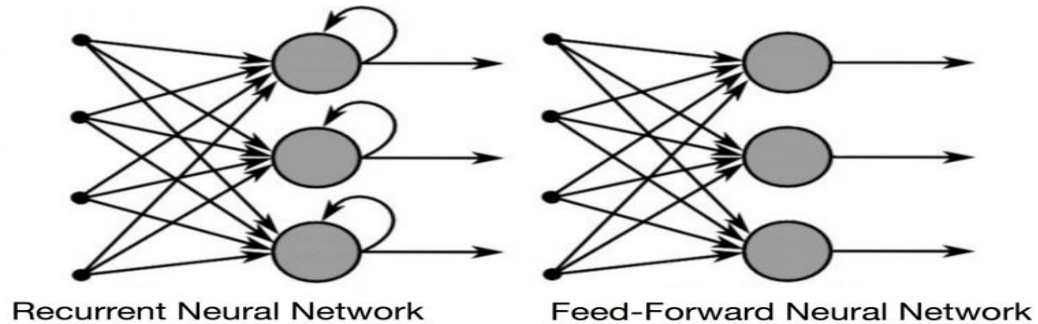


Figure 5.2: Recurrent Neural Network and a Feed-Forward Neural Network [100]

The Variant of RNN Architectures

In this thesis, LSTMs are used as the building blocks for the layers of an RNN [100]. LSTMs assign data "weights" that help RNNs decide whether to accept new information, ignore it, or give it enough relevance to improve the output. Furthermore, the LSTM model exhibits significantly more volatility throughout its gradient descent than the GRU model [101]. This could be because there are more gates for the gradients to pass through after a certain number of epochs, making constant progress more difficult to achieve. GRU uses fewer training parameters, requiring less memory and implementing faster than LSTM. Contrarily, LSTM is more accurate, particularly for larger datasets [102].

Long Short-Term Memory (LSTM)

The short-term memory problem in RNN models can be fixed with the use of an RNN architecture called LSTM. Input, forget, and output gates are the components of an

LSTM. These gates regulate the data flow to forecast the network's output. These gates determine whether additional input is allowed (input gate) if the information is removed (forget gate) and whether the output at the current time step is affected [102]. Figure 5.4 below displays an RNN with its three gates in place. The LSTM's analog gates have sigmoidal shapes and range from zero to one. They are analog, therefore they may conduct backpropagation. . The problem of disappearing gradients is overcome by LSTM since it keeps the gradients steep enough, resulting in a fast training time and good accuracy [101].

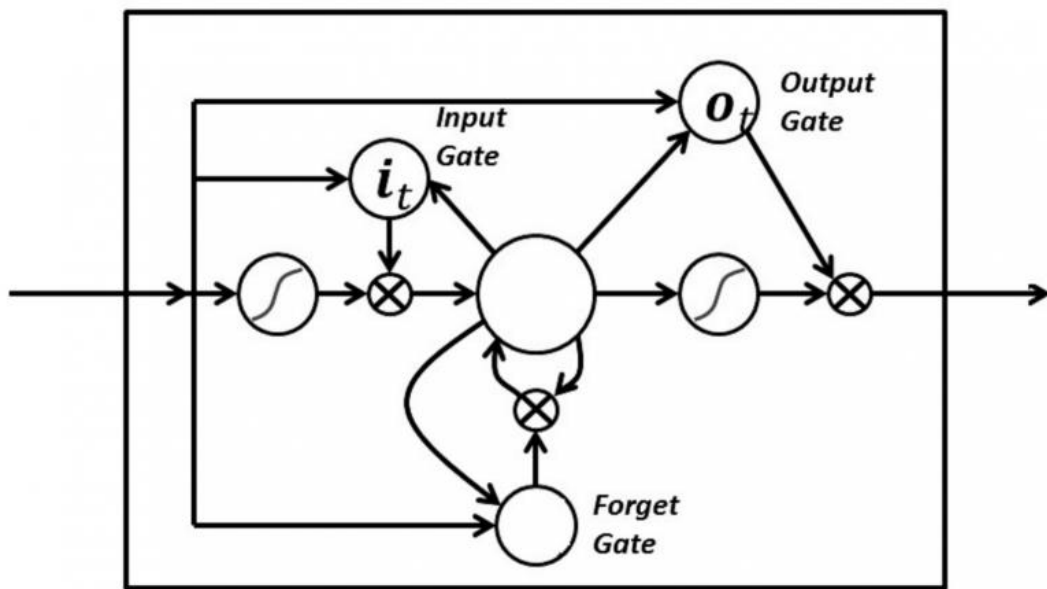


Figure 5.4: An illustration of a RNN with its three gates [99]

5.2 Optimisation Algorithm

The most popular method for optimizing neural networks is gradient descent, a first-order iterative optimization technique that identifies the parameters that minimize the loss function (prediction error) and uses a backpropagation of error process to update the weights [96]. The gradient descent algorithm tries to modify the weights to reduce the likelihood of an error in the evaluation that comes after (this implies that the

optimisation algorithm is going down the slope of error). Figure 5.5 below shows how the gradient descent method descends the derivative to produce the minimum.

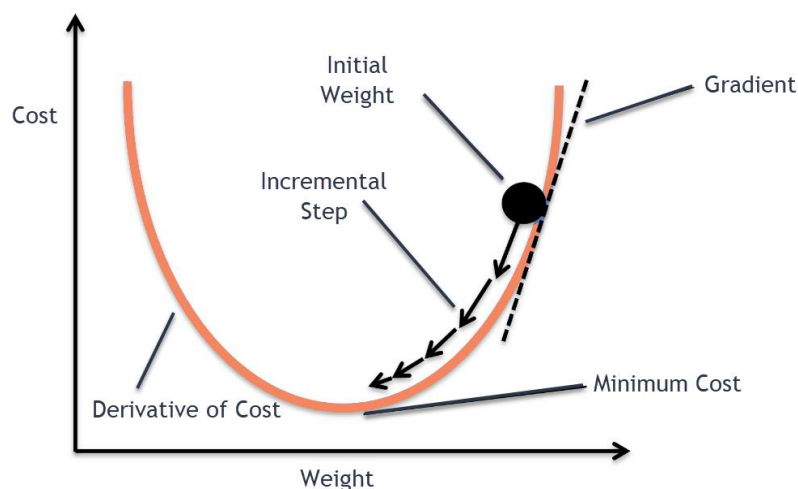


Figure 5.5: Gradient Descent Algorithm [103]

The limitation of using gradient descent is their hyper parameters that needed to be specified in advance, which depend greatly on the type of model and problem. Another disadvantage of gradient descent is that a similar learning rate is applied to all parameter updates, this may affect a situation of having sparse data, where it might be important to update the parameters in an unusual extent instead [104]. However, adaptive gradient descent algorithms such as Adagrad, Adadelta, RMSprop and Adam, produce a better alternative to traditional gradient descent. They contain per-parameter learning rate techniques, which provide a heuristic method without the need for the costly task of tuning hyperparameters for the learning rate arranged manually. Both Adagrad and Adam provide better results than gradient descent, but Adam is faster than Adagrad [104]. For this reason, the Adaptive Moment Optimisation algorithm is discussed further below.

5.2.1 ADAM

The Adaptive Moment Optimisation algorithm combines the methods of both Momentum and RMSProp [98]. The equations are stated below:

$$V_t = \beta_1 * V_{t-1} - (1 - \beta_1) * g_t \quad \text{Equation 1}$$

$$S_t = \beta_2 * S_{t-1} - (1 - \beta_2) * g_t^2 \quad \text{Equation 2}$$

$$\Delta\omega_t = -\dot{\eta} \frac{V_t}{\sqrt{S_t + \epsilon}} * g_t \quad \text{Equation 3}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t \quad \text{Equation 4}$$

$\dot{\eta}$: Initial Learning rate

g_t : Gradient at time t along ω^j

V_t : Exponential Average of gradients along ω^j

S_t : Exponential Average of squares of gradients along ω^j

β_1, β_2 : Hyperparameters.

From the equation stated above, the exponential average of the gradient with the squares of the gradient for each parameter is calculated in (Equations 1 and 2) [104]. To arrive at the learning step, the average of the gradient is multiplied by the learning rates and divided by the root mean square of the exponential average of the square of gradients in equation 3. Then, an update is added. The threshold for hyperparameter beta1 is 0.9 while beta2 is at 0.99. Epsilon is generally taken to be 1e-10.

5.3 Loss Function

In the context of an optimization algorithm, a loss function rates the results of a machine learning algorithm. The loss function estimates the error for one training set while the cost function is the average of the loss functions for the training set [104]. Whenever

an optimization method is used, the value assessed by the loss function is known as "loss" [105]. As a result, a loss function must be selected when estimating the model's error throughout the optimisation phase [106]. To determine what functions to use can be challenging, it is expected that the function should faithfully represent the design goals. The maximum likelihood estimation is a widely used method in the field of machine learning to determine the error of a set of weights in a neural network [107].

However, under the maximum likelihood framework, the error between two probability distributions is computed using cross-entropy [108]. Cross-entropy is employed as the loss function between the training data and the model's predictions when the selected parametric models specify a distribution $p(y|x)$; [36]. Cross-entropy evaluates the variance between estimated and predicted probability distributions in a given classification problem, while in regression problems, mean squared error (MSE) is used as the loss function [106]. The data distribution and the model distribution are frequently compared using cross-entropy [108]. The cross-entropy function is determined by the method used to describe the result. Under a framework of maximum likelihood, the default activation for the output layer is the softmax activation function while the lost function is Cross-Entropy (also referred to as Logarithmic) [108].

5.4 Activation Function

The activation function's job is to put values back into a controllable, acceptable range as they are transferred to the next layer. The activation function is linked to the signal's forward propagation across the network [109].

The input layer node does not carry the activation function; only the hidden and output layer nodes do [109]. For a certain input to produce a certain output or to activate a node, an activation function is necessary. It helps to determine and activate the nodes that contribute immensely to the required outcome depending on the final summation assigned. There are several types of activation functions some of which include but not limited to linear activation function (Sigmoid and hyperbolic tangent activation

functions), and nonlinear activation function such as, Relu. The activation function in a neural network transforms the node's summed weighted input into the activation of the output for that input [109]. The activation function keeps the values passed on to the next layers within an acceptable and practical range and passes the output [110].

5.5 Regularisation

A model that performs exceptionally well on training data yet poorly predicts test data is said to be overfit. During the training of a large network which could be challenging, there is a particular point during training at which the model end generalising and begins to learn statistical noise in the training data, hence overfitting sets in and creates an increase generalisation errors and makes the model inefficient at making predictions on the new data. Underfitting occurs when machine learning algorithms are unable to contain a basic pattern in the data or adequately fit the data. Avoiding overfitting can lead to improvement in a model's performance. Figure 5.6 below shows the structures of underfitting, overfitting and a good fitting in a neural network.

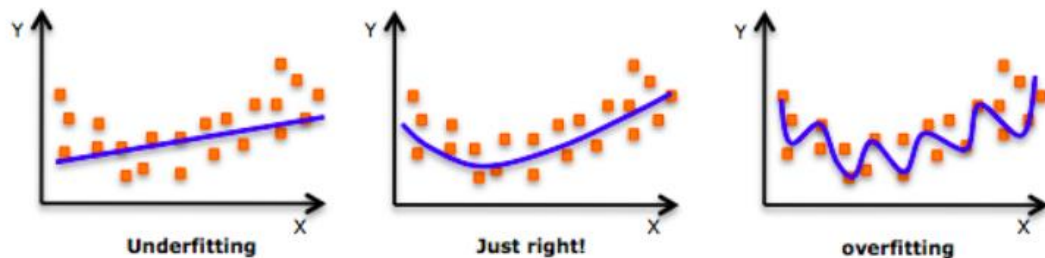


Figure 5.6: Structure of different training sets [39]

Exploring a regularisation technique is one solution to this problem. Regularisation is a method that modifies the learning algorithm to improve the model's generalisation and performance on new data [111].

5.5.1 Dropout

This is an interesting regularisation technique that is mostly studied in the field of deep learning. At a learning phase of a neural network, weights of neurons usually search for specific features providing some specialisation. The nearest neurons rely on this specialisation of which if considered too far can lead to a fragile model that specialises excess on training data. However, dropout is applied to avoid this drawback, randomly chosen neurons are disregarded during learning. As a result of this, they do not contribution whatsoever to the activating neurons. At the first phase of the system, the downstream neurons will be temporarily removed whilst the weight will no longer be considered for an update during the backward pass. Whenever the neuron is randomly dropped out of the network in the course of training, other neurons take the duty of handling the representation needed to make predictions for the missing neurons. This allows the networks to learn multiple independent internal representations, be less sensitivity to specific neuron weights, and better generalisation that minimises the rate of overfit [112].

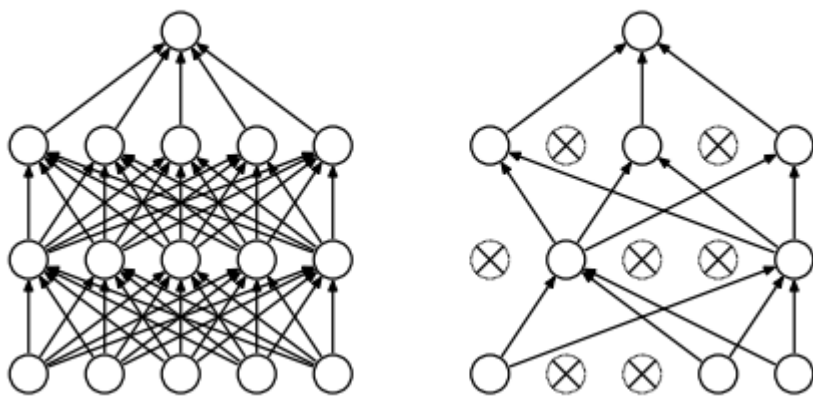


Figure 5.7: (a) Normal Neural Network (b) After Applying Dropout [112]

Figure 5.7 (a) depicts a standard neural net with two hidden layers, whereas Figure (b) depicts a thinned net formed after applying dropout to the network.

Dropout can be employed in both the input and hidden layers, as shown in the aforementioned diagrams. This is done by arbitrarily choosing some nodes and removing them along with their incoming and outgoing connections, as shown in the schematics above. This section entails a distinct set of nodes leading to a different set of output since the nodes were randomly chosen. The dropout function's hyperparameter is the likelihood of selecting the number of nodes to be dropped

5.5.2 Early Stopping

Early stopping is a form of the regularisation method that enables a random large number of training epochs to be defined and stop the training when the model performance degenerates on a validation dataset. A trigger could be initiated, and the training process will stop. The Early Stopping callback is triggered when instantiated through arguments. The performance of the model can be detected during training by setting out the evaluation metrics to be used on a choice of a dataset. It is usual practice to divide the dataset and utilize a subset as a validation dataset, which is not used to train the model but is used to evaluate the model's performance during training. This is achieved by using the loss on a validation dataset, which is popularly used as a metric or as a performance measure to monitor the performance during training. Part of the training procedure includes the loss of the model on the training dataset and other metrics can be evaluated and monitored on the training dataset. At the end of each epoch, the performance of the model is calculated on the validation set. The performance of the model on the validation set, such as loss is used to determine when a trigger is decided concerning when to stop training in the early stopping trigger method [113]. The Early Stopping callback will halt the training when activated.

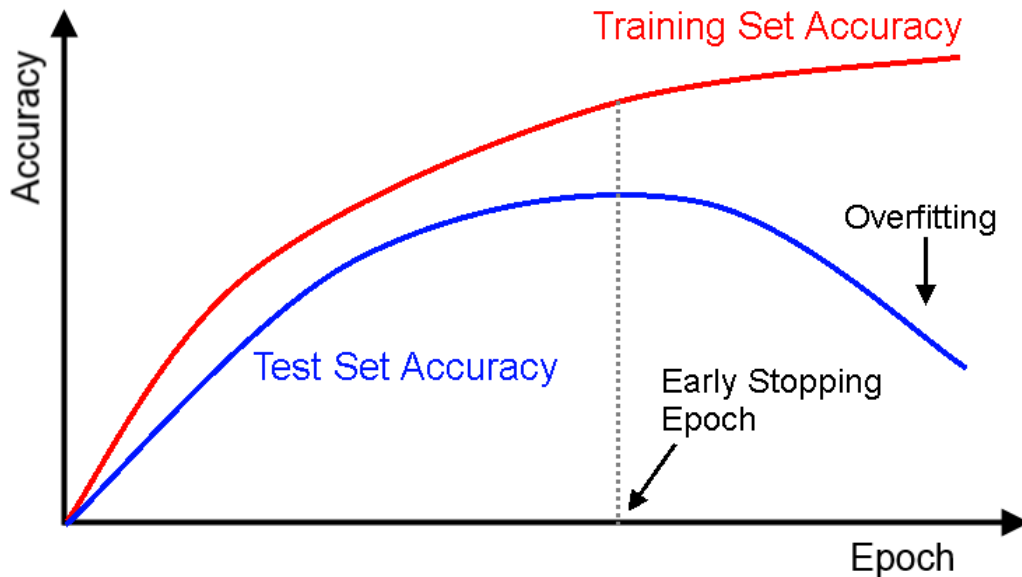


Figure 5.8: Training Set Accuracy Source [113]

5.6 Google Colaboratory

Google Colaboratory is an open-source cloud service tool for machine learning with Jupyter notebook settings which requires no setup to use. Google Colaboratory allows sharing Jupyter notebooks without the stress of downloading, or installing anything on your computer, using only a browser. In addition, it comprises most of the libraries such as Scikit-learn, TensorFlow, Matplotlib and dependencies already installed. The advantage of Google Colab is its open-source GPU service. Google Colaboratory provides a runtime completely developed for deep learning and open access to a robust GPU. On a free Tesla K80 GPU, deep learning applications can be designed with Google Colaboratory using Tensorflow, Keras and PyTorch [114].

5.7 TensorFlow

TensorFlow is a Python library developed and released by Google, it enhances quick numerical computing. It is a primary library that explores data flow graphs to develop deep learning models directly or by using wrapper libraries that resolve the process

developed on top of TensorFlow [115]. TensorFlow is employed for classification, and the learning algorithms are designed for pattern recognition and knowledge-based prediction using sensory data and an artificial network structure of nodes and weights. The network structure is commonly constructed with an input layer, one or more hidden layers, and an output layer, with each layer consisting of many nodes linked together [116].

5.8 Validation Metric

The error rates of the machine learning model are obtained through the validation technique. When setting up a classification model, over-fitting of the training set is one of the common problems to avoid. Over-fitting means there is a good level of accuracy in the training set but this drop significantly in a given new dataset. To avoid over-fitting, a train, test and validation split technique is used. Hence, a split of 70% for the training set, 30% for the test set and 33% for the test set is used to evaluate our model. This distribution helped to know how well the model will perform on any new given inputs, to avoid the model being biased. The training set refers to a sample of data used to fit the model and the testing set means a sample of data used to produce an unbiased judgment of a final model fit on the training set. A validation dataset is a portion of data kept off from training a model which gives an assessment of model performance while tweaking the model's hyperparameters.

Another approach to consider would be to apply k-folds cross-validation but it is seldom used for evaluation in the Neural Network model due to its larger computational expense; it involves number of models to be constructed and evaluated, which substantially adds to the evaluation time of the model [117]. In order to assess the performance of this classification, the metrics considered are precision, recall, f-measure and accuracy.

5.9 TensorFlow Implementation

This section describes the implementation of the experimental setup of each classification framework described in the previous chapter. As a reminder, the classification frameworks in question, are Sentiment, Posit (features on the basis of word-level information), Extended Posit (features on the basis of both word and character-level information) and the proposed framework, composite-based classification method. The feature sets considered in this section are (i) the ‘default’ 27 Posit features, (ii) the extension of Posit to include 44 character features (referred to as extended-Posit features), (iii) sentiment features and composite features (a mix of sentiment and syntactic features derived from the textual data). All of the features in these different sets were extracted from the three predefined categories of extremist Websites (with 2500 Webpages in each category). Each feature set was employed to determine their degree of effectiveness in classification, via two Neural Network Models algorithms such as the MLP and RNN.

In our approach, we decided to build the Neural Network model in each framework’s output data based on the content of the 7500 Webpages, without any further pre-processing. TensorFlow is employed to implement the Neural Network classifiers. TensorFlow is a machine learning library [115]. The data set was loaded into TensorFlow to run Neural Network classification models and utilised each model to predict the category for each Webpage. A big data collection of 7500 Webpages was used to develop the classification models in order to build the TensorFlow model. The greater the amount of data collected for training a model, the higher the accuracy should be. The manual data sets were merged into the excel, after which a class label column “category” was defined, denoting whether the data represented “pro-extremist” or “anti-extremist” or “neutral” Webpages. The model was tested for its accuracy in identifying class values for the “pro-extremist” or “anti-extremist” or “neutral” Webpages category.

However, a class of anti-extremist is labelled as 0, neutral as 1 and pro-extremist as 2. To make the training process well behave, the features were scaled using SciKit learn StandardScaler. TensorFlow was used in the experiment with various settings for the parameters relevant to the number of partitions, epochs, layers, learning rate, and regularisation. In respect to regularisation as a measure to avoid over-fitting, a dropout technique was employed. This is the probability of choosing number of nodes to be dropped in the dropout layer.

Dropout is incorporated in the model to perform back propagation and regularisation functions. Cross-entropy was employed to calculate the loss and the ADAM for the optimiser used in updating the model. During the training of the model, too many epochs can lead to overfitting and fewer epochs can led to underfitting of the model. However, early stopping is a part of the regularisation method explored that permits an arbitrary huge number of training epochs to be specified and halt the training once the model performance stops improving on a validation dataset.

We start by feeding a data point into the input layer using our dataset. The data is then routed through the hidden layer or layers, where weights and biases are added. Then the output layer classifies the result from the hidden layer, which eventually produces the output of extremist Web data. The architecture of the each model consists of hidden layer, an input layer which represent the number of each framework's features (for example, input shape size is 27 in Posit, 71 in Extended-Posit, 26 in sentiment and 53 in Composite features) and an output layer of 3, which represent anti-extremist, neutral and pro-extremist (the prediction class). The hidden layers transform inputs to output size using activation functions. Further architecture and hyperparameters of each model is detailed below.

5.9.1 The MLP (multi-layer perceptron)

Our MLP comprises of 3 dense layers, with each having 512 hidden neurons each with a relu activation function. Each dense layer has a follow-up dropout layer with rate of 0.3 to tackle overfitting tendencies. The final output layer is armed with a softmax activation function to process outputs between 0 and 1. Our final layer in the compilation layer which hosts the loss (categorical cross entropy) function, metric and the optimizer (adams).

5.9.2 RNN

The RNN is built to receive data in both sequential and vectorised format hence our data which was originally in a sequential format after acquisition was vectorised in the data pre-processing stage where text feature representation was implemented. Our RNN topology comprises of 2 LSTM layers, one which is the input layer with 26 nodes to receive the features. The other layer contains 150 hidden units for cycle/loop processing within the network. The tanh activation function is used to received the outputs from one of the LSTM layers which serves as input. The following linear activation functions are used for complex function mapping, they include the ‘sigmoid’ (maps inputs to outputs between 0.0 and 1.0) function and the hyperbolic tangent or tanh function (maps inputs into outputs between -1 and 1) while the ReLu maps continuous outputs as in the cases of regression modelling and analysis. The ‘adam’ optimizer was selected specifically for the sake of gradient descent and backpropagation which updates the weights and reduces the loss as accuracy and learning improves. Our loss choice is the ‘sparse categorical cross-entropy’. The tunable dropout layer is simply meant to reduce overfitting tendencies during modelling.

5.9.3 Early Stopping

Two different Neural Network models were used to predict the category of each Webpage in each of the framework. During the implementation each Neural Network model on each feature sets, early stopping is applied to avoid overfitting by preventing

many iterations. The early stopping is designed to cease the training at the point when validation loss starts to plateau but the initial indication of no more improvement may not be the perfect time to end training because the model may perhaps get worse before substantially getting better. However, a modification is done by setting a delay to the trigger for the number of epochs after which no progress is expected. This is achieved by setting the “patience” argument callback to 3, instructing the system not to stop till it reaches certain epoch threshold once a validation loss of a given test dataset degenerated (after three different plateau) and no point in continuing training. “Baseline” argument is set up to achieve this task and the epoch threshold was set to 20. Hence, if there are no changes to performance after 3 runs then it will stop. Eventually, an additional callback, known as ModelCheckpoint takes the snapshot of the system at each epoch and saves the best model observed during training. The training was done on GPU using Google Colab platform.

5.10 Classification Results

This section describes the results obtained from the TensorFlow implementation on Posit, Extended-Posit, Sentiment and Composite feature sets of the extremist Webpages. The results interpreted in this section are largely focused on the overall classification and particularly, the pro-extremist class. The results generated from the TensorFlow implementation are the different parameters for the number of partitions, epochs, layers, learning rate, and regularisation which were tested for optimum accuracy for each model and the best epoch were reported in the result classification section. The regularisation process envisioned with data plots is also presented. The parameters of each run and the corresponding results are also presented.

5.11 RNN Classification Results

This section describes the results obtained from Posit, Extended-Posit, Sentiment and Composite feature sets of the extremist Webpages using RNN Model

5.11.1 Sentiment-Based Framework (MF Imputation)

One of the most important traits of a healthy training curve is convergence, we achieved many convergences where the model produced the best generalisation performance on both test and training sets at epoch 20. The data plot of the learning process of the model indicated that the model does not overfit as the loss in both validation and training decreases significantly at 0.48 and 0.45 respectively at epoch 20. Figures 5.9-5.10 illustrate the data plots on the graphs. Overall classification of Webpages is recorded at 79.5%. Table 5.1 details the results of other classes. From the confusion matrix in Figure 5.11, the pro-extremist category had the highest level of correctly classified cases. The class produces the lowest false positive rate, at 89% and 6% respectively.

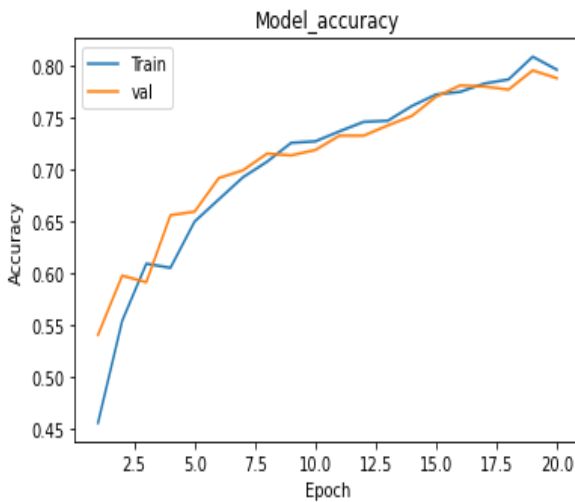


Figure 5.9: Model Accuracy Curve-MF

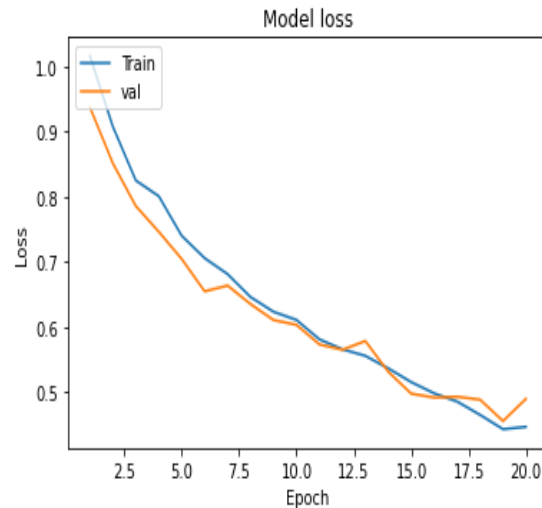


Figure 5.10: Model loss Curve-MF

Imputation

Imputation

False Positives Rate	Recall	Precision	F1	Class

False Positives Rate	Recall	Precision	F1	Class
0.187	0.858	0.703	0.773	Anti-extremist
0.062	0.634	0.833	0.72	Normal
0.06	0.89	0.88	0.885	Pro-extremist

Table 5.1: MF Imputation Classification Result using RNN

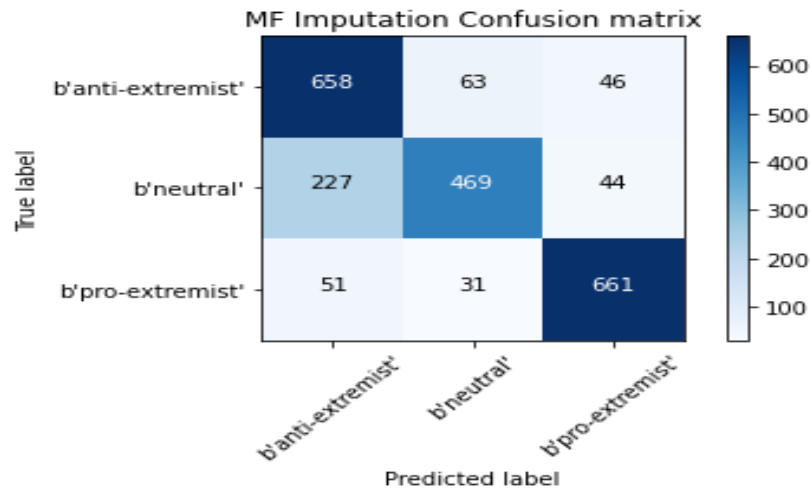


Figure 5.11: MF Imputation Confusion Matrix

5.11.2 KNN Imputation

The result obtained from KNN imputation is an improvement of MF imputation as we achieved an early convergence on the training curve. These can be seen in Figures 5.12 and 5.13 which are also reflected in the loss. The best performance of the model was recorded at epoch 20. The data plot indicated a good fit model as both validation and training loss decrease to 0.41 and 0.39 respectively at epoch 20. The model was able to

classify 81% of the extremist Webpages across the three categories. Pro-extremist had the highest correctly classified cases at 84% with the lowest false positive rate of 10 % compared to other categories in the confusion matrix, Figure 5.14. The result of applying this model is shown in Table 5.2 below.

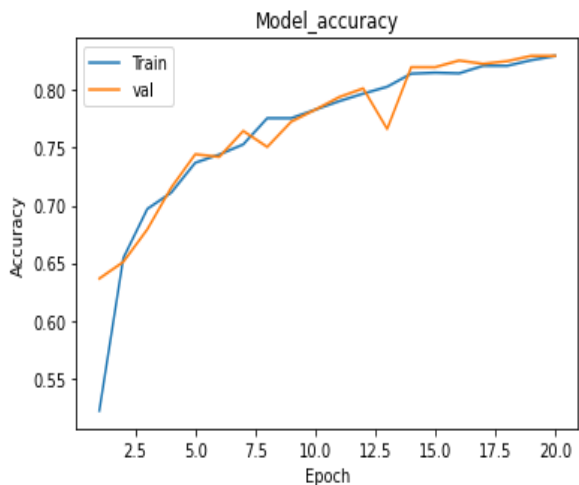


Figure 5.12: Model accuracy Curve-KNN Imputation

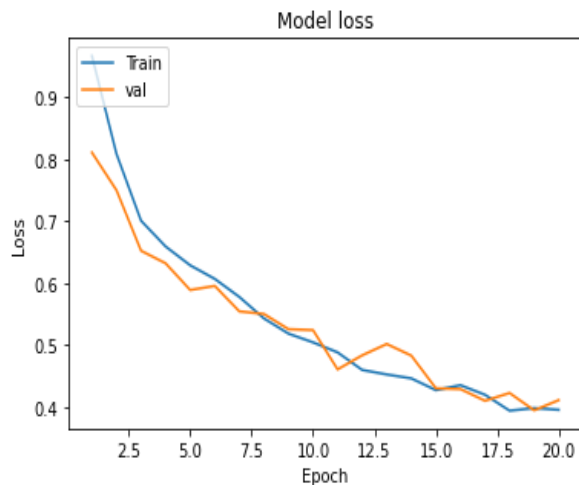


Figure 5.13: Model Loss Curve-KNN Imputation

False Positives Rate	Recall	Precision	F1	Class
0.068	0.780	0.856	0.816	Anti- Extremist
0.115	0.807	0.775	0.791	Neutral
0.100	0.849	0.808	0.828	Pro-extremist

Table 5.2: KNN Imputation Classification Result using RNN

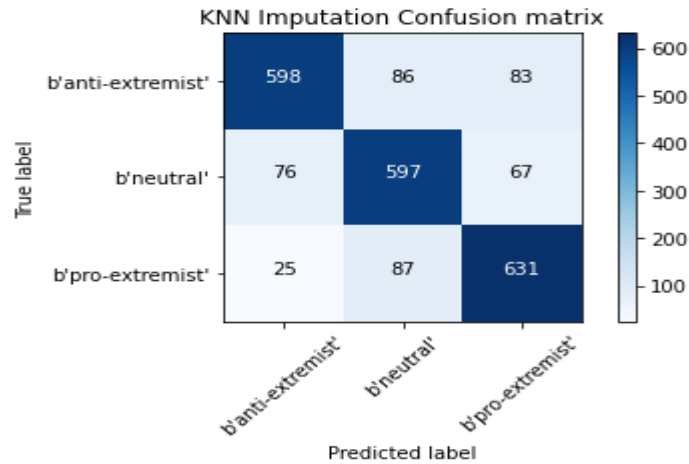


Figure 5.14: KNN Imputation Confusion Matrix

5.11.3 MICE Imputation

The performance of the model indicated that both the validation and training set performed well at epoch 20. At the aforementioned epoch, we could observe both validation and training loss decreases to 0.84 and 0.38 respectively. This shows that the model makes small errors and does not overfit. The regularisation process was visualised with data plots on graphs, model accuracy is detailed in Figure 5.15 while the model loss in Figure 5.16. The model was able to classify 83% of the extremist Webpages across the three categories. The result of applying this model is presented in Table 5.3 below. From the confusion matrix in Figure 5.17, the classifier correctly identified pro-extremist Webpages with the highest rate of 85%.

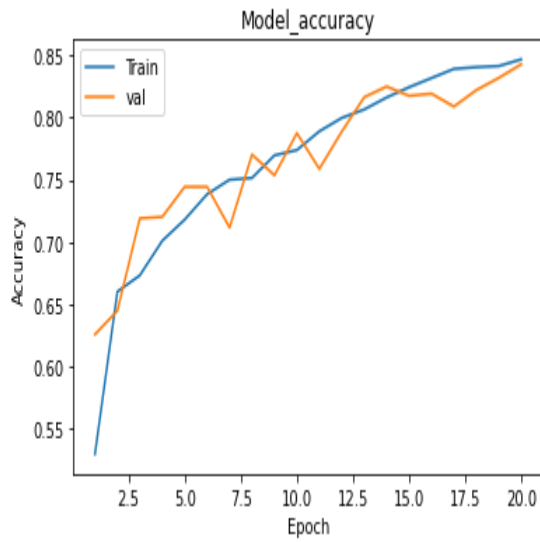


Figure 5.15: Model accuracy Curve-MICE Imputation

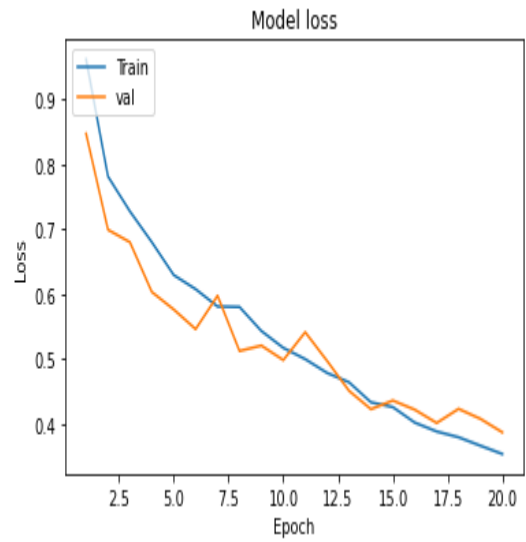


Figure 5.16: Model loss Curve-MICE Imputation

False Positives Rate	Recall	Precision	F1	Class
0.049	0.803	0.894	0.846	Anti-extremist
0.113	0.851	0.787	0.818	Neutral
0.087	0.847	0.828	0.837	Pro-extremist

Table 5.3: MICE Imputation Classification Result using RNN

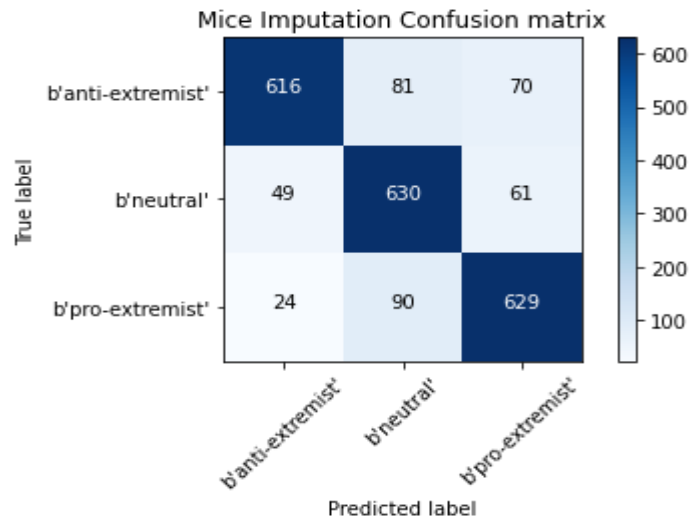


Figure 5.17: MICE Imputation Confusion Matrix

5.11.4 Posit Classification Frameworks

When Tensorflow was implemented on Posit features (27 features), the early stopping function was triggered based on the callback set-up to oversee the performance measure of the training at a point where the accuracy of the validation set degenerated to a level where there was no need to continue the training or when the model loss began to increase on the validation set. At this point, training was terminated and the ModelCheckpoint callback finally overwrote previously saved best models and indicated the optimum at epoch 20. The regularisation process was illustrated graphically using data plots and the performance, as visualised in Figures 5.18-5.19. The model loss error is shown in Figure 5.19, it clearly shows how the error decreases in validation and training set to 0.9 and 0.30 respectively at epoch 20. This showed that both the validation and training set performed well on the model and well fit. The RNN model performance showed that 86% of the whole set of Webpages was classified into the three categories, while the pro-extremist category gave the highest level of correctly classified cases and the lowest false positive rate at, 89% and 2.8% respectively. The classification result is presented in Table 5.4 and the confusion matrix in Figure 5.20 below.

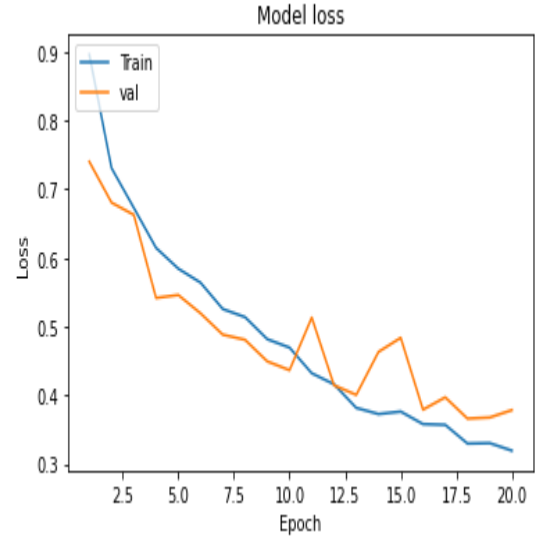
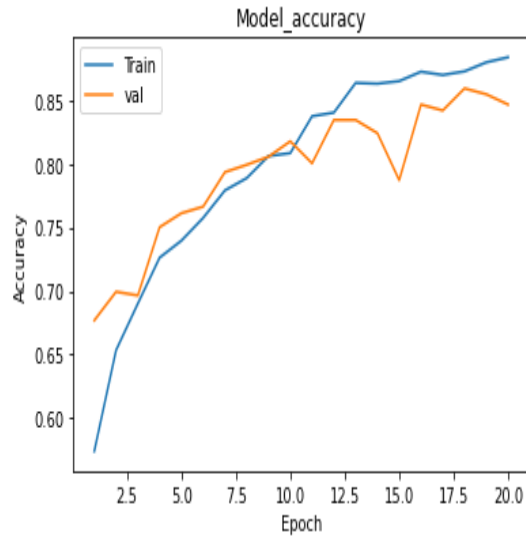


Figure 5.18: Model Accuracy Curve-Posit Imputation

Figure 5.19: Model Loss Curve-Posit Imputation

False Positives Rate	Recall	Precision	F1	Class
0.127	0.907	0.787	0.843	Anti-Extremist
0.046	0.801	0.895	0.845	Neutral
0.028	0.891	0.94	0.915	Pro-Extremist

Table 5.4: Posit Classification Result using RNN

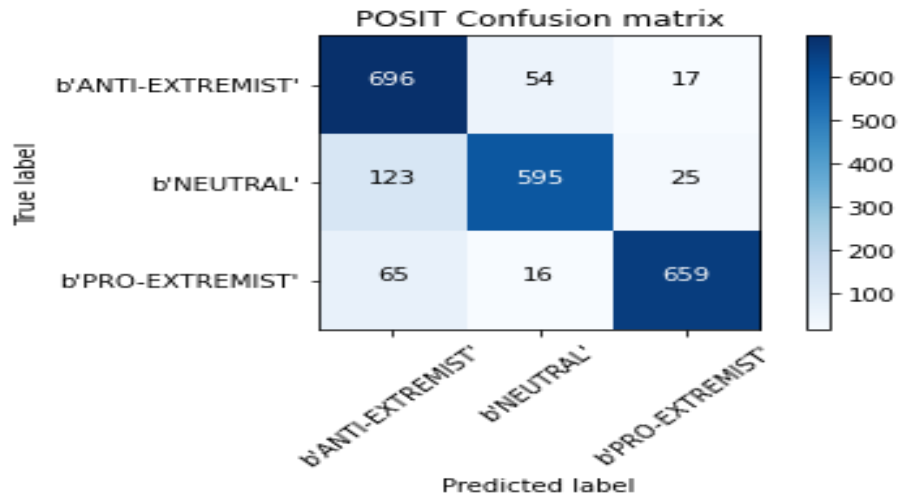


Figure 5.20: Posit Imputation Confusion Matrix

5.11.5 Extended Posit Classification Framework

In addition, the same RNN implementation was performed on a set of extended-Posit features (71 features). The optimum performance of the model was shown at epoch 20 when the classifier gave 91% overall correctly classified instances. The highest correctly classified instance of Webpages was recorded in the pro-extremist category at 96 %, and this category also had the least false positive rate, at 2.3%. The regularisation processes are illustrated in Figures 5.21-5.22. Figure 5.22 shows how the model loss decreases significantly in both validation and training. At epoch 20, loss was decreased to 0.18 in training while 0.25 in validation. This indicates a well fit model. The classification results and the confusion matrix are presented in Table 5.5 and Figure 5.23 respectively.

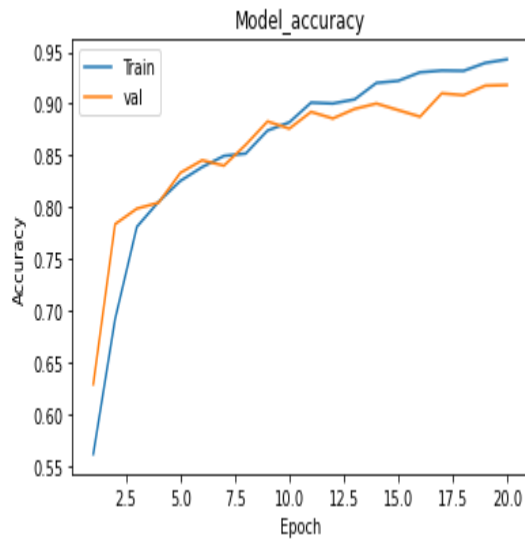


Figure 5.21: Model Accuracy Curve-Extended Posit

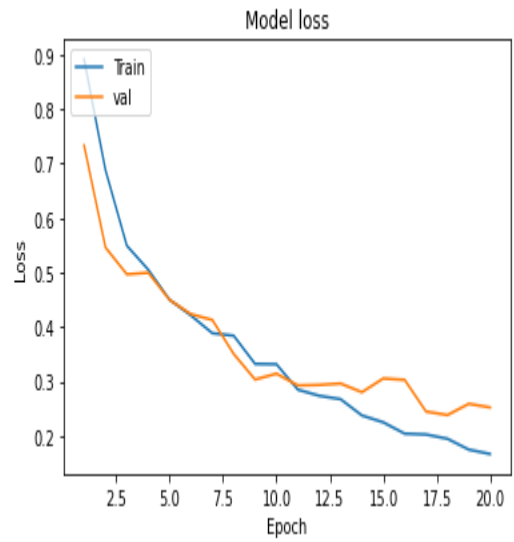


Figure 5.22: Model Loss Curve-Extended Posit

False Positives Rate	Recall	Precision	F1	Class
0.089	0.952	0.841	0.893	Anti-Extremist
0.019	0.834	0.958	0.892	Neutral
0.023	0.955	0.954	0.955	Pro-Extremist

Table 5.5: Extended Posit Imputation Classification Result using RNN

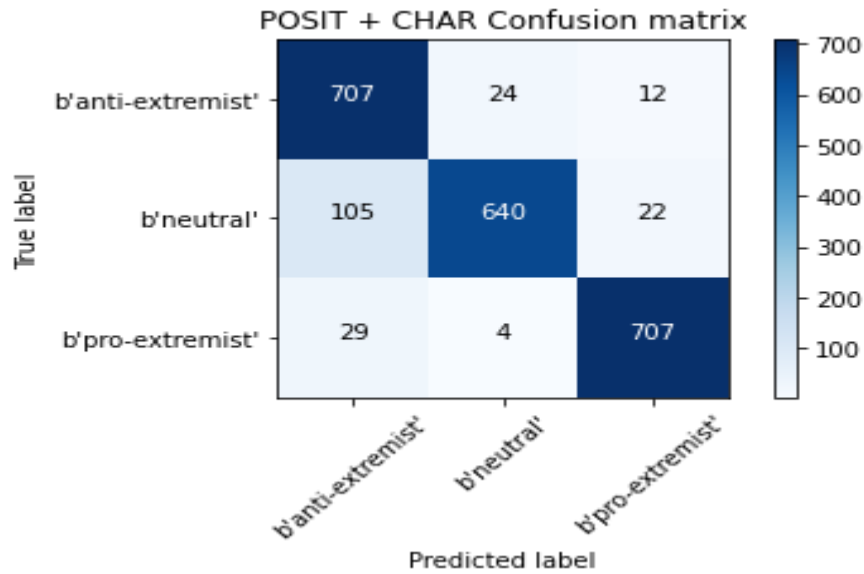


Figure 5.23: Extended Posit Imputation Confusion Matrix

5.11.6 Composite Classification Framework

The regularisation process was illustrated graphically using data plots and the performance, which are visualised in Figures 5.24 and 5.25. Both the validation and training set performed well on the model. Again, at epoch 20, the model loss error in both validation and training decreased significantly to 0.19 and 0.16 respectively (this indicates a low level of error rate), which is described in model the loss curve in Figure 5.25. The RNN model performance showed that 92% of the whole set of Webpages was classified into the three categories, while the pro-extremist category gave the highest level of correctly classified cases and the lowest false positive rate at, 98% and 3.2% respectively. The classification result is presented in Table 5.6 and the confusion matrix in Figure 5.26.

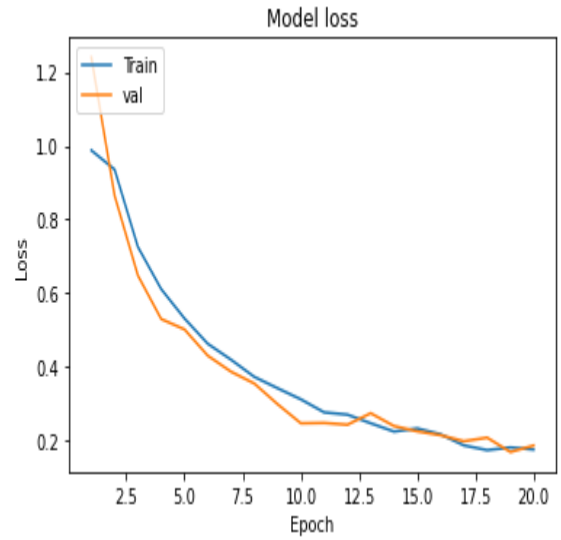
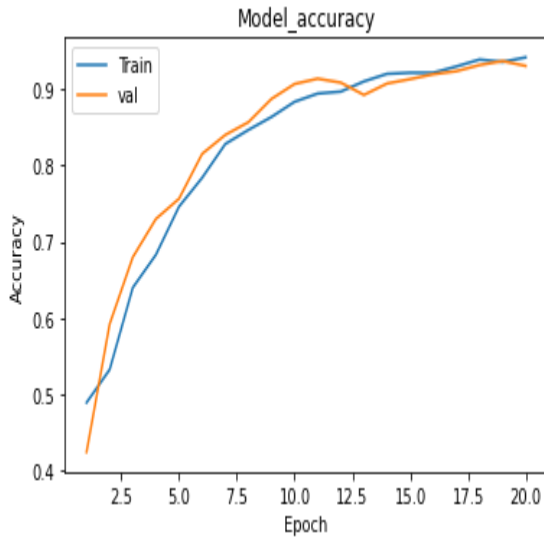


Figure 5.24: Model Accuracy Curve-Composite

Figure 5.25: Model Loss Curve-Composite

False Positives Rate	Recall	Precision	F1	
0.056	0.919	0.89	0.905	Anti-extremist
0.031	0.862	0.935	0.897	Neutral
0.032	0.984	0.938	0.96	Pro-extremist

Table 5.6: Composite-Based Classification (Posit-Mice) Result using RNN

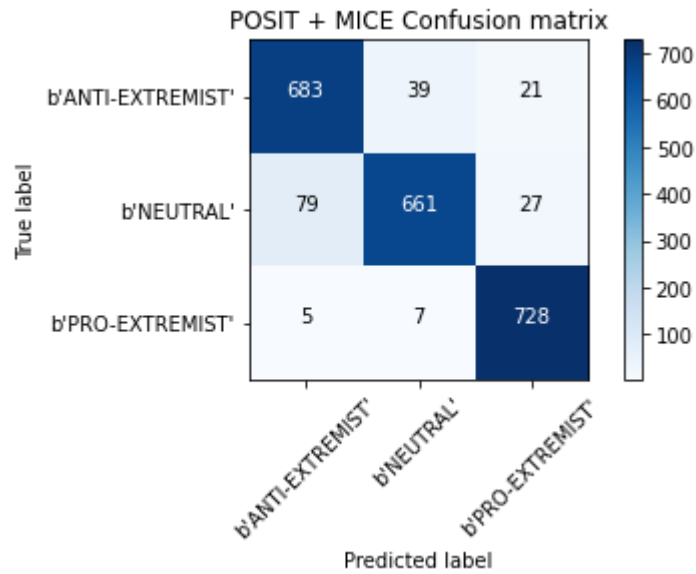


Figure 5.26: Composite-Based (Posit MICE) Confusion Matrix

5.12 MLP CLASSIFICATION RESULT

This section presents the result of each framework using MLP.

5.12.1 Sentiment-Based Framework: (MF, KNN and MICE Imputation)

When the MLP model was applied to the sentiment features (MF, KNN and MICE Imputation sets), the model recorded an optimum performance at epoch 20 each. At epoch 20, the training in each feature set produced the best generalisation performance on both test and training sets. This process is achieved through the aid of performance monitoring callbacks, early stopping and ModelCheckpoint. The regularisation process was visualised with a data plot on the graph in each imputation set, for example MF was depicted in Figures 5.27 and 5.28. The model was able to classify 82% of the extremist Webpages across the three categories in MF. The classification result is presented in Table 5.7 and the confusion matrix in Figure 5.33.

The overall classification result obtained in KNN is 82.3%. The model accuracy and model loss are shown in Figure 5.29 and 5.30 respectively. The classification result is presented in Table 5.8. Mice Imputation produced 85.7% of the overall classification of extremist Webpages. The classification results are presented in Table 5.9 and confusion matrix in Figure 5.35. Both the model accuracy and model loss are shown in Figure 5.31 and 5.32 respectively. The loss errors in each imputation set (sentiment-rule based feature) decrease significantly, this indicates a low level of error rate in each imputation set and hence each imputation doesn't overfit. The classifier correctly identified pro-extremist Webpages with the highest rate in each imputation set. For example, the pro-extremist category in MF produced 91.9% of identified cases. In Mice imputation, the pro-extremist cases were correctly identified at 95.4%. This is also applicable in KNN imputation where the pro-extremist cases were correctly identified at 84.5 %.

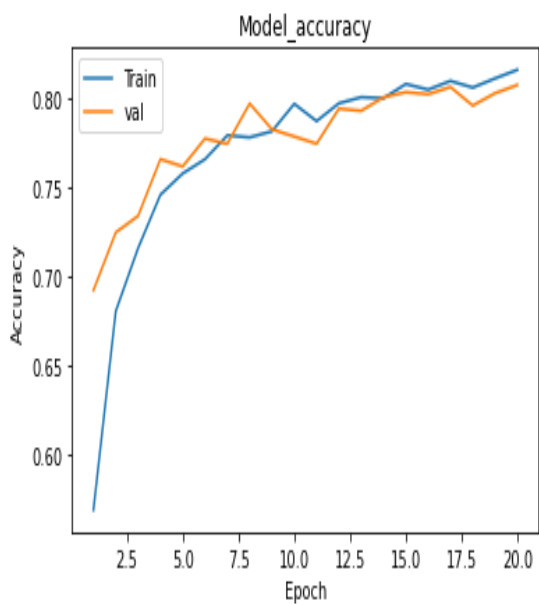


Figure 5.27: Model Accuracy Curve-MF Imputation

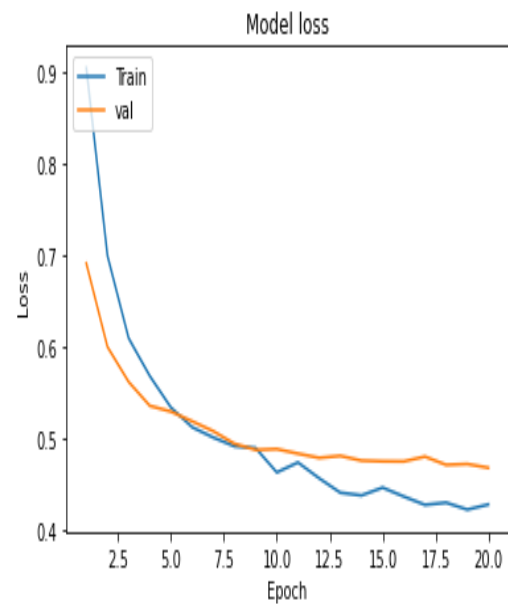


Figure 5.28: Model Loss Curve-MF Imputation

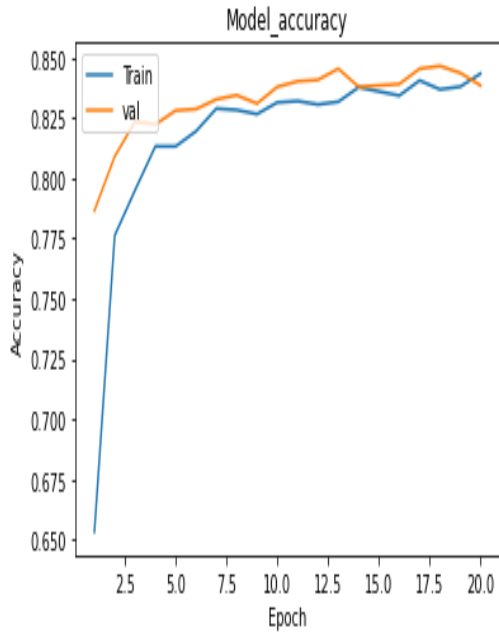


Figure 5.29: Model Accuracy Curve-KNN

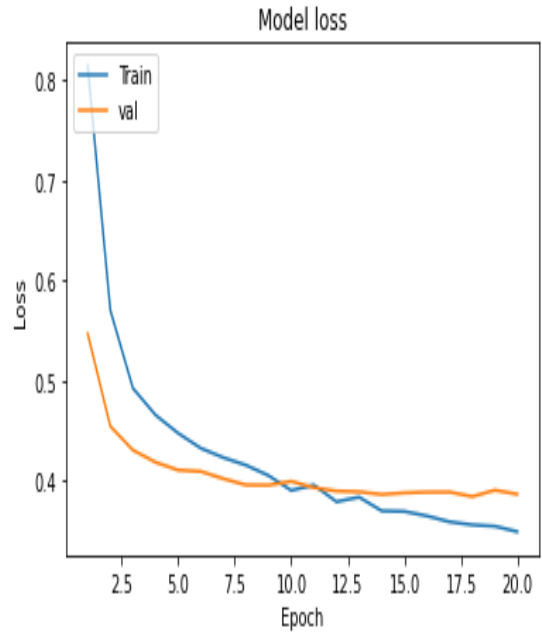


Figure 5.30: Model Loss Curve-KNN

Imputation

Imputation

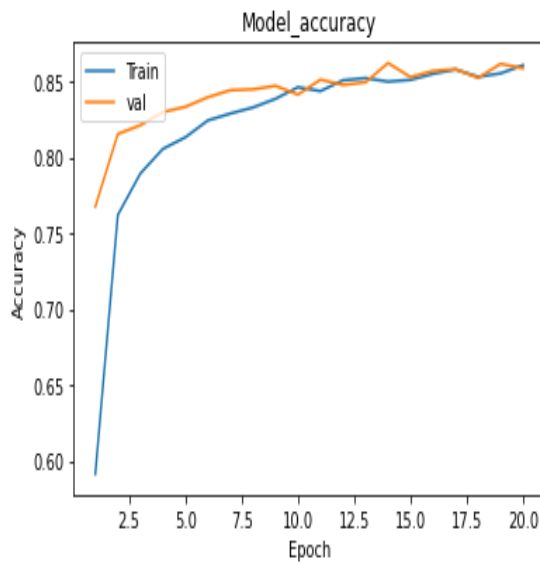


Figure 5.31: Model Accuracy Curve-Mice

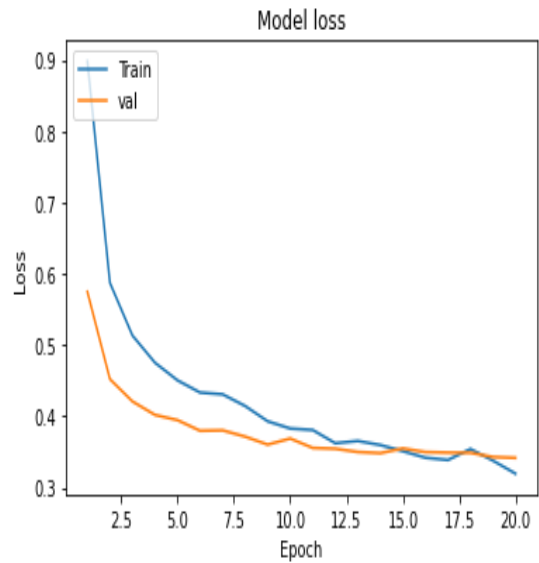


Figure 5.32: Model Loss Curve-MICE

Imputation

Imputation

False Positives Rate	Recall	Precision	F1	Class
0.079	0.731	0.816	0.771	Anti-extremist
0.125	0.803	0.76	0.781	Neutral
0.066	0.919	0.881	0.9	Pro-extremist

Table 5.7: MF Imputation Classification Result using MLP

False Positives Rate	Recall	Precision	F1	Class
0.093	0.785	0.813	0.799	Anti-extremist
0.069	0.84	0.853	0.846	Neutral
0.105	0.845	0.805	0.824	Pro-extremist

Table 5.8: KNN Imputation Classification Result using MLP

False Positives Rate	Recall	Precision	F1	Class
0.032	0.776	0.924	0.843	Anti-extremist
0.058	0.843	0.882	0.862	Neutral
0.124	0.954	0.789	0.864	Pro-extremist

Table 5.9: Mice Imputation Classification Result using MLP

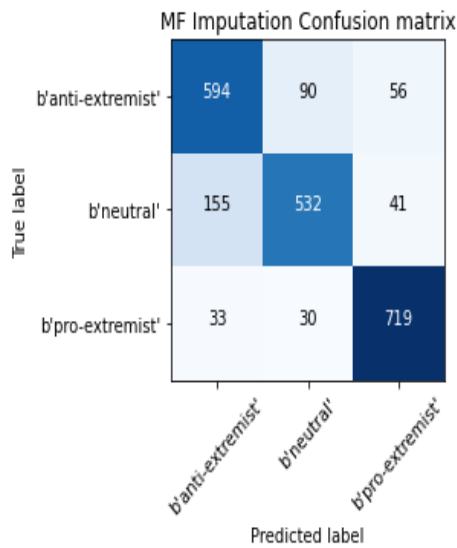


Figure 5.33: MF Imputation Confusion Matrix

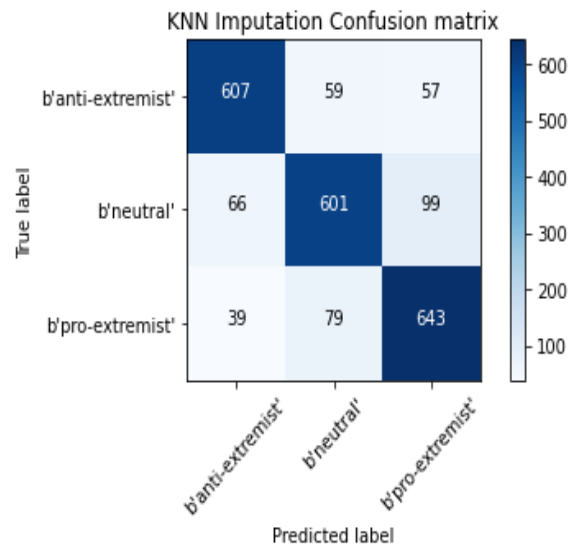


Figure 5.34: KNN Imputation Confusion Matrix

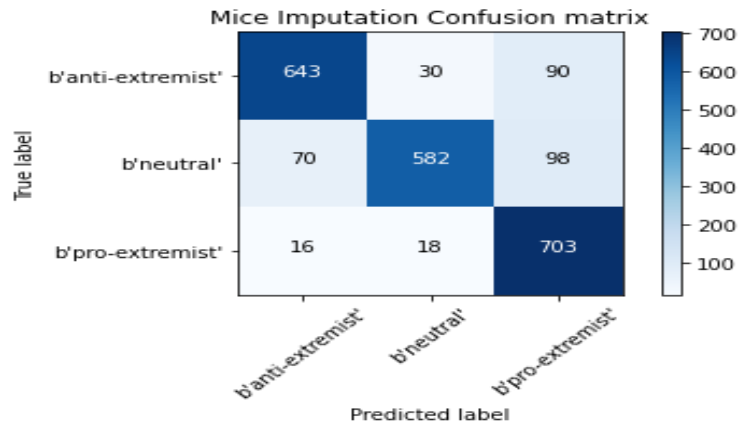


Table 5.35: Mice Imputation Confusion Matrix

5.12.2 Composite-Based Classification Framework

In the Composite (53 features), the MLP model classifies the category for each webpage and the overall classification of extremist Webpages was recorded at 95%. From the confusion matrix in Figure 5.38, the pro-extremist category had the highest level of correctly classified cases at 98.2%. The results are presented in Table 5.10. The performance of the model indicated that both the validation and training set performed well at epoch 20. Figures 5.36 and 5.37 detailed both the validation and training curves. Figure 5.37 shows how the model loss decreases significantly in validation and training at epoch 20, the loss was decreased to 0.13 in training while 0.18 in validation. This indicates a well-fit model.

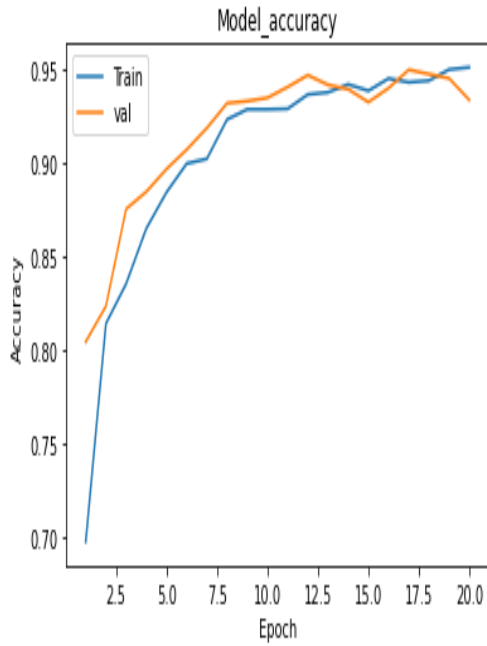


Figure 5.36: Model Accuracy Curve-Composite

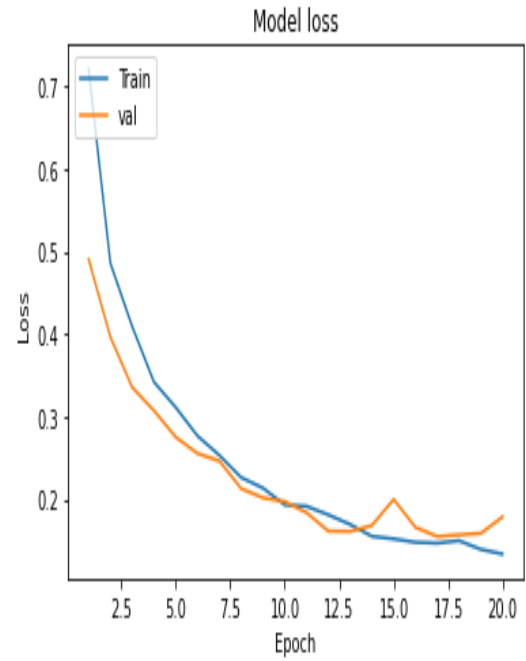


Figure 5.37: Model Loss Curve-Composite

False Positives Rate	Recall	Precision	F1	Class
0.042	0.948	0.915	0.931	Anti-extremist
0.02	0.921	0.96	0.941	Neutral
0.013	0.982	0.975	0.978	Pro-extremist

Table 5.10: The Composite Classification Result using MLP

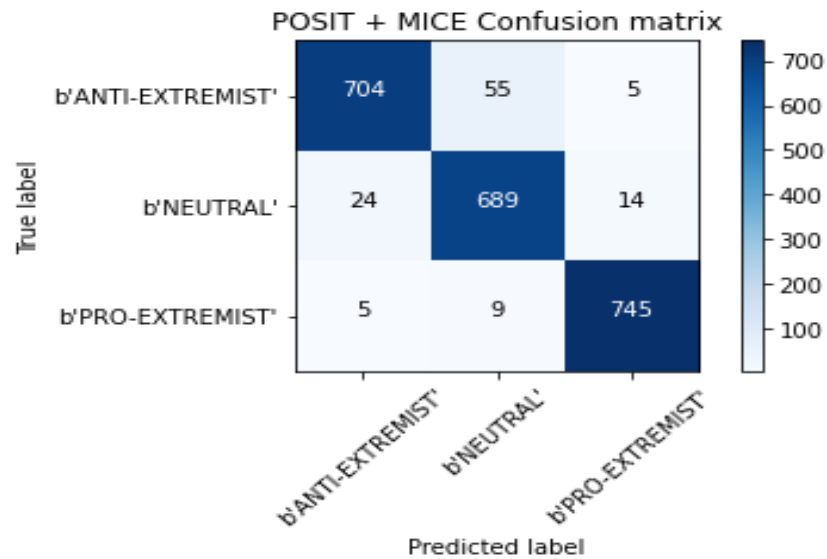


Figure 5.38: Composite Confusion Matrix

5.12.3 Posit-Based Classification Results

When Tensorflow was implemented on Posit features (27 features), the early stopping function was triggered based on the callback set up to oversee the performance measure of the training set. Eventually, the best model indicated its optimum performance at epoch 20. The regularisation process was illustrated graphically using data plots and the performance, as visualised in Figures 5.39 and 5.40, showed that both the validation and training set performed well on the model. The MLP gave 88% of the whole set of Webpages being classified into the three categories, while the pro-extremist category produced the highest level of correctly classified cases at, 90.9%. The classification result is presented in Table 5.11 and the confusion matrix in Figure 5.41. The little variances between the validation and training losses score indicate that the model doesn't overfit. The model error decreases from 0.86 to 0.32 in training while 0.62 to 0.29 in the validation set. The model loss is plotted in a graph displayed in Figure 5.40.

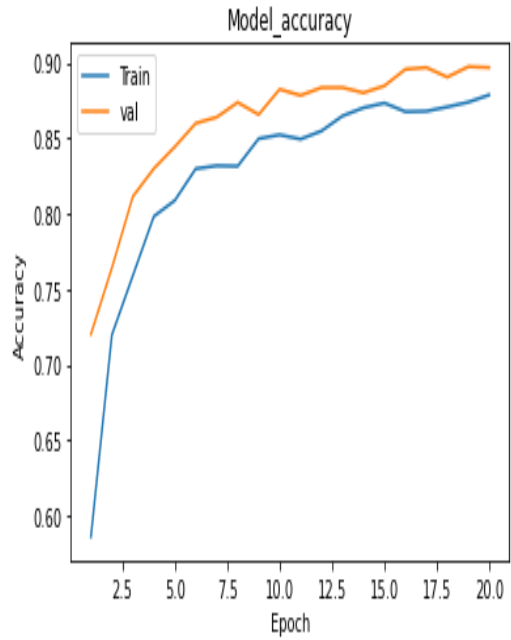


Figure 5.39: Model Accuracy Curve-Posit

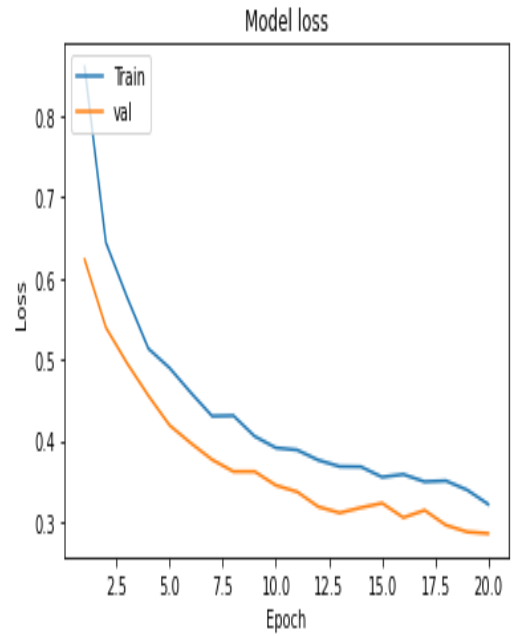


Figure 5.40: Model Loss Curve-Posit

False Positives Rate	Recall	Precision	F1	Class
0.056	0.849	0.881	0.865	Anti-extremist
0.072	0.885	0.86	0.872	Neutral
0.05	0.909	0.904	0.906	Pro-extremist

Table 5.11: Posit Classification Result

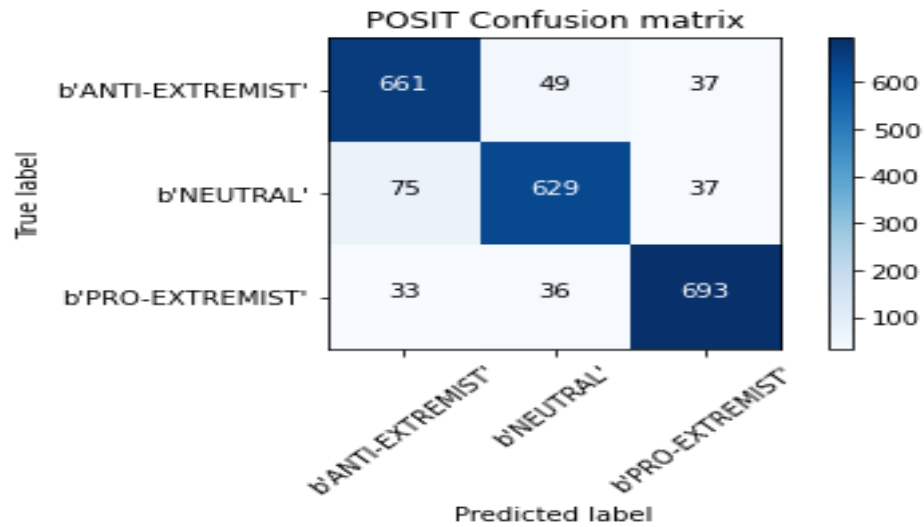


Figure 5.41: Posit Confusion Matrix

5.12.4 Extended Posit Classification Results

The implementation of MLP on Extended Posit (71 features) successfully achieved many convergences and gave optimum result at the 20th epoch. In addition, low loss error was recorded on both the validation and training set, this indicates that the model is fit. Figures 5.42 and 5.43 illustrate the regularisation process envisioned with data plots on the graphs. The model classifies the category for each webpage and the overall classification of extremist Webpages was recorded at 93.9%. The pro-extremist category had the most identified cases and the lowest false positive rate, at 95.4% and 2.1% respectively. Table 5.12 and Figure 5.44 detailed the results.

False Positives Rate	Recall	Precision	F1	Class
0.033	0.929	0.934	0.932	Anti-extremist
0.037	0.934	0.925	0.929	Neutral
0.021	0.954	0.958	0.956	Pro-extremist

Table 5.12: Extended-Posit Classification Result

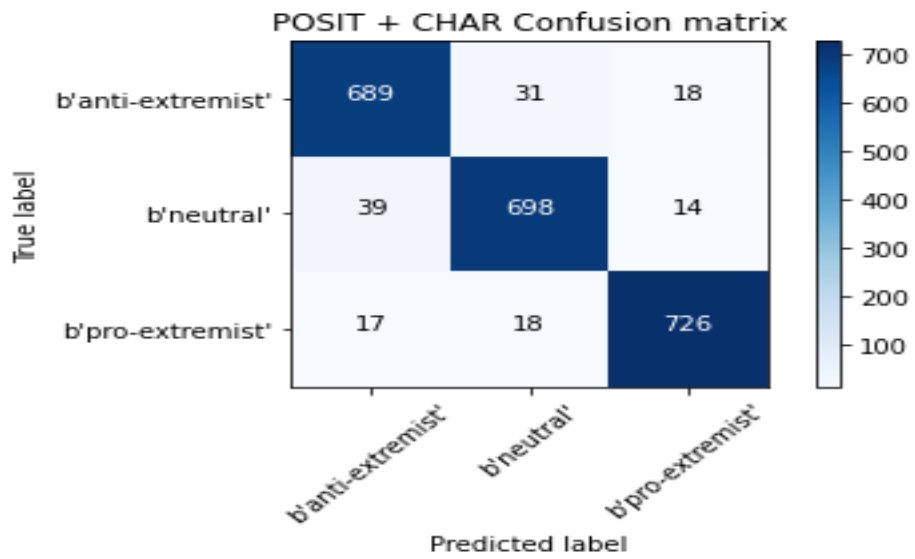


Figure 5.42: Extended Posit Confusion Matrix

Chapter 6: Results Analysis and Evaluations

As a reminder, the classification methods in question, are Sentiment (KNN, Missforest and MICE Imputation), Posit (features on the basis of word-level information), the Extended Posit (features on the basis of both word and character-level information) and the proposed framework, Composite-based classification method. The classification algorithms considered in this thesis are KNN, Random Forest, J48, RNN and MLP. The results described in this chapter reveal the classifier and classification framework that works best in classifying extremist content. The results also revealed the advantages of feature selection against hyperparameters turning on classification accuracy.

6.1 Sentiment-based Classification Framework

Research Question 1: Can the imputation method efficiently compensate for missing values faced by feature set obtained via sentiment analysis (a procedure that utilises top-k noun keywords to obtain sentiment values from text corpus) before being fed into machine learning for the classification task?

Missingness of data is inevitable in the sentiment analysis method developed in this thesis because utilizing top-k noun keywords to obtain sentiment values from text corpus will create a situation where some Webpages have few or none of the selected noun keywords and hence lead to missing data. However, the best approach to replace the missing data for the particular feature set led to many imputation techniques explored. Mice imputation technique was able to compensate efficiently for the missing values obtained in the sentiment analysis. Among the machine learning algorithms considered, the result showed Random Forest gave the best performance. Random Forest gave Missforest a classification accuracy of 86%, while 85% for KNN and MICE at 88% respectively. Table 6.1 shows the frameworks result comparison. However, the result from the classification algorithms employed showed that multivariate imputation approach (MICE) outperformed other imputation approaches such as, KNN and

Missforest. Hence, MICE imputation approach came out to be the best method to handle missing values in keyword-based sentiment feature set (a given data set with the conditions that the data is missing at random (MAR)) before being feed into machine learning algorithm. Multiple imputation by chain equations (MICE) approach has demonstrated flexibility in handling missing data nature of data better than other approaches.

6.2. Composite-Based Classification Framework

Research Question 2: Can the Composite approach (the combination of sentiment and syntactic features in textual content as a basis for text features) be effective to create a well working machine learning model?

The essence of this research question is to test the effectiveness of the classification methods with their various feature sets when classifiers are applied. Out of the deployed models on each feature set, composite feature set (a proposed framework that deployed a hybrid of both sentiment and syntactic features of texts) outperformed other classification frameworks with Random Forest as the best performing model, at 95.8%. In addition, the proposed framework showed the highest proportion of correctly identified pages across all three classes, compared to other frameworks. This is an indication that the composite-based classification method is more effective in discerning content that expresses extremism than other frameworks. Table 6.1 shows the result comparison of the classification frameworks. A likely reason for the greater effectiveness of composite features is that they afford a wider coverage of more useful features, both sentiment and syntactic, in a Web text. In addition, pro-extremist category had the most identified cases and highest precision in composite framework when compared with other categories in other classification frameworks.

	RF	J48	KNN	RNN	MLP
MF	86	83	82.1	79.5	82
KNN	85	84.3	81	81	82.3
MICE	88	86.5	83	83	85.7
Posit	93	89.4	89	86	88
Ext-Posit	95	89.6	90.3	91	93
COMPOSITE	95.8	91.5	92	92	95

Table 6.1: Comparison of the Classification Methods

6.3 Feature Selection vs Model Optimisation

Research Question 3: What is the cost of model optimisation (hyperparameter turning) over feature selection when creating a machine learning model?

Hyperparameter turning is a stopping criterion explored to achieve optimum values and minimize overfitting. In an attempt to minimize overfitting and enhance the optimal output of the model, hyperparameter tuning was employed using GridsearchCV. By fine-tuning the models, we obtained the best parameters across the different feature sets used in each model which also revealed how the different parameter values affect our final score (performance metrics). While feature selection operates by selecting a subset of relevant features for use in model construction to improve accuracy and run-time most especially in model construction where there are numerous features and comparatively few samples (or data points). Enhancing the prediction performance of the predictors to produce efficient and effective predictors is the main objective of feature selection.

However, when both wrapper method (feature selection) and the GridsearchCV method (model optimisation) were applied to each feature set, the results obtained from Random Forest, the wrapper method outperformed the GridsearchCV method in each framework. The same trends apply to J48. The results obtained from the J48 model indicated that the

Wrapper method outperformed the GridsearchCV method in all the classification methods except in composite where the GridsearchCV method gave better accuracy at, 91.5% against the wrapper method which achieved 91.3%. Table 6.2 and 6.3 below detailed the comparison of the results between the feature selection and hyperparameter turning.

s/n	Frameworks	% of feature subsets	Runtime For Wrapper Method (sec)	Accuracy of Wrapper Method (%)	Grid searchCV Accuracy (%)
1	Sentiment(Mice)	90	3.4	86.9	86.4
2	Posit	60	3.54	89.4	89.4
3	Composite	60	8.04	91.3	91.5
4	Ext Posit	45	9.32	90	89.6

Table 6.2: Result Comparison between Wrapper and GridsearchCV Methods using J48

s/n	Frameworks	% of feature subsets	Runtime for Wrapper Method (sec)	Accuracy of Wrapper Method (%)	Grid searchCV Accuracy (%)
1	Sentiment(Mice)	100	3.2	88.2	88%
2	Posit	100	9.6	93.9	93%
3	Composite	100	9.08	95.9	95.8%
4	Ext Posit	75	165	95.9	95%

Table 6.3: Result Comparison between Wrapper and GridsearchCV Methods using Random Forest

The results shown from both feature selection and model optimization (hyperparameter turning) when creating a machine learning model showed that hyperparameter tuning takes time and is computationally expensive. This isn't to suggest that hyperparameter tuning isn't vital; rather, when it comes to increasing a model's performance, it's a top priority. It takes a long time to cycle different hyperparameter combinations in order to obtain a tiny improvement. Even worse, if you have a large amount of data and a complicated model, each iteration consumes a lot of resources. As a result, performing feature selection to represent the problem well enough for models to learn and predict accurately is a more intelligent approach for achieving great results with quantum leaps of improvement in a shorter time frame. If time allows, we can investigate tweaking hyperparameters after we have great features, therefore feature selection should come first and hyperparameter tuning should follow second. Hence, great features are still important in determining a model's success and cost effectiveness for machine learning tasks. Taking cost into account, feature selection improves classification accuracy and saves time better than hyperparameter turning.

6.4 Machine Learning and Neural Network Algorithms

Research Question 4: Considering the selected Machine Learning and Neural Network algorithms (such as RNN, MLP, KNN, J48 and Random Forest) on a pre-processed feature, which model produce the best classification accuracy on extremist Web textual data?

6.4.1. Overall Classification Results

Based upon the outcomes of the experiments, Random Forest gave the best classification performance on the Posit-based classification framework when compared with other classifiers. The model was able to classify 93% of the Webpages into their various categories. Again, Radom Forest earned the extension of Posit (word level data with character-level information) with topmost accuracy when compared with other classifiers explored, it enhanced Extended-Posit performance to a creditable 95% correctly classified

instances of the overall Webpages. Table 6.1 shows the frameworks' results comparison. Clearly, the extension of Posit textual analysis to include both word and character-level features, outperformed word-level feature alone in the classification. Notably, the Posit approach to language analysis relied entirely upon the frequency of syntactic features.

Taking the overall parameters into account, when deployed with RNN and MLP classifiers, the composite-based classification framework outperformed the other frameworks. The results showed that the composite features gave the best classification accuracy of 95% with MLP as a better model when compared with RNN. In addition, among all the deployed models, composite feature set (a proposed framework that deployed a hybrid of both sentiment and syntactic features of texts), Random Forest gave the best performing model, at 95.8%. Hence, composite features are preferable to solely syntactic or sentiment features and can offer improved classification accuracy when used with machine learning algorithms. Conclusively, there was a noticeably better performance among the six frameworks when the Random Forest classifier was applied compared to the results obtained in other algorithm, at 95.8%. Table 6.1 presents the detailed results.

6.5 Validation of Nigerian Extremism Webpages

Research Question 5: Can a model based on the dataset used for these experiments be validated on another dataset of a similar domain but different source?

The objective of this task is to test the efficiency of a classifier on two different set of data, with similar topics and predefined classification. Then, if the result provides a good degree of match in the classification result, the latter could be taken as a validation set for the other dataset. With this in mind, the two datasets considered to test this research question are (i) a set of Nigerian extremist Webpages and (ii) a set of extremist web pages previously obtained by the TENE Web-crawler [5]. Both datasets had the same predefined categories of Webpages such as pro-extremist, anti-extremist and neutral.

After applying the classifiers, Random Forest outperformed J48 to produce 53% of correctly classified instances of overall Webpages into their respective categories in the Extended-Posit based classification framework. Again, Extended-Posit outperformed Posit in both Random Forest and J48. In Extended-Posit, J48 produced 49% overall accuracy better than 48% obtained in Posit. In addition, all the five classifiers were applied in the experiment to ascertain the best classification model but similar trends of the results stated above were observed but we reported the two best performing models to avoid tautology following the explicit presentation of the aforementioned analysis and results. Sentiment and composite feature sets could not be used as validation because of the difference and unrelated feature each possessed.

Considering, the training curves from sections 4.10-4.10.2, we could see that the training curves were healthy, well-fitted models. This indicates that a model based on the dataset used for these experiments can be validated on another dataset of a similar domain and different source provided that both datasets possess related features. However, the low overall accuracy recorded can be traced to the small dataset used for the validation process.

6.6. Results comparison with the literature

Our proposed approach in sentiment analysis outperformed the method explored in [5]. The method improved the sentiment-based classification method better than what was obtainable in the literature which also explore the same sentiment analysis approach and dataset [5]. J48 gave a classification accuracy of 86.5% unlike the 80.6% overall classification performance obtained in [5]. Table 6.5 below detailed the result comparison.

Moreover, the textual content classification method was improved further with a composite classification framework that deployed a hybrid of both sentiment and syntactic features of texts, the J48 algorithm gave 91.5% correctly classified instances of

the Webpages using J48. The pro-extremist category had the highest degree of correctly identified pages, at 93.2%. The hybrid method showed the highest proportion of correctly identified pages across all three classes, compared to the ICCRC sentiment analysis method in the literature [5]. Taking into account the overall evaluation parameters such as precision, recall and f-score in the analysis, the results show that the composite method is more effective in discerning content that expresses extremism than the ICCRC sentiment analysis method in the literature [5]. Table 6.4 detailed the comparison of our results with the literature.

Classification Frameworks	Overall Accuracy	Pro	Anti	Neu
Mice Imputation (Sentiment Analysis)	86.5	86.5	86.5	86.4
ICCRC Method [5] (Sentiment Analysis)	80.51	92.7	88	68
Composite Method	91.5	93.2	90.9	90.6

Table 6.4: Results comparison with the literature

6.7 Human-Verification of Manually Labelled Data

We have three manual categories, anti, pro-extremist and neutral. According to how the pages were manually classified, all pages from extremist Websites were all gathered into the pro-extremist class. However, not all pages in this category might have 100% extremist content. Some Webpages crawled from this domain may have neutral contents such contact us, about us for example. This is an error that automated classification was able to correct by putting such content in the category where it belongs. Although this situation was minimal as human verification was done randomly on the manual labelled

data. Out of one hundred pages manually checked, only five pages were found to be mismatched.

However, conducting our automated classification on such will not deter the efficiency of our models. The mismatched cases won't affect classification accuracy significantly because it makes up only a minute percentage of the extracted webpages wherein the correctly classified cases are much more, if we factor this into a percentage, about 90% of the webpages were correctly classified, on which our model is trained on. In addition, we carried out the validation process for each hold-out dataset giving us the validation performance rate, metric and accuracies. We also paid further attention to the precision, recall and f-score.

Chapter Seven: Conclusions and Future Work

This chapter details the conclusion of the studies carried out in the research reported in this thesis.

7.1 Conclusions

The rapid increase of extremist documents online has created the need for efficient automated systems for the classification and identification of such Webpages. This ability will assist in the triage and further investigation on the particular Web pages that are likely to relate to terrorism or extremism. Additionally, this will aid in countering extremist activities such as recruitment and radicalisation on the Internet. In this thesis, six different classification frameworks were developed, specifically, Sentiment-based (Mice, KNN and MissForest imputation), Posit-textual, Extended-Posit and Composite-based classification frameworks. The machine learning algorithms explored are MLP, RNN, Random Forest, J48 and KNN.

CNN is another interesting neural network algorithm but it was not explored because we do not necessarily need to perform convolutions on text-based datasets as they have been known to be inefficient in such applications but better applications are in MLP, RNN or LSTM wherein a better representation of words and vectors are established as well as the context vectors. As an approach to overcome overfitting or underfitting, we performed overfitting analysis using the learning curve. This is done using hyperparameters turning in each model over a range of values. A plot of the test and train accuracy at each hyperparameter value was drawn. The curve was observed. Then, the first peak of test set performance was recorded as the best generalisation performance. Taking all the classification models and evaluation metrics into account, the standardised data produced better results than normalised data and we presented standardization results in the thesis only. The normalised data's results were not presented to avoid unnecessary details. Lastly, the effectiveness of the models was compared and the conclusions were drawn.

7.1.2 The Contributions from the Thesis

The study evaluated the application of different classification frameworks; this is to establish their effectiveness as a basis of text features used in building a classification model. The frameworks are:

i. Sentiment analysis-based method. The thesis evaluated different types of imputation methods applied to compensate for missing values faced by the sentiment analysis method that relies on the use of top-k noun keywords to obtain sentiment around each Web page,

ii. Posit (on the basis of word-level information),

iii. Extension of Posit (with an additional 44 character features) on textual data and

iv. a novel framework, a composite-based (a computational framework that explores the combination of both sentiment and syntactic features of textual contents as a basis for text features which enhances textual data classification model). The reason behind the hybrid features in the composite approach is to use the substantial feature set that feeds into building a classification model,

The thesis analysed the performance of Neural Network algorithms (such as RNN and MLP) and traditional machine learning algorithms (such as the J48 decision tree, K-Nearest Neighbor and Random Forest) on text corpora, extremist Web text, this is to determine the best model for such text content classification.

The thesis also evaluated the cost of hyperparameter turning over feature selection in creating a machine learning model.

Models based on the dataset used for these experiments were validated on the Nigerian dataset (a dataset of a similar domain but different source) this is to check if it can be taken as validation for the approach since it would seem to work well across differently sourced data sets (for the same classification tasks).

The thesis concluded by giving the summary of the outcome of each research question respectively. The results are given below:

i. The composite features are preferable to solely syntactic or sentiment features and can offer improved classification accuracy when used with machine learning algorithms. Consequently, there was a noticeably better performance among the six frameworks when the Random Forest classifier was applied compared to the results obtained in other algorithms, at 95.8%. The extension of Posit textual analysis to include both word and character-level features outperformed word-level features alone in the classification.

ii. Imputation approach can compensate for the missing values on a dataset when a machine learning algorithm is applied. Among all the imputation methods considered, the MICE imputation approach came out to be the best method to handle missing values faced by sentiment feature obtained via sentiment analysis (a sentiment analysis procedure that utilises top-k noun keywords to obtain sentiment values from text corpus) before being fed into machine learning for the classification task. The multivariate imputation (MICE) approach has demonstrated flexibility in handling varying nature of data such as the continuous or binary data better than other approaches.

iii. Taking cost into account, feature selection improves classification accuracy and saves time better than hyperparameter turning

vi. Considering the selected machine learning and neural network algorithms (such as RNN, MLP, KNN, J48 and Random Forest) on a pre-processed feature, Random Forest offered the best classification accuracy on extremist Web textual data.

v. A model based on the dataset used for these experiments can be validated on another dataset of a similar domain and different source provided that both datasets possess related features.

7.2 Future Work

The BERT algorithm (Bidirectional Encoder Representations from Transformers) is a deep learning algorithm for natural language processing. This is another algorithm that was considered for inclusion in this thesis but BERT is a very sophisticated algorithm that demands time and more hardware resources in terms of Ram and GPU which would be costly. Hence we decided to focus on more comfortable ML algorithms that can handle our data size but yet be very accurate hence eliminating heavy training costs. This algorithm will be considered in future work.

In addition, a future study on textual analysis and classification would be projected toward the exploration and investigation of security threats through cyber-attacks on the Internet. We would be exploring the dynamics of the composite-based model that utilizes a vast range of sentiment and syntactic features contained within a text block. This would further stretch the boundaries of the composite-based model as we would be employing a much larger variety of datasets from various sources including job relaying sites, public business offer sites, social media sites and a lot more. This data would be based on fake job listings, false ROI for business opportunities, and fake SMS from fraudsters posing to represent the customer's bank thereby asking for salient banking details and others. The composite-based model will be utilised for identifying text contents that have fraudulent motives.

Bibliography

[1] A. N. Awan, A. Hoskins, and B. Loughlin, "Radicalisation and Media: Connectivity and Terrorism in the New Media Ecology", New York: Routledge, pp154, 2011. ISBN: 978-0-415-64199.

[2] L. Bowman-Grieve, "Exploring "Stormfront": A Virtual Community of the Radical Right", *Studies in Conflict & Terrorism*, Vol 32, pp 989-1007, 2009. DOI: 10.1080/10576100903259951

[3] M. Midlarsky, "Origins of political extremism. Mass violence in the twentieth century and Beyond", Cambridge: University Press, 2011.

[4] R. Rogers, "Digital Methods. Cambridge", MIT Press, 2013.

[5] J. Mei and R. Frank, "Sentiment crawling: Extremist Content Collection through a Sentiment Analysis Guided Web-Crawler," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social.*

[6] Q. Schiermeier, "Terrorism: Terror prediction hits limits." *Nature*, vol. 517, no. 7535, p. 419, 2015.

[7] L. Eric, "Bin Laden Chose 9/11 Targets, Al Qaeda Leader Says" in the *New York Times*, 2003.

[8] G. Birchall, W. Christmas and P. Harper, "TERROR IN THE CAPITAL," in the *Sun News Paper*, Retrieved from: <https://www.thesun.co.uk/news/3151868/london-westminster-terror-attack-bridge-victims-about/>

[9] J. S. Rivinius, "START Background Report," published in Department of Homeland Security Center of Excellence, University of Maryland. Retrieved from: <http://www.start.umd.edu/news/proportion-terrorist-attacks-religious-and-right-wing-extremists-rise-united-states>, November 2017.

[10] A. Geron, Hands-On Machine Learning With Scikit-Learn and Tensor-Flow: Concepts Tools and Techniques to Build Intelligent Systems, Sebastopol, CA, USA:O'Reilly Media, pp. 543, 2017, [online] Available: <http://shop.oreilly.com/product/0636920052289.do>

[11] J. Wang, P.Liu, M. F.H.She, S. Nahavandi, A. Kouzani, "Bag-of-words representation for biomedical time series classification", Biomedical Signal Processing and Control, Volume 8, Issue 6, pp 634-644, November 2013.

[12] Y. Xia, L. Wang, K. F. Wong, M. Xu, "Lyric-based Song Sentiment Classification with Sentiment Vector Space Model", in Proceedings of ACL-08: HLT, Short Papers (Companion Volume), pages 133–136, 2008.

[13] R. Scrivens and R. Frank, "Sentiment-Base Classification of Radical Texts on the Web," in Proceeding of the European Intelligence and Security Informatics Conference, 2016, pp 104–107.

[14] G. R. S. Weir, E. D Santos, B. Cartwright and R.Frank, "Positing The Problem: Enhancing Classification of Extremist Web Content Through Textual Analysis," in Proceedings of the 4th International Conference on Cybercrime and Computer Forensics (ICCCF), Simon Fraser University, Vancouver, Canada. 2016.

- [15] B. Cartwright, G. R. S. Weir, and R. Frank, Fighting Disinformation Warfare with Artificial Intelligence, Tenth International Conference on Cloud Computing, GRIDs, and Virtualisation, 2019.
- [16] M. Thelwall and K. Buckley, "Topic-based sentiment analysis for the social Web: The role of mood and issue-related words," *Journal of the American Society for Information Science and Technology*, vol. 64, pp. 1608-1617, 2013.
- [17] S. Buuren, & K. G. Groothuis-Oudshoorn, "MICE: Multivariate imputation by chained equations in R", *Journal of Statistical Software*, Vol. 45, pp1-67, 2011.
- [18] M. Nosrati, "Python: An appropriate language for real world programming", *World Applied Programming*, Vol. 1, 2011. pp110-117.
- [19] T. Richards, "An intellectual history of NUD*IST and NVivo", *International Journal of Social Research Methodology*, Vol 5, pp199-214, 2002. DOI: 10.1080/13645570210146267.
- [20] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, C.J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*. Vol.9, 2008, pp1871–1874.
- [21] S. Deepu, P. Raj and S. Rajaraajeswari, "A Framework for Text Analytics using the Bag of Words (BoW) Model for Prediction", in proc. 1st International Conference on Innovations in Computing & Networking (ICICN16) CSE RRCE ISSN: 0975–0282.

- [22] B. Heap, M. Bain, W. Wobcke, A. Krzywicki, and S. Schmeidl, “Word vector enrichment of low frequency words in the bag-of-words model for short text multi-class classification problems”, CoRR, abs/1709.05778, 2017.
- [23] J. Sang, S. Pang, Y. Zha, Y, and F. Yang, “ Design and analysis of a general vector space model for data classification in Internet of Things”, J Wireless Com Network, Vol 263, 2019. <https://doi.org/10.1186/s13638-019-1581-3>
- [24] J. Ababneh, O. Almomani, W. Hadi, N.K.T. El-Omari, A. Al-Ibrahim, “Vector space models to classify Arabic text”, Int. J. Comput. Trends Technol. (IJCTT), Vol 7, pp. 219-223, 2014.
- [25] A. Törnberg and P.Törnberg, “Muslims in social media discourse: Combining topic modeling and critical discourse analysis”, Discourse, Context & Media, Volume 13, pp 132-142, September 2016.
- [26] D. Zha & C. Li, “Multi-label dataless text classification with topic modeling”, Knowledge and Information Systems volume 61, pages137–160, 2019
- [27] D. M. Blei, A.Y. Ng, & M.I. Jordan, “Latent Dirichlet Allocation.”, Journal of Machine Learning Research, Vol. 3, pp993–1022, 2003.
- [28] L. Yang, F. Liu, J. Kizza and R. Ege, “Discovering topics from dark websites”, in Computational Intelligence in Cyber Security, CICS '09. IEEE Symposium. pp 175 – 179, 2009.
- [29] D. Inkpen and A. H. Razavi, “Topic Classification using Latent Dirichlet Allocation at Multiple Levels”, IJCLA VOL. 5, NO. 1, PP. 43–55, 2014.

- [30] T. A. Rana¹ and Y. Cheah, "Aspect extraction in sentiment analysis: comparative analysis and survey," in *Artif Intell Rev*, Springer vol. 46 pp459–483. 2016. DOI 10.1007/s10462-016-9472-z
- [31] J. Andoh, L. Asiedu, A. Lotsi, C. Chapman-Wendy, "Statistical analysis of public sentiment on the Ghanaian government: A machine learning approach", *Hindawi Advances in Human-Computer Interaction*, vol.1, pp1-7, 2021.
- [32] R. Strimaitis, P. Stefanovic, S. Ramanauskaite, & A. Slotkiene, "Financial context news sentiment analysis for the Lithuanian language", *Applied Sciences*, vol.11, pp1-13, 2021.
- [33] M. S. Basarslan, & F. Kayaalp, "Sentiment analysis with machine learning methods on social media", *Advances in Distributed Computing and Artificial Intelligence Journal Regular Issue*, vol. 3, pp5-15, 2020.
- [34] R. Feldman, "Techniques and applications for sentiment analysis" in *Communications of the ACM*, 56(4), pp. 82-88, 2013.
- [35] A. Abbasi and H. Chen, "Applying authorship analysis to extremist group Web forum messages," in *Intelligent Systems*, 20(5), pp. 67-75, 2005.
- [36] A. Bermingham, M. Conway, L. McInerney, N. O'Hare, and A. F. Smeaton, "Combining Social Network Analysis and Sentiment Analysis to Explore the Potential

for Online Radicalisation,” in 2009 International Conference on Advances in Social Network Analysis and Mining (ASONAM). IEEE, pp. 231–236, 2009.

[37] X. Qi, K. Christensen, R. Duval, E. Fuller, A. Spahiu, Q. Wu, and C.-Q. Zhang, “A Hierarchical Algorithm for Clustering Extremist Web Pages,” in 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 458–463, 2010.

[38] J. R. Scanlon and M. S. Gerber, “Automatic Detection of Cyber Recruitment by Violent Extremists,” *Security Informatics*, vol. 3, no. 1, pp. 1–10, 2014.

[39] A. Agarwal, B. Xie, I. Vovsha, O. Rambow and R. Passonneau, “Sentiment Analysis of Twitter Data,” *Inproceedings of the Workshop on Language in Social Media (LSM 2011)*, Portland, Oregon, pages 30–38, 2011.

[40] A. Abbasi, H. Chen and A. Salem, “Sentiment Analysis in Multiple Languages: Feature Selection for Opinion Classification in Web Forums,” *ACM Transactions on Information Systems*, Vol. 26, pp 1-34, 2008.

[41] H. Chen, W. Chung, J. Qin, E. Reid, M. Sageman, and G. Weimann, “Uncovering the Dark Web: A Case Study of Jihad on the Web,” *Journal of the American Society for Information Science and Technology*, vol. 59, no. 8, pp. 1347–1359, 2008.

[42] Z. Kechaou, M. Ammar, and A. Alimi, “A Mutli-Agent Based System for Sentiment Analysis of User-Generated Content,”. *International Journal on Artificial Intelligence Tools*, Vol 22, 2013. Doi: 10.1142/S0218213013500048.

- [43] M. Asif, A. Ishtiaq, H. Ahmad, H. Aljuaid and J. Shah, "Sentiment analysis of extremism in social media from textual information", *Telematics Informat.*, vol. 48, May 2020.
- [44] X. Chen et al., "Sentiment Classification Using Negative and Intensive Sentiment Supplement Information", *Data Sci. Eng.*, vol. 4, no. 2, pp. 109-118, 2019.
- [45] N. Medagoda, S. Shanmuganathan, "Keywords based temporal sentiment analysis", *Proc. 12th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, pp. 1418-1425, Aug. 2015.
- [46] G. R. S. Weir and T. Ozasa, "Learning from Analysis of Japanese EFL Texts." *Educational Perspectives*, *Journal of the College of Education/University of Hawaii at Manoa*, vol. 43, pp. 56-66, 2010.
- [47] G. R. S. Weir and N. K. Anagnostou, "Exploring Newspapers: A Case Study in Corpus Analysis," in *ICTATLL Workshop 2007*, International Education Centre, Hiroshima International University, Japan, 2007.
- [48] A. Oberacker, "Textual Analysis for Document Forensics", an MSc dissertation submitted to the department of Computer and Information Sciences, University of Strathclyde, United Kingdom, August, 2017.
- [49] B. Cartwright, L. Nahar, G. R. S. Weir, L. Nahar, K. Padda and R. Frank, "The Weaponisation of Cloud-based Social Media: Prospects for Legislation and Regulation", *Tenth International Conference on Cloud Computing, GRIDs, and Virtualisation*, 2019.

- [50] X. Zhang, J. Zhao, Y. LeCun, "Character-level convolutional networks for text classification", arXiv preprint arXiv:1509.01626. 2015.
- [51] S. Ahmad M. Z. Asghar F. M. Alotaibi I. Awan, "Detection and classification of social media-based extremist affiliations using sentiment analysis techniques", *Human-centric Computing and Information Sciences* vol. 9, pp. 24, 2019.
- [52] S. A. Azizan and I. A. Aziz, "Terrorism Detection Based on Sentiment Analysis Using Machine Learning", *Journal of Engineering and Applied Sciences*. Vol 12, pp 691-698, 2017.
- [53] D. Ali, M. M. S. Missen, and M. Husnain, "Multiclass event classification from text," *Scientific Programming*, vol. 2021, Article ID 6660651, 15 pages, 2021.
- [54] Z. Kastrati, L. Ahmed., A. Kurti, F. Kadriu, D. Murtezaj, & F. Gashi, "A deep learning sentiment analyzer for social media comments in low-resource languages", *Electronics*, Vol.10, 1-19, 2021.
- [55] F. Rustam, M. Khalid, W. Aslam, V. Rupapara, A. Mehmood and G. S. Choi, "A performance comparison of supervised machine learning models for COVID-19 tweets sentiment analysis", *PLoS ONE*, vol. 16, no. 2, Feb. 2021
- [56] L Nizzoli, M Avvenuti, S Cresci, M Tesconi, "Extremist Propaganda Tweet Classification with Deep Learning in Realistic Scenarios", In 11th ACM Conference on Web Science (WebSci '19), 2019.

- [57] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, “Multiple imputation by chained equations: what is it and how does it work”, *International Journal of Methods in Psychiatric Research*, Vol. 20, pp40-49, 2011.
- [58] I. B. Helenowski, “Advantages and advancements of multiple imputation”, *Biometrics and Biostatistics International Journal*, Vol.2, pp93-94, 2015.
- [59] N. J. Horton & S. R. Lipsitz, “Multiple imputation in practice: Comparison of software packages for regression models with missing variables”, *The American Statistician*, Vol. 55, 244-254, 2001.
- [60] D. Schunk, “A Markov Chain Monte Carlo algorithm for multiple imputation in large surveys”, *Advances in Statistical Analysis*, Vol.92, pp101-114, 2008.
- [61] J. W. Graham, “Missing data analysis: Making it work in the real world”, *Annual Review of Psychology*, Vol.60, pp549-576, 2009.
- [62] J. I. Schafer, “Multiple imputation: a primer”, *Statistical Methods in Medical Research*, Vol. 8, pp3-15, 1999.
- [63] D. B. Rubin, “Multiple imputations for nonresponse in surveys”, New York: John Wiley and Sons, 1987.
- [64] C. M Salgado, C. Azevedo, H. Proenca, S. M. Vieira, “Missing data”, *Secondary Analysis of Electronic Health Records*, Vol. 1, pp143-162, 2016.

- [65] R. Samant, & S. Rao, "Effects of missing data imputation on classifier accuracy", *International Journal of Engineering Research and Technology*, Vol.2, pp264-266, 2013.
- [66] M. Moeur, & A. R. Stage, "Most similar neighbor: An improved sampling inference procedure for natural resource planning", *Forest Science*, Vol.41, pp337-359, 1995.
- [67] L. Beretta, & A. Santaniello, "Nearest neighbor imputation algorithms: a critical evaluation", *BMC medical Informatics and Decision Making*, Vol.16, pp197-208, 2016.
- [68] S. Bhattacharya, "Nearest neighbor classifiers with improved accuracy and efficiency", [M.Sc. Thesis]. University of Texas, USA, 2017.
- [69] K. Grace-Martin, "Missing data: Two big problems with mean imputation", Retrieved from <https://www.theanalysisfactor.com/mean-imputation/>, 17th March, 2021.
- [70] M. Jamshidian, & M. Mata, "Advances in analysis of mean and covariance structure when data are incomplete in Handbook of Latent Variable and Related Models", *Handbook of Computing and Statistics with Applications*, Vol.1, pp21-44, 2007.
- [71] C. K Enders, "Applied missing data analysis", New York: The Guilford Press, 2010.

- [72] M. Jamshidian, & P. M. Bentler, "ML estimation of mean and covariance structures with missing data using complete data routines", *Journal of Educational and Behavioral Statistics*, 24[1], 21-41, 1999.
- [73] D. J. Stekhoven, & P. Buhlmann, "MissForest – nonparametric missing value imputation for mixed type data", *Bioinformatics*, Vol.28, pp112-118, 2011.
- [74] Y. Andre, "Imputation algorithm? Say goodbye to KNN-Impute. Assessed at <https://towardsdatascience.com/missforest-the-best-missing-data-imputation-algorithm-4d01182aed3> [March 17, 2021]
- [75] L. Morgan, "MissForest – missing data imputation using iterated random forests", Assessed at <https://rpubs.com/lmorgan95/MissForest#:~:text=Disadvantages%3A,object%20you%20can%20store%20somewhere>. [March 17, 2021]
- [76] D. T. A. Eisenberg, "Telomere length measurement validity: the coefficient of variation is invalid and cannot be used to compare quantitative polymerase chain reaction and Southern blot telomere length measurement technique". *International Journal of Epidemiology*. Vol.45, pp. 1295–1298, 2016. doi:10.1093/ije/dyw191. ISSN 0300-5771. PMID 27581804.
- [77] D. P. Doane and L E. Seward, "Measuring skewness: a forgotten statistic", *Journal of Statistics Education*, Vol 19, pp1-18, 2011

- [78] M. Shanker, M.Y Hu, M.S Hung, “Effect of Data Standardisation on Neural Network Training”, Omega, Int. J. Mgmt Sci. Vol 24. No.4, pp 385-397, 1996.
- [79] G. Weir, K. Owoeye, A. Oberacker and H. Alshahrani, “Cloud-based textual analysis as a basis for document classification”, International Conference on High Performance Computing & Simulation (HPCS), pp. 672-676, July 2018
- [80] J. R. Quinlan, " C4.5 Programs for Machine Learning", Morgan Kaufmann, San Mateo, 1993.
- [81] KK Hiran, RK Jain, K Lakhwani, R Doshi, “Machine Learning: Master Supervised and Unsupervised Learning Algorithms with Real Examples (English Edition)”, BPB Publications, pp146-147, 2021.
- [82] SJ Buckley, RJ Harvey, Z Shan, “Application of the random forest algorithm to Streptococcus pyogenes response regulator allele variation: from machine learning to evolutionary models”, Sci Rep 11, 12687 (2021). <https://doi.org/10.1038/s41598-021-91941-6>
- [83] S. Suthaharan, “Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning”, Springer Publishing Company, Incorporated, 1st edition, 2015.
- [84] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, “ Data Mining: Practical Machine Learning tools and techniques”, Morgan Kaufmann, 2011.
- [85] G. Forman, “An Extensive Empirical Study of Feature Selection Metrics for Text Classification”, Journal of Machine Learning Research, 2003. Vol.3, pp1289-1305.

[86] K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan, “ Data Mining: A Knowledge Discovery Approach”, Springer-Verlag New York, Inc.,Secaucus, NJ, USA, 2007.

[87] E. Elgeldawi, A. Sayed, A. R Galal, A.M. Zaki, A, “Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis”, in Informatics, In Informatics, vol. 8, no. 4, pp. 79. Multidisciplinary Digital Publishing Institute, 2021.

[88] P. Probst, M. N. Wright and A. L. Boulesteix, "Hyperparameters and tuning strategies for random forest", Wiley Interdisciplinary Rev. Data Mining Knowl. Discov., vol. 9, no. 3, pp. 1-15, 2019.

[89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, "Scikit-learn: Machine learning in Python." the Journal of machine Learning research Vol.12, pp. 2825-2830, 2011.

[90] D Elavarasan, DR Vincent PM, K Srinivasan, and C.Y. Chang, “ A hybrid CFS filter and RF-RFE wrapper-based feature extraction for enhanced agricultural crop yield prediction modeling”, Agriculture, Vol.10, pp.400, 2020.

[91] Q. Chen, Z. Meng, X. Liu, Q Jin, R. Su, “Decision variants for the automatic determination of optimal feature subset in RF-RFE”, Genes, Vol.9, pp.301, 2018.

[92] X. W. Chen, and J. C Jeong, “Enhanced recursive feature elimination”, In Sixth International Conference on Machine Learning and Applications (ICMLA 2007), pp. 429-435. IEEE. 2007, December.

- [93] Y. Bengio, A. Courville and P. Vincent "Representation Learning: A Review and New Perspectives", IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013. pp 1798–1828, Vol. 35.
- [94] Y. Bengio, "Learning Deep Architectures for AI", Foundation and Trends in Machine Learning, vol 2, no 1, pp 1–127, 2009.
- [95] D. Hof, Robert, "Is Artificial Intelligence Finally Coming into Its Own?" MIT Technology Review, MIT Technology Review, 29 Mar. 2016, www.technologyreview.com/s/513696/deep-learning/.
- [96] J. Schmidhuber "Deep Learning in Neural Networks: An Overview", Neural Networks, 2015. Pp 85-117, Vol.61.
- [97] T. Pejman, H. Ardeshir, "Application of a Modular Feedforward Neural Network for Grade Estimation", Natural Resources Research, 2011. Vol.20, pp25–32. doi:10.1007/s11053-011-9135-3.
- [98] S.E Dreyfus, "Artificial Neural Networks, Back Propagation, and the Kelley-Bryson Gradient Procedure", Journal of Guidance, Control, and Dynamics, 1990. Vol.13,pp 926–928
- [99] Niklas Donges, "A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks", BuiltIn Expert Contributor, July 29, 2021. Retrieved from: <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>.

- [100] IBM Cloud Education, “Recurrent Neural Networks”, 14 September 2020. Retrieved from: <https://www.ibm.com/cloud/learn/recurrent-neural-networks>.
- [101] B.C. Mateus, M. Mendes, J.T. Farinha, R. Assis, A.M. Cardoso, “Comparing LSTM and GRU Models to Predict the Condition of a Pulp Paper Press”, *Energies*, Vol.14, pp.6958, 2021. <https://doi.org/10.3390/en14216958>.
- [102] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural Computation*, Vol. 9, pp1735–1780, 1997.
- [103] O’Reilly Media, Retrieved from: <https://www.oreilly.com/library/view/learn-arcore/9781788830409/e24a657a-a5c6-4ff2-b9ea-9418a7a5d24c.xhtml>, 2020.
- [104] D. Kingma and J. Ba, “Adam: A method for stochastic optimisation”, *ICLR*, arXiv preprint arXiv:1412.6980, 2015. URL <http://arxiv.org/abs/1412.6980>.
- [105] K. Janocha and W. M. Czarnecki, “On Loss Functions for Deep Neural Networks in Classification”, arXiv preprint arXiv:1702.05659, 2017.
- [106] R. Reed and R. Marks, *Neural Smithing: Supervised Learning in Feed-forward Artificial Neural Networks*, (Cambridge, Massachusetts, MIT Press, 1999).
- [107] C. M. Bishop, “Neural networks for pattern recognition”, Oxford university press, 1995.
- [108] G. Ian, B. Yoshua, and C. Aaron, “Deep Learning”, 2016. MIT Press.

[109] A. Moujahid, “A Practical Introduction to Deep Learning with Caffe and Python”, Retrieved from adilmoujahid: <http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/>, February 19, 2018.

[110] A. F. Agarap, “Deep learning using rectified linear units (ReLU)”, arXiv preprint arXiv:1803.08375, 2018.

[111] S. Jain,” An Overview of Regularisation Techniques in Deep Learning (with Python code)”, Retrieved from: <https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularisation-techniques/>, April 19, 2018.

[112] S. Nitish, H. Geoffrey, K. Alex, S. Ilya, S.Ruslan ,” Dropout: A simple way to prevent neural networks from overfitting”, in J. Mach. Learn. Res. Vol.15, pp1929–1958, January 2014.

[113] L. Prechelt, “Early stopping - but when?“ Neural Networks”, in Tricks of the Trade - Second Edition 2012, pp. 53-67.

[114] T. Carneiro, R. V. M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, P. P. R. Filho, "Performance analysis of google colaboratory as a tool for accelerating deep learning applications", IEEE Access, vol. 6, pp. 61 677-61 685, 2018.

[115] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-scale machine learning,” 12th USENIX Symposium on Operating, 2016.

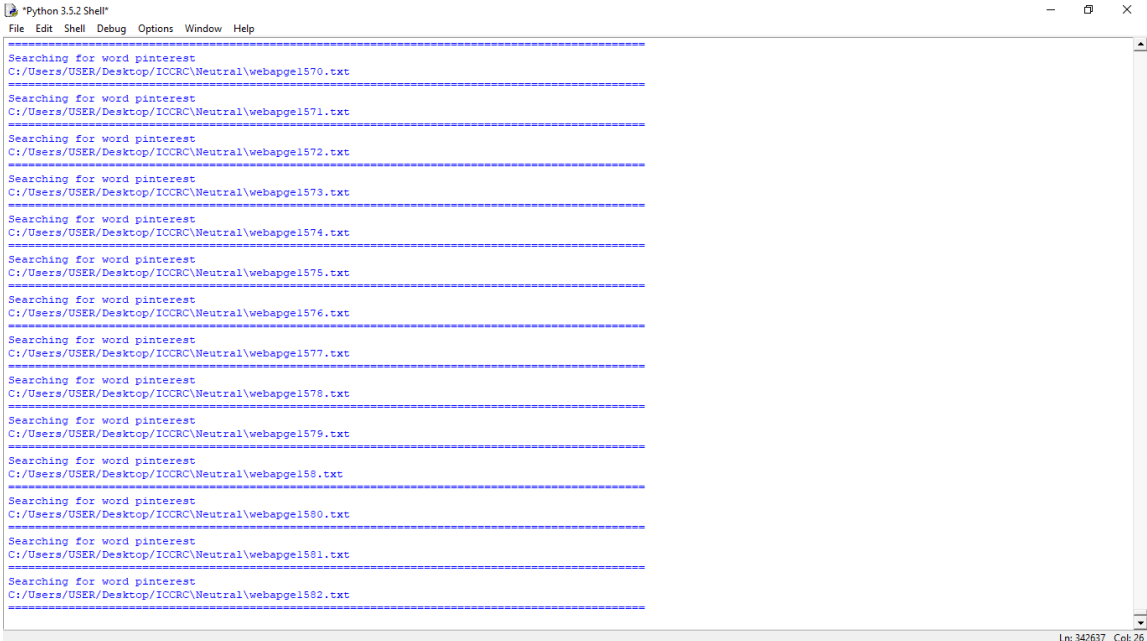
[116] T. C. Kietzmann, P. McClure, and N. Kriegeskorte, “Deep neural networks in computational neuroscience,” bioRxiv, pp.133504-133527, 2018.

[117] J. Brownlee, “Evaluate the Performance of Deep Learning Models in Keras” Retrieved from: <https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/> , May 26, 2016.

Appendices

The details of all the implementations for this thesis can be found in the following.

Appendice A.1: Implementations for the Sentiment Analysis



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1570.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1571.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1572.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1573.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1574.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1575.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1576.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1577.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1578.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1579.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1580.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1581.txt
=====
Searching for word pinterest
C:/Users/USER/Desktop/ICCRC/Neutral/webpage1582.txt
=====
Ln: 342637 Col: 26
```

Numbering the Webpages

Numbering the Webpages Implementation

```
from __future__ import division
from collections import Counter
import nltk, re, pprint, os
import re

#regex = r"(?i)((?:\S+\s+){0,3})\bSmartphone\b((?:\S+\s+){0,3})"
```

```

#test_str = "Nokia Lumia 930 Smartphone, Display 5 pollici, Fotocamera 20 MP, 2GB
RAM, Processore Quad-Core 2,2GHz, Memoria 32GB, Windows Phone 8.1, Bianco
[Germania]"

#matches = re.finditer(regex, test_str)

#for matchNum, match in enumerate(matches):
#  matchNum = matchNum + 1

#  print ("Match {matchNum} was found at {start}-{end}: {match}".format(matchNum =
= matchNum, start = match.start(), end = match.end(), match = match.group()))

#  for groupNum in range(0, len(match.groups())):
#    groupNum = groupNum + 1

#    print ("Group {groupNum} found at {start}-{end}: {group}".format(groupNum =
groupNum, start = match.start(groupNum), end = match.end(groupNum), group =
match.group(groupNum)))

indir = '//ds.strath.ac.uk/hdrive/21/kqb16121/cis/windows/Desktop/ICCRC'
#usedWords =
['war','terrorist','weapon','bomb','jihad','attacker','violence','gun','News','ridiculist','progra
m','party','Officer','security','police','nato','safeguard','council','support','cnn','celebrity','ph
oto','peace','islam','america']
#wordfound = []
#word_frequencies = []
#print('=====')
#=====')
i = 0
for subdir, dirs, files in os.walk(indir): #loop sub directory in a folder
    for file in files: #loop files in a folder
        #print os.path.join(subdir, file)
filepath = subdir + os.sep + file    #get the file path
        #content = open(filepath,'rU',encoding="utf-8")    #open a file read, Universal
        #print(filepath)    #display the full path of the file
i += 1
os.rename(filepath,subdir + os.sep+"webapge{ }.txt".format(i))
#
print('=====')
#=====')
#  for line in content:    # loop throught the content of the file
#    text1 = line.strip()    # get the string content of the file
#    noofwords = nltk.word_tokenize(text1.lower())    #conver content to list of
words

```

```

# #print(nltk.pos_tag(noofwords))
# print('words found')
# print(noofwords) #Display the list of words
# cnt = 0
# for word in noofwords: #loop through the list of words to get
frequently used words
# if word in usedWords: #if a word match frequently used word
# wordfound += [word] # create a list containing the frequencies
from each files

# for occ in wordfound: # loop through the frequency list
# print('{} position at {}'.format(occ,noofwords.index(occ))) #position of
keyword in the content of the file
# #noofwords[noofwords.index(occ)] = "###" + occ + "###" #mark the
keyword in the content
# print('{} five words after at {}'.format(occ,noofwords[21:5]))
# # if occ not in word_frequencies: #get distinct word from the frequency
list
# # word_frequencies +=[occ] #create another list containing distinct
word from the frequency list
# #for Z in word_frequencies: #loop through the distinct frequency list
to get the no of occurrence from the frequency list
# # print('{} occurs {}'.format(Z,wordfound.count(Z))) #print the word and the
frequency
# # print('Words after {}'.format(noofwords[noofwords.index(occ):6]))
# print("New Words")
# print( " ".join(str(x) for x in noofwords))
# #text_file = open(indir+"output/"+file, "w")
# #text_file.write(" ".join(str(x) for x in noofwords))
# #text_file.close()
# wordfound = []
# word_frequencies = []
# content.close()

```

Sentistrength Implementation

```

from __future__ import division
from collections import Counter
import re, pprint, os
import re
indir = '//ds.strath.ac.uk/hdrive/21/kqb16121/cis/windows/Desktop/ICCRC'
outdir = '//ds.strath.ac.uk/hdrive/21/kqb16121/cis/windows/Desktop/output/'

```

```

#usedWords =
['dead','war','prison','violence','terrorist','security','enemies','jihad','islamic','sahaba','territo
ry','saint','civilian','counterterrorism','kingdom','taxes','science','president','weapon','cnn','i
sis','police','fighters','soldiers','politics','militants','twitter','memorandum','narcotic']
usedWords =
['islam','war','muslims','news','government','politics','military','jihad','court','rights','affairs'
,'program','security','policy','press','safeguards','president','ebola','crime','twitter','family','s
yria','victims','facebook','trial','cnn']
wordfound = []
output = ""
print('=====')
print("Program Starting.....")
for words in usedWords:
regex = r'(?!)((?:\S+\s+){0,5})\b'+words+r'\b(?:\S+\s+){0,5})'

    for subdir, dirs, files in os.walk(indir): #loop sub directory in a folder
        for file in files: #loop files in a folder
            #print os.path.join(subdir, file)
            filepath = subdir + os.sep + file    #get the file path

                content = open(filepath,'rU',encoding="utf-8")    #open a file read, Universal
                print('Searching for word {}'.format(words))
                print(filepath)

print('=====')

    for line in content:    # loop through the content of the file
        text1 = line.strip()
        text1 = re.sub(r'^\w|', '', text1)
        text1 = text1.replace(" ", ", ")

            #print(text1+"\n")

            #print(text1)
            matches = re.finditer(regex, text1.lower())
            for matchNum, match in enumerate(matches):
matchNum = matchNum + 1
                word = match.group().replace(", ", ", ").replace(" ", ", ").replace(" ", ", ")
                #wordfound.append(word)
                #print('{} {}'.format(filename,word))
text_file = open(outdir+words+".txt", "a",encoding="utf-8")
text_file.write(file + "-" + word + "\n")

```

```

#         for wrd in wordfound:
#             output += file + " " + wrd + "\n"
#text_file = open(outdir+words+".txt", "a",encoding="utf-8")
#text_file.write(output)
#         #text_file.close()
#wordfound = []
#output = ""

```

10 words Around Keyword to Pinpoint Sentiment Implementation

```

from __future__ import division
from collections import Counter
import re, pprint,os
import re
indir = '//ds.strath.ac.uk/hdrive/21/kqb16121/cis/windows/Desktop/ICCRC'
outdir = '//ds.strath.ac.uk/hdrive/21/kqb16121/cis/windows/Desktop/output/'
#usedWords =
['islam','muslims','allah','war','america','right','politics','democracy','europe','cnn','isis','afri
ca','news','video','sport','asia','program','official','security','affairs','media','officer','policy','
government','politics','militants','twitter','memorandum','narcostic']
usedWords = ['taliban','islam','al-
rahman','abdullah','bomber','usama','president','program','government','blood','radio','desi
gner','news','politics','twitter','soldier','attackers','family']
wordfound = []
output = ""
print('=====')
print("Program Starting.....")
for words in usedWords:
    regex = r'(?i)((?:\S+\s+){0,10})\b'+words+r'\b((?:\S+\s+){0,10})'

    for subdir, dirs, files in os.walk(indir): #loop sub directory in a folder
        for file in files: #loop files in a folder
            #print os.path.join(subdir, file)
            filepath = subdir + os.sep + file    #get the file path

            content = open(filepath,'rU',encoding="utf-8")    #open a file read, Universal
            print('Searching for word {}'.format(words))
            print(filepath)

print('=====')

```

```

=====')
    for line in content:          # loop through the content of the file
        text1 = line.strip()
        text1 = re.sub(r'[^\w]', '', text1)
        text1 = text1.replace(" ", ", ")

        #print(text1+"\n")

        #print(text1)
        matches = re.finditer(regex, text1.lower())
        for matchNum, match in enumerate(matches):
matchNum = matchNum + 1
            word = match.group().replace(","," ").replace(" ", " ").replace(" ", " ")
            #wordfound.append(word)
            #print('{} {}'.format(filename,word))
text_file = open(outdir+words+".txt", "a",encoding="utf-8")
text_file.write(file + "-" + word + "\n")












#         for wrd in wordfound:
#             output += file + " " + wrd + "\n"
#text_file = open(outdir+words+".txt", "a",encoding="utf-8")
#text_file.write(output)
#         #text_file.close()
#wordfound = []
#output = ""

```

Implementation of Imputation

Appendice A.2: Sentiment Analysis Output

: PC > Local Disk (C:) > SentStrength_Data

Name	Date modified	Type	Size
 English	29/09/2018 18:26	File folder	
 BoosterWordList.txt	11/06/2010 22:12	Text Document	1 KB
 EmoticonLookupTable.txt	09/06/2010 22:46	Text Document	1 KB
 EmotionLookupTable.txt	20/09/2011 10:38	Text Document	70 KB
 EnglishWordList.txt	30/07/2009 19:38	Text Document	587 KB
 IdiomLookupTable.txt	01/08/2009 20:20	Text Document	1 KB
 IronyTerms.txt	19/01/2017 09:41	Text Document	0 KB
 NegatingWordList.txt	11/06/2010 22:13	Text Document	1 KB
 QuestionWords.txt	07/02/2011 11:08	Text Document	1 KB
 sentistrength-0.1.jar	18/01/2017 11:59	Executable Jar File	6,408 KB
 SlangLookupTable.txt	18/03/2009 17:58	Text Document	2 KB

A Sample of Different Level of Details of SentiStrength

Name	Date modified	Type	Size
affairs.txt	29/09/2018 16:04	Text Document	3 KB
cnn.txt	29/09/2018 16:04	Text Document	1 KB
court.txt	29/09/2018 16:04	Text Document	4 KB
crime.txt	29/09/2018 17:42	Text Document	1 KB
ebola.txt	29/09/2018 17:45	Text Document	0 KB
facebook.txt	29/09/2018 16:04	Text Document	3 KB
family.txt	29/09/2018 16:04	Text Document	1 KB
government.txt	29/09/2018 17:42	Text Document	13 KB
islam.txt	29/09/2018 16:04	Text Document	3 KB
jihad.txt	29/09/2018 16:05	Text Document	2 KB
military.txt	29/09/2018 16:05	Text Document	7 KB
muslim.txt	29/09/2018 16:05	Text Document	2 KB
news.txt	29/09/2018 16:05	Text Document	6 KB
news1_out.txt	29/09/2018 17:56	Text Document	7 KB
policy.txt	29/09/2018 16:05	Text Document	1 KB

A Sample of the Keywords List

1	Positive	Negative	Text
2	1	-1	webapge123.txt-at the commonwealth heads of government meeting in westminster yesterday
3	1	-1	webapge128.txt-gujari village mafa local government area of borno state
4	1	-1	webapge128.txt-tambashe village in dikwa local government area of borno state
5	1	-1	webapge128.txt-from yaro grematalti yunusari local government area is believed to
6	1	-3	webapge132.txt-their western allies the nigerian government insists that the war
7	1	-1	webapge132.txt- where the limit of government controlled territory is marked
8	1	-2	webapge132.txt-headquarters in maiduguri the government insisted he was shot
9	1	-1	webapge132.txt-which is that the nigerian government doesn t give a
10	1	-1	webapge132.txt-workers reckon that the nigerian government has posted no more
11	2	-1	webapge134.txt-i m hopeful that the government would listen to your
12	1	-1	webapge134.txt-said so far the nigerian government has appeared impotent in
13	1	-2	webapge134.txt-few options left for a government facing strong criticism from
14	1	-1	webapge136.txt-aid efforts and undercut government authority in nigeria and
15	1	-3	webapge137.txt-the islamists northeast heartland government planes attacked the fighters
16	1	-1	webapge137.txt-276 girls kidnapped from the government girls secondary school in
17	1	-3	webapge137.txt-the failure of the nigerian government to find the girls
18	1	-2	webapge138.txt-notably absent is the nigerian government it published an
19	1	-2	webapge143.txt-even swap which the nigerian government refused demanding that

```

2 1 -1 webapge123.txt-at the commonwealth heads of government meeting in westminster yesterda
3 1 -1 webapge128.txt-gujari village mafa local government area of borno state
4 1 -1 webapge128.txt-tambashe village in dikwa local government area of borno state
5 1 -1 webapge128.txt-from yaro grematalti yunusari local government area is believed to
6 1 -3 webapge132.txt-their western allies the nigerian government insists that the war
7 1 -1 webapge132.txt- where the limit of government controlled territory is marked
8 1 -2 webapge132.txt-headquarters in maiduguri the government insisted he was shot
9 1 -1 webapge132.txt-which is that the nigerian government doesn t give a
0 1 -1 webapge132.txt-workers reckon that the nigerian government has posted no more
1 2 -1 webapge134.txt-i m hopeful that the government would listen to your
2 1 -1 webapge134.txt-said so far the nigerian government has appeared impotent in
3 1 -2 webapge134.txt-few options left for a government facing strong criticism from
4 1 -1 webapge136.txt-aid efforts and undercut government authority in nigeria and
5 1 -3 webapge137.txt-the islamists northeast heartland government planes attacked the fight
6 1 -1 webapge137.txt-276 girls kidnapped from the government girls secondary school in
7 1 -3 webapge137.txt-the failure of the nigerian government to find the girls
8 1 -2 webapge138.txt-notably absent is the nigerian government it published an

```

A Sample of Sentiment Scores across the Keywords

Imputation Implementation for MICE, KNN and MF

```

# -*- coding: utf-8 -*-

"""Imputations Dataset Complexion.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1iIEOqGekPqpLRe_45ZkBOCKsMMZD6OCE

"""

```

```
# Commented out IPython magic to ensure Python compatibility.
from scipy.io import arff
from scipy.io.arff import loadarff
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
import warnings
warnings.filterwarnings("ignore")

data1 = arff.loadarff('KNN_Imputation.arff')
knn = pd.DataFrame(data1[0])

data2 = arff.loadarff('MF_Imputation.arff')
mf = pd.DataFrame(data2[0])

data3 = arff.loadarff('MICE_Imputation.arff')
mice = pd.DataFrame(data3[0])

data4 = arff.loadarff('POSIT.arff')
posit = pd.DataFrame(data4[0])

knn.head()

mf.head()
```

```

mice.head()

posit.head()

def do_hist(df, series):
    x = df.iloc[series].to_list()
    n = len(x)
    r = max(x) - min(x)
    root = math.sqrt(n)
    b = int(root + 1)

    plt.hist(x, bins = b)
    plt.show()

descriptions = []
num_feature_words = len(list(knn.columns))-1

knn_desc = knn.describe()
print("Statistical summary of KNN Imputation dataset", '\n')
display(knn_desc)
descriptions.append(knn_desc)

#visualizations
ind = 0
for i in list(knn_desc.index.values):
    print('\n'*2, "Histogram for ", i, " across all features in KNN Imputation dataset")
    do_hist(knn_desc, ind)
    print('\n'*2)
    ind+=1

```

```

mf_desc = mf.describe()
print("Statistical summary of MF Imputation dataset", '\n')
display(mf_desc)
descriptions.append(mf_desc)

#visualizations
ind = 0
for i in list(mf_desc.index.values):
    print('\n'*2, "Histogram for ", i, " across all features in MF Imputation dataset")
    do_hist(mf_desc, ind)
    print('\n'*2)
    ind+=1

mice_desc = mice.describe()
print("Statistical summary of MICE Imputation dataset", '\n')
display(mice_desc)
descriptions.append(mice_desc)

#visualizations
ind = 0
for i in list(mice_desc.index.values):
    print('\n'*2, "Histogram for ", i, " across all features in MICE Imputation dataset")
    do_hist(mice_desc, ind)
    print('\n'*2)
    ind+=1

c_knn = descriptions[0]
old = list(c_knn.columns)

```

```

new = []
for i in old:
    i = "knn_" + i
    new.append(i)
c_knn.columns = new
#display(c_knn)

c_mf = descriptions[1]
old = list(c_mf.columns)
new = []
for i in old:
    i = "mf_" + i
    new.append(i)
c_mf.columns = new
#display(c_mf)

c_mice = descriptions[2]
old = list(c_mice.columns)
new = []
for i in old:
    i = "mice_" + i
    new.append(i)
c_mice.columns = new
#display(c_mice)

all_dfs = pd.concat([c_knn, c_mf, c_mice], axis = 1)
display(all_dfs)

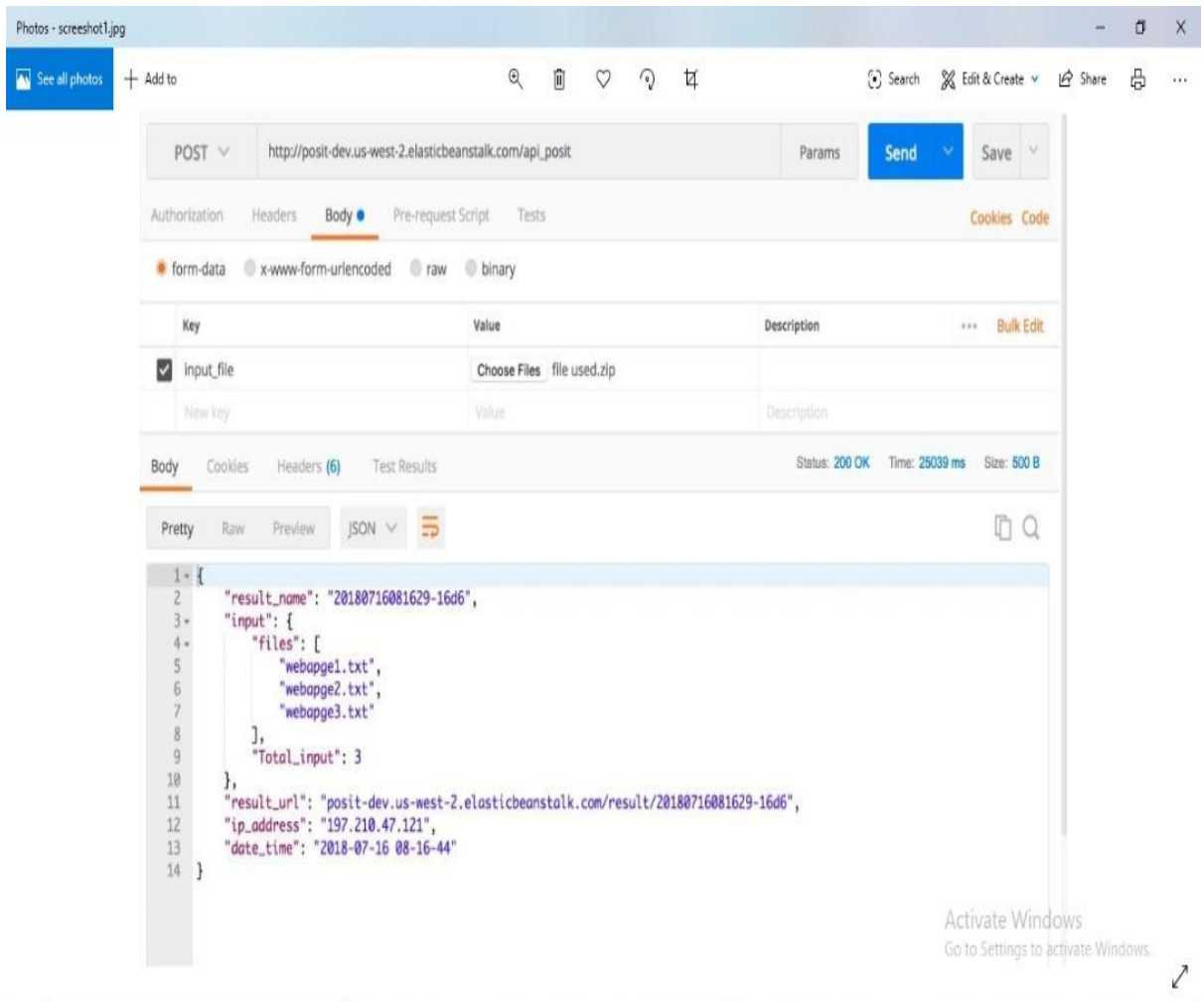
#visualizations

```

```
ind = 0
for i in list(all_dfs.index.values):
    print('\n'*2, "Histogram for ", i, " across all features in All datasets")
    do_hist(all_dfs, ind)
    print('\n'*2)
    ind+=1
```

Appendice B1











Posit Analysis Output Data



The Posit-API version

aggregates	File folder					16/09/2018 10:45
pos_types	File folder					16/09/2018 10:45
pos_tokens.txt	Text Document	2 KB	No	6 KB	73%	16/09/2018 10:45
pos_totals.txt	Text Document	1 KB	No	1 KB	55%	16/09/2018 10:45
summary.txt	Text Document	1 KB	No	1 KB	46%	16/09/2018 10:45
tagged_text.txt	Text Document	2 KB	No	6 KB	64%	16/09/2018 10:45

Different Level of Details of Posit Analysis for Each Webpage

 adjective_types.agg	16/09/2018 12:19	AGG File	1 KB
 adverb_types.agg	16/09/2018 12:19	AGG File	1 KB
 determiner_types.agg	16/09/2018 12:19	AGG File	1 KB
 interjection_types.agg	16/09/2018 12:19	AGG File	1 KB
 noun_types.agg	16/09/2018 12:19	AGG File	1 KB
 particle_types.agg	16/09/2018 12:19	AGG File	1 KB
 personal_pronoun_types.agg	16/09/2018 12:19	AGG File	1 KB
 possessive_pronoun_types.agg	16/09/2018 12:19	AGG File	1 KB
 preposition_types.agg	16/09/2018 12:19	AGG File	1 KB
 verb_types.agg	16/09/2018 12:19	AGG File	1 KB

A Sample of Aggregates for a Webpage

```

0 adjective_comparatives.txt
33 adjective_or_numerals_ordinals.txt
1 adjective_superlatives.txt
34 total

```

A Sample of Adjective Types for Each Webpage

adjective_comparatives.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45
adjective_or_numeral_ordinals.txt	Text Document	1 KB	No	1 KB	67%	16/09/2018 10:45
adjective_superlatives.txt	Text Document	1 KB	No	1 KB	6%	16/09/2018 10:45
adverb_comparative_form.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45
adverb_form.txt	Text Document	1 KB	No	1 KB	61%	16/09/2018 10:45
adverb_superlative_form.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45
common_nouns.txt	Text Document	1 KB	No	2 KB	72%	16/09/2018 10:45
determiners.txt	Text Document	1 KB	No	1 KB	52%	16/09/2018 10:45
interjections.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45
modal_aux.txt	Text Document	1 KB	No	1 KB	7%	16/09/2018 10:45
nouns_common_plurals.txt	Text Document	1 KB	No	1 KB	71%	16/09/2018 10:45
nouns_proper_plural.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45
particles.txt	Text Document	1 KB	No	1 KB	7%	16/09/2018 10:45
prepositions.txt	Text Document	1 KB	No	1 KB	60%	16/09/2018 10:45
pronouns_personal.txt	Text Document	1 KB	No	1 KB	49%	16/09/2018 10:45
pronouns_possessive.txt	Text Document	1 KB	No	1 KB	49%	16/09/2018 10:45
proper_nouns.txt	Text Document	1 KB	No	1 KB	71%	16/09/2018 10:45
verbs_base_form.txt	Text Document	1 KB	No	1 KB	43%	16/09/2018 10:45
verbs_gerund_form.txt	Text Document	1 KB	No	1 KB	59%	16/09/2018 10:45
verbs_past_form.txt	Text Document	1 KB	No	1 KB	53%	16/09/2018 10:45
verbs_past_participle_form.txt	Text Document	1 KB	No	1 KB	67%	16/09/2018 10:45
verbs_present_3rd_form.txt	Text Document	1 KB	No	1 KB	47%	16/09/2018 10:45
verbs_present_not3rd_form.txt	Text Document	1 KB	No	1 KB	53%	16/09/2018 10:45
wh_adverbs.txt	Text Document	1 KB	No	0 KB	0%	16/09/2018 10:45

A Sample of POS-Types for Each Webpage

```

4 number/nn
4 /nn
3 software/nn
3 set/nn
3 opinion/nn
3 classification/nn
3 analysis/nn
2 word/nn

```

A Sample of Common Nouns for a Web page

```

2 automated/jj
2 article/nn
2 are/vbp
2 an/dt
2 about/in
2 /vbp
2 /jj
1 were/vbd
1 well/rb
1 websites/nns
1 ...../...

```

A Sample Pos-Tokens for a Web page

```
adjective_comparatives:  
adjective_or_numeral_ordinals: 52  
adjective_superlatives: 1  
adverb_comparative_form:  
adverb_form: 13  
adverb_superlative_form:  
common_nouns: 92  
determiners: 42  
interjections:  
modal_aux: 1  
nouns_common_plurals: 58  
nouns_proper_plural:  
particles: 1  
prepositions: 56  
pronouns_personal: 7  
pronouns_possessive: 7  
proper_nouns: 63  
verbs_base_form: 7  
verbs_gerund_form: 13  
verbs_past_form: 10  
verbs_past_participle_form: 22  
verbs_present_3rd_form: 12  
verbs_present_not3rd_form: 9  
wh_adverbs:
```

A Sample of POS-Totals for a Webpage

```

485 :Total words (tokens)
269 :Total unique words (types)
0.554639 :Type/Token Ratio (TTR)
16 :Number of sentences
30.3125 :Average sentence length (ASL)
3333 :Number of characters
6.87216 :Average word length (AWL)

```

NUMBER OF TOKEN TYPES

```

147 :noun_types
54 :verb_types
34 :adjective_types
15 :preposition_types
12 :adverb_types
8 :determiner_types
4 :possessive_pronoun_types
4 :personal_pronoun_types
1 :particle_types
0 :interjection_types

```

NUMBER OF POS TYPES

```

426 :nouns
148 :verbs
112 :prepositions
106 :adjectives
84 :determiners
26 :adverbs
14 :possessive pronouns
14 :personal pronouns
2 :particles
0 :interjections

```

A Sample of Summary generated for a Web page

```

Ekiti/NNP State/NNP Gubernatorial/NNP Election/NNP Poll/NNP using/VBG Posit/NNP Analytical/NNP Tool./NNP
Ekiti/NNP State/NNP Gubernatorial/NNP Election/NNP Poll/NNP using/VBG Posit/NNP Cloud-Based/NNP System./NNP
Introduction/NN
Opinion/NN polling/VBG about/IN attitudes/NNS to/TO the/DT leaders/NNS of/IN various/JJ political/JJ parties/NNS can/MD be/VB fo
to/TO carry/VB out/RP opinion/NN polling/VBG to/TO gauge/VB voting/NN intention/NN
the/DT next/JJ general/JJ election/NN is/VBZ scheduled/VBN to/TO be/VB held/VBN no/RB later/RB than/IN 5/CD May/NNP 2022./CD [(
Most/JJS opinion/NN polls/NNS cover/VBP only/RB Great/NNP Britain/NNP ,, excluding/VBG Northern/NNP Ireland/NNP as/IN its/PRP$
Twitter/NNP has/VBZ been/VBN the/DT major/JJ social/JJ network/NN used/VBN for/IN conducting/VBG polls/NNS eg/NN brexit/NN
We/PRP analyzed/VBD nearly/RB 500,000/CD tweets/NNS related/VBN to/TO the/DT Ekiti/NNP State/NNP Gubernatorial/NNP election/NN ;
A./NN Posit-Textual/JJ Analysis/NN
Posit/NNP textual/JJ analysis/NN toolset/NN is/VBZ a/DT UNIX/NNP scripting/VBG program/NN that/WDT is/VBZ capable/JJ of/IN gene
syntactic/JJ features/NNS of/IN textual/JJ contents/NNS of/IN a/DT web/NN are/VBP obtained/VBN through/IN ,, a/DT textual/JJ ar
In/IN this/DT article/NN ,, we/PRP implemented/VBD our/PRP$ framework/NN on/IN a/DT set/NN of/IN manually/RB classified/VBN ext
In/IN this/DT article/NN ,, we/PRP implemented/VBD a/DT data-mining/JJ algorithm/NN in/IN a/DT knowledge/NN extraction/NN softw
Various/JJ types/NNS of/IN textual/JJ classification/NN techniques/NNS have/VBP

```

A Sample of a Tagged text

Appendice C1

Neural Network and Machine Learning Model Implementations

K-Nearest Neighbors Implementation

```
# -*- coding: utf-8 -*-
```

```
"""KNN-StandardScaler.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

```
https://colab.research.google.com/drive/1p0OzG5zQfUiLdwGh9A7LPZ6czYVj8Eeb
```

```
"""
```

```
!pip install pycm
```

```
!pip install liac-arff
```

```
from sklearn import preprocessing, svm
import pandas as pd
import numpy as np
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
import sklearn.datasets as skds
from pathlib import Path
from scipy.io import arff
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.metrics import plot_confusion_matrix
from pycm import *
import os.path
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
```

```

from sklearn.model_selection import GridSearchCV
from IPython.display import display
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import validation_curve

from google.colab import drive
drive.mount('/content/drive')

DATA_DIR = "drive/MyDrive/dataset/"
all_metric = []
all_features = []
all_accuracy = []
embedded_features = []
runtimes=[]
embedded_runtimes =[]

from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel
from time import time
import itertools
from sklearn.linear_model import LassoCV

def splitData(FEATURES_COUNT,dt):
    classMapping = {label: idx for idx, label in enumerate(np.unique(dt["CLASS"]))}
    dt["CLASS"] = dt["CLASS"].map(classMapping)

    dt_label_class = dt['CLASS'].astype(float)
    dt_features = dt.iloc[:, 0:FEATURES_COUNT].apply(np.ceil)
    RANDOM_SEED = 7
    #train_x, test_x, y_train, y_test = train_test_split(dt_features, dt_label_class,
test_size=0.3, shuffle=True, random_state=RANDOM_SEED)
    scaler = StandardScaler()
    scaler.fit(dt_features)
    X = scaler.transform(dt_features)
    y = dt_label_class
    return (X,y,classMapping)

def roundUp(test_dict):

```

```

K = 3
res = dict()
for key in test_dict:
    # rounding to K using round()
    res[str(key)] = round(test_dict[key], K)
return res

def generate_confusion_matrix(cnf_matrix, classes, normalize=False, title='Confusion
matrix'):
    if normalize:
        cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cnf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cnf_matrix.max() / 2.

    for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
        plt.text(j, i, format(cnf_matrix[i, j], fmt), horizontalalignment="center",
                color="white" if cnf_matrix[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    return cnf_matrix

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation + '
Confusion matrix')

```



```

plt.show()

def evaluate_model(data_x, data_y):
    cv_outer = KFold(5, shuffle=True, random_state=7)
    param_name = "n_neighbors"
    predicted_targets = np.array([])
    actual_targets = np.array([])
    gridSearch = {"mean_train_score": [], "mean_test_score": [], "param_ranges": []}
    gridSearch_df = {}
    param_range = [i for i in range(1, 15, 2)]
    param_grid = {
        'n_neighbors': param_range,
        'weights': ['uniform'],
        'leaf_size': range(1,10),
    }
    index = 0

    for train_ix, test_ix in cv_outer.split(data_x):
        train_x, train_y, test_x, test_y = data_x[train_ix], data_y[train_ix], data_x[test_ix],
        data_y[test_ix]
        # define the model
        model = KNeighborsClassifier()
        # configure the cross-validation procedure
        cv_inner = KFold(3, shuffle=True, random_state=7) # execute search
        # define search
        search = GridSearchCV(model, param_grid, scoring='accuracy', n_jobs=-1,
cv=cv_inner, refit=True, return_train_score=True)
        result = search.fit(train_x, train_y)
        # get the best performing model fit on the whole training set
        best_model = result.best_estimator_
        # evaluate model on the hold out dataset
        predicted_labels = best_model.predict(test_x)
        #train_scores, test_scores = validation_curve(search, train_x, train_y,
param_name="n_neighbors", param_range=param_range,scoring="accuracy", n_jobs=1)
        #print("test_scores", train_scores, test_scores)

        predicted_targets = np.append(predicted_targets, predicted_labels)
        actual_targets = np.append(actual_targets, test_y)
        print("Best Parameter %s :" %search.best_params_)

    cv_results = search.cv_results_
    scores_df = pd.DataFrame(cv_results).sort_values(by='rank_test_score')
    gridSearch_df[index] = search

```

```

    index = index + 1

    plot_grid_search_validation_curve(gridSearch_df, param_name)
    plot_grid_search_validation_curve(gridSearch_df, 'leaf_size')
    return predicted_targets, actual_targets

def plot_grid_search_validation_curve(grids, param_to_vary,
                                     title='Validation Curve', ylim=None,
                                     xlim=None, log=None):
    """Plots train and cross-validation scores from a GridSearchCV instance's
    best params while varying one of those params."""

    plt.clf()
    plt.figure(figsize=(16, 16))
    plot_fn = plt.plot
    if log:
        plot_fn = plt.semilogx
    plt.title(title)
    plt.xlabel(param_to_vary)
    plt.ylabel('Score')

    if (ylim is None):
        plt.ylim(0.0, 1.1)
    else:
        plt.ylim(*ylim)

    if (not (xlim is None)):
        plt.xlim(*xlim)

    lw = 1
    fold = 1
    for index in grids.keys():
        grid = grids[index]

        df_cv_results = pd.DataFrame(grid.cv_results_)
        train_scores_mean = df_cv_results['mean_train_score']
        valid_scores_mean = df_cv_results['mean_test_score']
        train_scores_std = df_cv_results['std_train_score']
        valid_scores_std = df_cv_results['std_test_score']
        param_cols = [c for c in df_cv_results.columns if c[:6] == 'param_']

        param_ranges = [grid.param_grid[p[6:]] for p in param_cols]

```

```

param_ranges_lengths = [len(pr) for pr in param_ranges]

train_scores_mean = np.array(train_scores_mean).reshape(*param_ranges_lengths)
valid_scores_mean = np.array(valid_scores_mean).reshape(*param_ranges_lengths)
train_scores_std = np.array(train_scores_std).reshape(*param_ranges_lengths)
valid_scores_std = np.array(valid_scores_std).reshape(*param_ranges_lengths)

param_to_vary_idx = param_cols.index('param_{}'.format(param_to_vary))

slices = []
for idx, param in enumerate(grid.best_params_):
    if (idx == param_to_vary_idx):
        slices.append(slice(None))
        continue
    best_param_val = grid.best_params_[param]
    idx_of_best_param = 0
    if isinstance(param_ranges[idx], np.ndarray):
        idx_of_best_param = param_ranges[idx].tolist().index(best_param_val)
    else:
        idx_of_best_param = param_ranges[idx].index(best_param_val)
    slices.append(idx_of_best_param)

train_scores_mean = train_scores_mean[tuple(slices)]
valid_scores_mean = valid_scores_mean[tuple(slices)]
train_scores_std = train_scores_std[tuple(slices)]
valid_scores_std = valid_scores_std[tuple(slices)]

param_range = param_ranges[param_to_vary_idx]
print("slices",slices, param_ranges, param_to_vary_idx )
plot_fn(param_range, train_scores_mean, label= "(Fold - {} Training score
)".format(fold), lw=lw)

    """plt.fill_between(param_range, train_scores_mean -
train_scores_std,train_scores_mean + train_scores_std, alpha=0.1,
                        color='r', lw=lw)"""
    plot_fn(param_range, valid_scores_mean, label= "(Fold - {} Cross-validation score
)".format(fold), lw=lw)
    """plt.fill_between(param_range, valid_scores_mean - valid_scores_std,
                        valid_scores_mean + valid_scores_std, alpha=0.1,
                        color='b', lw=lw) """

fold = fold + 1
"""if (not isinstance(param_range[0], numbers.Number))):

```

```

    param_range = [str(x) for x in param_range]'''

plt.legend(loc='lower right')

plt.show()

def classifier(imputation, X, y, class_names):
    print("=====")
    #print(imputation)
    #print("=====")
    -----")
    predicted_target, actual_target = evaluate_model(X, y)
    plot_confusion_matrix(predicted_target, actual_target, imputation, class_names)
    cm = ConfusionMatrix(actual_vector=actual_target,
predict_vector=predicted_target)
    test_acc = accuracy_score(actual_target, predicted_target)

    # print("\n Overall Accuracy Score \t", "%.3f" %(test_acc))

    all_accuracy.append({"score": test_acc, "imputation": imputation})

    metric = [ {"Dataset": imputation,
                "Overall Accuracy":test_acc,
                "Accuracy": roundUp(cm.ACC),
                'F1': roundUp(cm.F1),
                'True Positives': cm.TP,
                'False Positives':cm.FP,
                'False Positives Rate':cm.FPR,
                'Recall': roundUp(cm.TPR) ,
                'Precision':roundUp(cm.PPV)}]
    display(pd.DataFrame(metric))
    all_metric.append(metric)
    #GridSearch_table_plot(model, "max_depth", negative=False)

#print("=====")
=====)

def plotAccuracy(all_accuracy):
    label = []
    scores = []

    for accuracy in all_accuracy:

```

```

label.append(accuracy.get("imputation"))
scores.append(round(accuracy.get("score"), 3))
plt.figure(figsize=(10, 10))
plt.bar(label, scores)
plt.title("Compare accuracy for all datasets")
plt.xlabel("Dataset")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

all_metric = []
all_accuracy = []

for subdir, dirs, files in os.walk("./"+DATA_DIR) :
    for file in files:
        data = arff.loadarff("./"+DATA_DIR + file)
        dt = pd.DataFrame(data[0])
        percentages = values = [i/100 for i in range(15, 115, 15)]
        if file == "MICE_Imputation.arff":
            imputation = "Mice Imputation"
            print("\n =====" + imputation+
"=====")
            FEATURES_COUNT = 26
            X, y, classMapping = splitData(FEATURES_COUNT,dt)
            classifier(imputation, X, y, classMapping)

        elif file == "MF_Imputation.arff":
            imputation = "MF Imputation"
            print("\n =====" + imputation+
"=====")
            FEATURES_COUNT = 26
            X, y, classMapping = splitData(FEATURES_COUNT,dt)
            classifier(imputation, X, y, classMapping)

        elif file == "KNN_Imputation.arff":
            FEATURES_COUNT = 26
            imputation = "KNN Imputation"
            print("\n =====" + imputation+
"=====")
            FEATURES_COUNT = 26
            X, y, classMapping = splitData(FEATURES_COUNT,dt)
            classifier(imputation, X, y, classMapping)
        elif file == "combine-posit.arff":
            FEATURES_COUNT = 53

```

```

    imputation = "POSIT + MICE"
    print("\n =====" + imputation+
"=====")
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)
elif file == "combine-postchar.arff":
    FEATURES_COUNT = 71
    imputation = "POSIT + CHAR"
    print("\n =====" + imputation+
"=====")
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)
if file == "POSIT.arff":
    FEATURES_COUNT = 27
    imputation = "POSIT"
    print("\n =====" + imputation+
"=====")
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)

df = pd.DataFrame(all_metric)
print(df.to_string())
plotAccuracy(all_accuracy)

```

Random Forest Implementation

-*- coding: utf-8 -*-

"""Random Forest - StandardScaler.ipynb

Automatically generated by Colaboratory.

Original file is located at

```

https://colab.research.google.com/drive/1ahv49zfuYxknj3KNLGNZ8P25xjNqmt20
"""

!pip install pycm
!pip install liac-arff

from sklearn import preprocessing, svm
import pandas as pd
import numpy as np
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout
from sklearn.preprocessing import MultiLabelBinarizer ,LabelEncoder
import sklearn.datasets as skds
from pathlib import Path
from scipy.io import arff
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.metrics import plot_confusion_matrix
from pycm import *
import os.path
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from IPython.display import display
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

from google.colab import drive
drive.mount('/content/drive')

DATA_DIR = "drive/MyDrive/dataset/"
all_metric = []
all_features = []
all_accuracy = []
embedded_features = []
runtimes=[]

```

```

embedded_runtimes =[]

from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel
from time import time
import itertools
from sklearn.linear_model import LassoCV

def splitData(FEATURES_COUNT,dt):
    classMapping = {label: idx for idx, label in enumerate(np.unique(dt["CLASS"]))}
    dt["CLASS"] = dt["CLASS"].map(classMapping)

    dt_label_class = dt['CLASS'].astype(float)
    dt_features = dt.iloc[:, 0:FEATURES_COUNT].apply(np.ceil)
    RANDOM_SEED = 7
    #train_x, test_x, y_train, y_test = train_test_split(dt_features, dt_label_class,
test_size=0.3, shuffle=True, random_state=RANDOM_SEED)
    scaler = StandardScaler()
    scaler.fit(dt_features)
    X = scaler.transform(dt_features)
    y = dt_label_class
    return (X,y,classMapping)

def featureSelection(imputation, percentage, X, y, noOfFeatures):
    # X = X_train.append(X_test)
    #y = y_train.append(y_test)
    start = time()
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=noOfFeatures)
    model = RandomForestClassifier()
    pipeline = Pipeline(steps=[('s',rfe),('m',model)])
    # evaluate model
    cv = KFold(n_splits=5, shuffle=True, random_state=7)    # execute search
    n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
    stop = round(time() - start, 3);
    # report performance
    test_acc = mean(n_scores)
    #print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
    # report performance

```



```

all_features.append(round(test_acc, 3))
runtimes.append(stop)

def EmbeddedfeatureSelection(imputation, percentage,X, y, noOfFeatures):
    # X = X_train.append(X_test)
    #y = y_train.append(y_test)
    model = None
    start = time()
    # execute search
    fs = SelectFromModel(RandomForestClassifier(), max_features=noOfFeatures)
    model = RandomForestClassifier()
    pipeline = Pipeline(steps=[('s',fs),('m',model)])
    # evaluate model
    cv = KFold(n_splits=5, shuffle=True, random_state=7)    # execute search
    n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')

    stop = round(time() - start, 3);
    test_acc = mean(n_scores)
    # report performance
    embedded_features.append(test_acc)
    embedded_runtimes.append(stop)

def roundUp(test_dict):
    K = 3
    res = dict()
    for key in test_dict:
        # rounding to K using round()
        res[str(key)] = round(test_dict[key], K)
    return res

def generate_confusion_matrix(cnf_matrix, classes, normalize=False, title='Confusion
matrix'):
    if normalize:
        cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cnf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))

```

```

plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cnf_matrix.max() / 2.

for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
    plt.text(j, i, format(cnf_matrix[i, j], fmt), horizontalalignment="center",
            color="white" if cnf_matrix[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

return cnf_matrix

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation + '
Confusion matrix')
    plt.show()

def evaluate_model(data_x, data_y, imputation):
    cv_outer = KFold(5, shuffle=True, random_state=7)
    param_name = "max_depth"
    predicted_targets = np.array([])
    actual_targets = np.array([])
    gridSearch = {"mean_train_score": [], "mean_test_score": [], "param_ranges": []}
    gridSearch_df = {}
    param_grid = {
        "criterion": ["entropy"],
        "max_depth": [i for i in np.arange(1, 21)],
        "min_samples_leaf": range(2,5)
    }
    index = 0
    for train_ix, test_ix in cv_outer.split(data_x):
        train_x, train_y, test_x, test_y = data_x[train_ix], data_y[train_ix], data_x[test_ix],
data_y[test_ix]
        # define the model
        model = RandomForestClassifier()

```

```

# configure the cross-validation procedure
cv_inner = KFold(3, shuffle=True, random_state=7) # execute search
# define search
search = GridSearchCV(model, param_grid, scoring='accuracy', n_jobs=1,
cv=cv_inner, refit=True, return_train_score=True)
result = search.fit(train_x, train_y)
# get the best performing model fit on the whole training set
best_model = result.best_estimator_
# evaluate model on the hold out dataset
predicted_labels = best_model.predict(test_x)

predicted_targets = np.append(predicted_targets, predicted_labels)
actual_targets = np.append(actual_targets, test_y)
print("Best Parameter %s :" %search.best_params_)

cv_results = search.cv_results_
scores_df = pd.DataFrame(cv_results).sort_values(by='rank_test_score')
gridSearch_df[index] = search

index = index + 1

plot_grid_search_validation_curve(gridSearch_df, param_name, title="Validation
Curve - "+imputation)
return predicted_targets, actual_targets

def plot_grid_search_validation_curve(grids, param_to_vary,
title='Validation Curve', ylim=None,
xlim=None, log=None):
"""Plots train and cross-validation scores from a GridSearchCV instance's
best params while varying one of those params."""

plt.clf()
plt.figure(figsize=(16, 16))
plot_fn = plt.plot
if log:
plot_fn = plt.semilogx
plt.title(title)
plt.xlabel(param_to_vary)
plt.ylabel('Score')

if (ylim is None):
plt.ylim(0.0, 1.1)
else:

```

```

plt.ylim(*ylim)

if (not (xlim is None)):
    plt.xlim(*xlim)

lw = 1
fold = 1
for index in grids.keys():
    grid = grids[index]

    df_cv_results = pd.DataFrame(grid.cv_results_)
    train_scores_mean = df_cv_results['mean_train_score']
    valid_scores_mean = df_cv_results['mean_test_score']
    train_scores_std = df_cv_results['std_train_score']
    valid_scores_std = df_cv_results['std_test_score']
    param_cols = [c for c in df_cv_results.columns if c[:6] == 'param_']

    param_ranges = [grid.param_grid[p[6:]] for p in param_cols]
    param_ranges_lengths = [len(pr) for pr in param_ranges]

    train_scores_mean = np.array(train_scores_mean).reshape(*param_ranges_lengths)
    valid_scores_mean = np.array(valid_scores_mean).reshape(*param_ranges_lengths)
    train_scores_std = np.array(train_scores_std).reshape(*param_ranges_lengths)
    valid_scores_std = np.array(valid_scores_std).reshape(*param_ranges_lengths)

    param_to_vary_idx = param_cols.index('param_{}'.format(param_to_vary))

    slices = []
    for idx, param in enumerate(grid.best_params_):
        if (idx == param_to_vary_idx):
            slices.append(slice(None))
            continue
        best_param_val = grid.best_params_[param]
        idx_of_best_param = 0
        if isinstance(param_ranges[idx], np.ndarray):
            idx_of_best_param = param_ranges[idx].tolist().index(best_param_val)
        else:
            idx_of_best_param = param_ranges[idx].index(best_param_val)
        slices.append(idx_of_best_param)

    train_scores_mean = train_scores_mean[tuple(slices)]
    valid_scores_mean = valid_scores_mean[tuple(slices)]
    train_scores_std = train_scores_std[tuple(slices)]

```

```

valid_scores_std = valid_scores_std[tuple(slices)]

param_range = param_ranges[param_to_vary_idx]

plot_fn(param_range, train_scores_mean, label= "(Fold - { } Training score
)".format(fold), lw=lw)

    """plt.fill_between(param_range, train_scores_mean -
train_scores_std,train_scores_mean + train_scores_std, alpha=0.1,
        color='r', lw=lw)"""
    plot_fn(param_range, valid_scores_mean, label= "(Fold - { } Cross-validation score
)".format(fold), lw=lw)
    """plt.fill_between(param_range, valid_scores_mean - valid_scores_std,
        valid_scores_mean + valid_scores_std, alpha=0.1,
        color='b', lw=lw) """

    fold = fold + 1
    """if (not isinstance(param_range[0], numbers.Number)):
        param_range = [str(x) for x in param_range]"""

plt.legend(loc='lower right')

plt.show()
#plt.savefig(title+'.eps', format='eps')

def classifier(imputation, X, y, class_names):
    print("=====")
    #print(imputation)
    #print("-----")
    -----")
    predicted_target, actual_target = evaluate_model(X, y, imputation)
    plot_confusion_matrix(predicted_target, actual_target, imputation, class_names)
    cm = ConfusionMatrix(actual_vector=actual_target,
predict_vector=predicted_target)
    test_acc = accuracy_score(actual_target, predicted_target)

    # print("\n Overall Accuracy Score \t", "%.3f" %(test_acc))

    all_accuracy.append({"score": test_acc, "imputation": imputation})

    metric = [ {"Dataset": imputation,
        "Overall Accuracy":test_acc,
        "Accuracy": roundUp(cm.ACC),

```

```

        'F1': roundUp(cm.F1),
        'True Positives': cm.TP,
        'False Positives':cm.FP,
        'False Positives Rate':roundUp(cm.FPR),
        'Recall': roundUp(cm.TPR) ,
        'Precision':roundUp(cm.PPV)}}
display(pd.DataFrame(metric))
all_metric.append(metric)
#GridSearch_table_plot(model, "max_depth", negative=False)

#print("=====
=====")

def plotAccuracy(all_accuracy):
    label = []
    scores = []

    for accuracy in all_accuracy:

        label.append(accuracy.get("imputation"))
        scores.append(round(accuracy.get("score"), 3))
    plt.figure(figsize=(10, 10))
    plt.bar(label, scores)
    plt.title("Compare accuracy for all datasets")
    plt.xlabel("Dataset")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

def plotFeatureSelection(all_features, percentages, imputation, method, runtimes):

    plt.figure(figsize=(11, 11))
    plt.plot(percentages, all_features, '-o', label='accuracy')
    plt.plot(percentages, runtimes, '-o', label='time')
    for x,y in zip(percentages,runtimes):

        label = "({:.3f}, {:.3f})".format(y,x)

        plt.annotate(label, # this is the text
                    (x,y), # these are the coordinates to position the label
                    textcoords="offset points", # how to position the text
                    xytext=(0,10), # distance from text to points (x,y)
                    ha='center') # horizontal alignment can be left, right or center
    for x,y in zip(percentages,all_features):

```

```

label = "{:.3f}, {:.3f}".format(y,x)

plt.annotate(label, # this is the text
             (x,y), # these are the coordinates to position the label
             textcoords="offset points", # how to position the text
             xytext=(0,10), # distance from text to points (x,y)
             ha='center') # horizontal alignment can be left, right or center
plt.title(method + " Feature Selection " + imputation)
plt.xlabel("Percentage of Features")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

all_metric = []
all_accuracy = []

for subdir, dirs, files in os.walk("./"+DATA_DIR) :
    for file in files:
        data = arff.loadarff("./"+DATA_DIR + file)
        dt = pd.DataFrame(data[0])
        percentages = values = [i/100 for i in range(15, 115, 15)]
        if file == "MICE_Imputation.arff":
            imputation = "Mice Imputation"
            print("\n =====" + imputation +
"=====")
            all_features = []
            embedded_features = []
            runtimes=[]
            embedded_runtimes =[]
            FEATURES_COUNT = 26
            X, y, classMapping = splitData(FEATURES_COUNT,dt)
            for index, p in enumerate(percentages):
                if p > 1:
                    percentages[index] = 1
                    p = 1
                    nf = int(p * FEATURES_COUNT)

                    #print("\n=====Wrapper Feature Selection
=====")
                    featureSelection(imputation, p, X, y, nf)
                    #print("\n=====Embedded Feature Selection
=====")
                    EmbeddedfeatureSelection(imputation, p, X, y, nf)

```

```

        plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
        plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
        classifier(imputation, X, y, classMapping)

elif file == "MF_Imputation.arff":
    imputation = "MF Imputation"
    print("\n===== " + imputation+
"=====")
    FEATURES_COUNT = 26
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)

elif file == "KNN_Imputation.arff":
    FEATURES_COUNT = 26
    imputation = "KNN Imputation"
    print("\n===== " + imputation+
"=====")
    FEATURES_COUNT = 26
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)
elif file == "combine-posit.arff":
    FEATURES_COUNT = 53
    imputation = "POSIT + MICE"
    print("\n ===== " + imputation+
"=====")
    all_features = []
    embedded_features = []
    runtimes=[]
    embedded_runtimes =[]

    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    for index, p in enumerate(percentages):
        if p > 1:
            percentages[index] = 1
            p = 1
            nf = int(p * FEATURES_COUNT)

            #print("\n=====Wrapper Feature Selection
=====")
            featureSelection(imputation, p, X, y, nf)
            #print("\n=====Embedded Feature Selection

```



```

=====")
    EmbeddedfeatureSelection(imputation, p, X, y, nf)

    plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
    plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
    classifier(imputation, X, y, classMapping)

elif file == "combine-postchar.arff":
    FEATURES_COUNT = 71
    imputation = "POSIT + CHAR"
    print("\n =====" + imputation+
"=====")
    all_features = []
    embedded_features = []
    runtimes=[]
    embedded_runtimes =[]
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    for index, p in enumerate(percentages):
        if p > 1:
            percentages[index] = 1
            p = 1
            nf = int(p * FEATURES_COUNT)
            #print("\n=====Wrapper Feature Selection
=====")
            featureSelection(imputation, p, X, y, nf)
            #print("\n=====Embedded Feature Selection
=====")
            EmbeddedfeatureSelection(imputation, p, X, y, nf)

            plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
            plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
            classifier(imputation, X, y, classMapping)

if file == "POSIT.arff":
    FEATURES_COUNT = 27
    imputation = "POSIT"
    print("\n =====" + imputation+
"=====")
    all_features = []
    embedded_features = []

```

```

runtimes=[]
embedded_runtimes =[]
X, y, classMapping = splitData(FEATURES_COUNT,dt)
for index, p in enumerate(percentages):
    if p > 1:
        percentages[index] = 1
        p = 1
        nf = int(p * FEATURES_COUNT)

        #print("\n=====Wrapper Feature Selection
=====")
        featureSelection(imputation, p, X, y, nf)
        #print("\n=====Embedded Feature Selection
=====")
        EmbeddedfeatureSelection(imputation, p, X, y, nf)

        plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
        plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
        classifier(imputation, X, y, classMapping)

df = pd.DataFrame(all_metric)
print(df.to_string())
plotAccuracy(all_accuracy)

```

J48 Implementation

```
!pip install pycm
!pip install liac-arff

from sklearn import preprocessing, svm
import pandas as pd
import numpy as np
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
import sklearn.datasets as skds
from pathlib import Path
from scipy.io import arff
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.metrics import plot_confusion_matrix
from pycm import *
import os.path
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from IPython.display import display
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

from google.colab import drive
drive.mount('/content/drive')

DATA_DIR = "drive/MyDrive/dataset/"
all_metric = []
all_features = []
all_accuracy = []
embedded_features = []
runtimes = []
embedded_runtimes = []
```

```

from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel
from time import time
import itertools
from sklearn.linear_model import LassoCV

def splitData(FEATURES_COUNT,dt):
    classMapping = {label: idx for idx, label in enumerate(np.unique(dt["CLASS"]))}
    dt["CLASS"] = dt["CLASS"].map(classMapping)

    dt_label_class = dt['CLASS'].astype(float)
    dt_features = dt.iloc[:, 0:FEATURES_COUNT].apply(np.ceil)
    RANDOM_SEED = 7
    #train_x, test_x, y_train, y_test = train_test_split(dt_features, dt_label_class,
test_size=0.3, shuffle=True, random_state=RANDOM_SEED)
    scaler = StandardScaler()
    scaler.fit(dt_features)
    X = scaler.transform(dt_features)
    y = dt_label_class
    return (X,y,classMapping)

def featureSelection(imputation, percentage, X, y, noOfFeatures):
    # X = X_train.append(X_test)
    #y = y_train.append(y_test)
    start = time()
    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=noOfFeatures)
    model = DecisionTreeClassifier()
    pipeline = Pipeline(steps=[('s',rfe),('m',model)])
    # evaluate model
    cv = KFold(n_splits=5, shuffle=True, random_state=1) # execute search
    n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
    stop = round(time() - start, 3);
    # report performance
    test_acc = mean(n_scores)
    #print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
    # report performance
    all_features.append(round(test_acc, 3))

```

```

runtimes.append(stop)

def EmbeddedfeatureSelection(imputation, percentage,X, y, noOfFeatures):
    # X = X_train.append(X_test)
    #y = y_train.append(y_test)
    model = None
    start = time()
    # execute search
    fs = SelectFromModel(DecisionTreeClassifier(), max_features=noOfFeatures)
    model = DecisionTreeClassifier()
    pipeline = Pipeline(steps=[('s',fs),('m',model)])
    # evaluate model
    cv = KFold(n_splits=5, shuffle=True, random_state=1)    # execute search
    n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')

    stop = round(time() - start, 3);
    test_acc = mean(n_scores)
    # report performance
    embedded_features.append(test_acc)
    embedded_runtimes.append(stop)

def roundUp(test_dict):
    K = 3
    res = dict()
    for key in test_dict:
        # rounding to K using round()
        res[str(key)] = round(test_dict[key], K)
    return res

def generate_confusion_matrix(cnf_matrix, classes, normalize=False, title='Confusion
matrix'):
    if normalize:
        cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cnf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)

```

```

plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cnf_matrix.max() / 2.

for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
    plt.text(j, i, format(cnf_matrix[i, j], fmt), horizontalalignment="center",
             color="white" if cnf_matrix[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

return cnf_matrix

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation + '
Confusion matrix')
    plt.show()

def evaluate_model(data_x, data_y, imputation):
    cv_outer = KFold(5, shuffle=True, random_state=7)
    param_name = "max_depth"
    predicted_targets = np.array([])
    actual_targets = np.array([])
    gridSearch = {"mean_train_score": [], "mean_test_score": [], "param_ranges": []}
    gridSearch_df = {}
    param_grid = {
        "criterion": ["entropy"],
        "max_depth": [i for i in np.arange(1, 21)],
        "min_samples_leaf": range(2,5)
    }
    index = 0
    for train_ix, test_ix in cv_outer.split(data_x):
        train_x, train_y, test_x, test_y = data_x[train_ix], data_y[train_ix], data_x[test_ix],
data_y[test_ix]
        # define the model
        model = DecisionTreeClassifier(random_state=7)
        cv_inner = KFold(n_splits=3, shuffle=True, random_state=7) # execute search

```

```

# define search
search = GridSearchCV(model, param_grid, scoring='accuracy', n_jobs=1,
cv=cv_inner, refit=True, return_train_score=True)
# configure the cross-validation procedure
result = search.fit(train_x, train_y)
# get the best performing model fit on the whole training set
best_model = result.best_estimator_
# evaluate model on the hold out dataset
predicted_labels = best_model.predict(test_x)

predicted_targets = np.append(predicted_targets, predicted_labels)
actual_targets = np.append(actual_targets, test_y)
print("Best Parameter %s :" %search.best_params_)

cv_results = search.cv_results_
scores_df = pd.DataFrame(cv_results).sort_values(by='rank_test_score')
gridSearch_df[index] = search

index = index + 1

plot_grid_search_validation_curve(gridSearch_df, param_name, title="Validation
Curve - "+imputation)
return predicted_targets, actual_targets

def plot_grid_search_validation_curve(grids, param_to_vary,
title='Validation Curve', ylim=None,
xlim=None, log=None):
"""Plots train and cross-validation scores from a GridSearchCV instance's
best params while varying one of those params."""

plt.clf()
plt.figure(figsize=(16, 16))
plot_fn = plt.plot
if log:
plot_fn = plt.semilogx
plt.title(title)
plt.xlabel(param_to_vary)
plt.ylabel('Score')

if (ylim is None):
plt.ylim(0.0, 1.1)
else:
plt.ylim(*ylim)

```

```

if (not (xlim is None)):
    plt.xlim(*xlim)

lw = 1
fold = 1
for index in grids.keys():
    grid = grids[index]

    df_cv_results = pd.DataFrame(grid.cv_results_)
    train_scores_mean = df_cv_results['mean_train_score']
    valid_scores_mean = df_cv_results['mean_test_score']
    train_scores_std = df_cv_results['std_train_score']
    valid_scores_std = df_cv_results['std_test_score']
    param_cols = [c for c in df_cv_results.columns if c[:6] == 'param_']

    param_ranges = [grid.param_grid[p[6:]] for p in param_cols]
    param_ranges_lengths = [len(pr) for pr in param_ranges]
    train_scores_mean = np.array(train_scores_mean).reshape(*param_ranges_lengths)
    valid_scores_mean = np.array(valid_scores_mean).reshape(*param_ranges_lengths)
    train_scores_std = np.array(train_scores_std).reshape(*param_ranges_lengths)
    valid_scores_std = np.array(valid_scores_std).reshape(*param_ranges_lengths)

    param_to_vary_idx = param_cols.index('param_{}'.format(param_to_vary))

    slices = []
    for idx, param in enumerate(grid.best_params_):
        if (idx == param_to_vary_idx):
            slices.append(slice(None))
            continue
        best_param_val = grid.best_params_[param]
        idx_of_best_param = 0
        if isinstance(param_ranges[idx], np.ndarray):
            idx_of_best_param = param_ranges[idx].tolist().index(best_param_val)
        else:
            idx_of_best_param = param_ranges[idx].index(best_param_val)
        slices.append(idx_of_best_param)

    train_scores_mean = train_scores_mean[tuple(slices)]
    valid_scores_mean = valid_scores_mean[tuple(slices)]
    train_scores_std = train_scores_std[tuple(slices)]
    valid_scores_std = valid_scores_std[tuple(slices)]

    param_range = param_ranges[param_to_vary_idx]

```



```

    plot_fn(param_range, train_scores_mean, label= "(Fold - { } Training score
)".format(fold), lw=lw)

    """plt.fill_between(param_range, train_scores_mean -
train_scores_std,train_scores_mean + train_scores_std, alpha=0.1,
                        color='r', lw=lw)"""
    plot_fn(param_range, valid_scores_mean, label= "(Fold - { } Cross-validation score
)".format(fold), lw=lw)
    """plt.fill_between(param_range, valid_scores_mean - valid_scores_std,
                        valid_scores_mean + valid_scores_std, alpha=0.1,
                        color='b', lw=lw) """

    fold = fold + 1
    """if (not isinstance(param_range[0], numbers.Number)):
        param_range = [str(x) for x in param_range]"""

plt.legend(loc='lower right')

plt.show()

def classifier(imputation, X, y, class_names):
    print("=====")
    #print(imputation)
    #print("-----")
    -----")
    predicted_target, actual_target = evaluate_model(X, y, imputation)
    plot_confusion_matrix(predicted_target, actual_target, imputation, class_names)
    cm = ConfusionMatrix(actual_vector=actual_target,
predict_vector=predicted_target)
    test_acc = accuracy_score(actual_target, predicted_target)

    # print("\n Overall Accuracy Score \t", "%.3f" %(test_acc))

    all_accuracy.append({"score": test_acc, "imputation": imputation})

    metric = [ {"Dataset": imputation,
                "Overall Accuracy":test_acc,
                "Accuracy": roundUp(cm.ACC),
                'F1': roundUp(cm.F1),
                'True Positives': cm.TP,
                'False Positives':cm.FP,
                'False Positives Rate':cm.FPR,
                'Recall': roundUp(cm.TPR) ,

```

```

        'Precision':roundUp(cm.PPV)}}
display(pd.DataFrame(metric))
all_metric.append(metric)
#GridSearch_table_plot(model, "max_depth", negative=False)

#print("=====
=====")

def plot_gridSearch(gridSearch, param_name):
    plt.figure(figsize=(8, 8))
    plot_fn = plt.plot
    train_scores_mean = gridSearch['mean_train_score']
    valid_scores_mean = gridSearch['mean_test_score']
    param_ranges = gridSearch['param_ranges']
    param_ranges_lengths = len(param_ranges)
    train_scores_mean = np.array(train_scores_mean).reshape(*param_ranges_lengths)
    valid_scores_mean = np.array(valid_scores_mean).reshape(*param_ranges_lengths)
    lw = 2
    #param_range = [i for i in np.arange(1, 22)]
    '''if (not isinstance(param_range[0], numbers.Number)):
        param_range = [str(x) for x in param_range]'''

    plot_fn(param_ranges, train_scores_mean, label='Training score', color='r',lw=lw)
    plt.fill_between(param_ranges, train_scores_mean, alpha=0.1,color='r', lw=lw)

    plot_fn(param_ranges, valid_scores_mean, label='Cross-validation score', color='b',
lw=lw)
    plt.fill_between(param_ranges, valid_scores_mean, alpha=0.1, color='b', lw=lw)

    plt.legend(loc='lower right')

    plt.show()

def plotAccuracy(all_accuracy):
    label = []
    scores = []

    for accuracy in all_accuracy:

        label.append(accuracy.get("imputation"))
        scores.append(round(accuracy.get("score"), 3))
    plt.figure(figsize=(10, 10))
    plt.bar(label, scores)
    plt.title("Compare accuracy for all datasets")

```

```

plt.xlabel("Dataset")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

def plotFeatureSelection(all_features, percentages, imputation, method, runtimes):

    plt.figure(figsize=(11, 11))
    plt.plot(percentages, all_features, '-o', label='accuracy')
    plt.plot(percentages, runtimes, '-o', label='time')
    for x,y in zip(percentages,runtimes):

        label = "{:.3f}, {:.3f}".format(y,x)

        plt.annotate(label, # this is the text
                    (x,y), # these are the coordinates to position the label
                    textcoords="offset points", # how to position the text
                    xytext=(0,10), # distance from text to points (x,y)
                    ha='center') # horizontal alignment can be left, right or center
    for x,y in zip(percentages,all_features):

        label = "{:.3f}, {:.3f}".format(y,x)

        plt.annotate(label, # this is the text
                    (x,y), # these are the coordinates to position the label
                    textcoords="offset points", # how to position the text
                    xytext=(0,10), # distance from text to points (x,y)
                    ha='center') # horizontal alignment can be left, right or center
    plt.title(method + " Feature Selection " + imputation)
    plt.xlabel("Percentage of Features")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

all_metric = []
all_accuracy = []

for subdir, dirs, files in os.walk("./"+DATA_DIR) :
    for file in files:
        data = arff.loadarff("./"+DATA_DIR + file)
        dt = pd.DataFrame(data[0])
        percentages = values = [i/100 for i in range(15, 115, 15)]
        if file == "MICE_Imputation.arff":
            imputation = "Mice Imputation"

```

```

print("\n =====" + imputation+
"=====")
    all_features = []
    embedded_features = []
    runtimes=[]
    embedded_runtimes =[]
    FEATURES_COUNT = 26
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    for index, p in enumerate(percentages):
        if p > 1:
            percentages[index] = 1
            p = 1
            nf = int(p * FEATURES_COUNT)

            #print("\n=====Wrapper Feature Selection
=====")
            featureSelection(imputation, p, X, y, nf)
            #print("\n=====Embedded Feature Selection
=====")
            EmbeddedfeatureSelection(imputation, p, X, y, nf)

            plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
            plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
            classifier(imputation, X, y, classMapping)

elif file == "MF_Imputation.arff":
    imputation = "MF Imputation"
    print("\n =====" + imputation+
"=====")
    FEATURES_COUNT = 26
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)

elif file == "KNN_Imputation.arff":
    FEATURES_COUNT = 26
    imputation = "KNN Imputation"
    print("\n =====" + imputation+
"=====")
    FEATURES_COUNT = 26
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    classifier(imputation, X, y, classMapping)
elif file == "combine-posit.arff":

```

```

FEATURES_COUNT = 53
imputation = "POSIT + MICE"

print("\n =====" + imputation+
"=====")
all_features = []
embedded_features = []
runtimes=[]
embedded_runtimes =[]
X, y, classMapping = splitData(FEATURES_COUNT,dt)
for index, p in enumerate(percentages):
    if p > 1:
        percentages[index] = 1
        p = 1
        nf = int(p * FEATURES_COUNT)

        #print("\n=====Wrapper Feature Selection
=====")
        featureSelection(imputation, p, X, y, nf)
        #print("\n=====Embedded Feature Selection
=====")
        EmbeddedfeatureSelection(imputation, p, X, y, nf)

        plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
        plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
        classifier(imputation, X, y, classMapping)

elif file == "combine-postchar.arff":
    FEATURES_COUNT = 71
    imputation = "POSIT + CHAR"

    print("\n =====" + imputation+
"=====")
    all_features = []
    embedded_features = []
    runtimes=[]
    embedded_runtimes =[]
    X, y, classMapping = splitData(FEATURES_COUNT,dt)
    for index, p in enumerate(percentages):
        if p > 1:
            percentages[index] = 1
            p = 1

```

```

nf = int(p * FEATURES_COUNT)

#print("\n=====Wrapper Feature Selection
=====")
featureSelection(imputation, p, X, y, nf)
#print("\n=====Embedded Feature Selection
=====")
EmbeddedfeatureSelection(imputation, p, X, y, nf)

plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
classifier(imputation, X, y, classMapping)
if file == "POSIT.arff":
FEATURES_COUNT = 27
imputation = "POSIT"
print("\n =====" + imputation+
"=====")
all_features = []
embedded_features = []
runtimes=[]
embedded_runtimes =[]
X, y, classMapping = splitData(FEATURES_COUNT,dt)
for index, p in enumerate(percentages):
if p > 1:
percentages[index] = 1
p = 1
nf = int(p * FEATURES_COUNT)

#print("\n=====Wrapper Feature Selection
=====")
featureSelection(imputation, p, X, y, nf)
#print("\n=====Embedded Feature Selection
=====")
EmbeddedfeatureSelection(imputation, p, X, y, nf)

plotFeatureSelection(all_features, percentages, imputation, "Wrapper Method",
runtimes)
plotFeatureSelection(embedded_features, percentages, imputation, "Embedded
Method", embedded_runtimes)
classifier(imputation, X, y, classMapping)

```

```
df = pd.DataFrame(all_metric)
print(df.to_string())
plotAccuracy(all_accuracy)
```

RNN Implementation

```
# -*- coding: utf-8 -*-
"""RNN.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

```
https://colab.research.google.com/drive/1aHkvuP3PBDMGdsZwGif-NVfHJySfZX14
"""
```

```
!pip install pycm
```

```
from google.colab import drive
drive.mount('/content/drive')
FEATURES_COUNT = 0
INPUT_SHAPE = 0
DATA_DIR = "drive/MyDrive/dataset/"
timing = []
all_metric = []
all_accuracy = []
```

```

from sklearn import preprocessing, svm
import pandas as pd
import numpy as np
from keras.utils import np_utils
from keras.wrappers.scikit_learn import KerasClassifier
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, Dropout, Embedding, LSTM, Lambda,
Activation, GlobalMaxPooling1D, SpatialDropout1D

from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
import sklearn.datasets as skds
from pathlib import Path
from scipy.io import arff
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.metrics import plot_confusion_matrix
import keras
import tensorflow as tf
from time import time
import os.path
from pycm import *
from sklearn.model_selection import GridSearchCV
from IPython.display import display
import itertools

class TimingCallback(keras.callbacks.Callback):
    def __init__(self):
        self.logs=[]
    def on_epoch_begin(self, epoch, logs={}):
        self.starttime=time()
    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(time()-self.starttime)

def baseline_model():
    rnn_model=Sequential()

    rnn_model.add(LSTM(150,return_sequences=True,input_shape=(INPUT_SHAPE,1)))

```



```

#input layer
rnn_model.add(tf.keras.layers.LSTM(units=26, activation='tanh'))
rnn_model.add(Dropout(0.5))

rnn_model.add(tf.keras.layers.Dense(units=3, activation='sigmoid')) #output layer
rnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy']) #compiling the model
rnn_model.summary()
return rnn_model

def splitData(FEATURES_COUNT,dt):
    classMapping = {label: idx for idx, label in enumerate(np.unique(dt["CLASS"]))}
    dt["CLASS"] = dt["CLASS"].map(classMapping)

    dt_label_class = dt['CLASS'].astype(float)
    dt_features = dt.iloc[:, 0:FEATURES_COUNT].apply(np.ceil)
    RANDOM_SEED = 7
    train_x, test_x, y_train, y_test = train_test_split(dt_features, dt_label_class,
test_size=0.3, shuffle=True, random_state=RANDOM_SEED)
    X_train = np.array(train_x)
    X_test = np.array(test_x)
    y_train = np.array(y_train)
    y_test = np.array(y_test)
    X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
    X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)
    return (X_train, X_test, y_test,y_train, classMapping)

def GridSearch_table_plot(grid_clf, param_name,
    num_results=15,
    negative=True,
    graph=True,
    display_all_params=False):

    """Display grid search results

    Arguments
    -----

    grid_clf      the estimator resulting from a grid search
                  for example: grid_clf = GridSearchCV( ...

    param_name    a string with the name of the parameter being tested

    num_results   an integer indicating the number of results to display

```

Default: 15

negative boolean: should the sign of the score be reversed?
 scoring = 'neg_log_loss', for instance
 Default: True

graph boolean: should a graph be produced?
 non-numeric parameters (True/False, None) don't graph well
 Default: True

display_all_params boolean: should we print out all of the parameters, not just the
ones searched for?
 Default: True

Usage

```
GridSearch_table_plot(grid_clf, "min_samples_leaf")
```

```
'''
```

```
clf = grid_clf.best_estimator_  
clf_params = grid_clf.best_params_  
if negative:  
    clf_score = -grid_clf.best_score_  
else:  
    clf_score = grid_clf.best_score_  
clf_stdev = grid_clf.cv_results_['std_test_score'][grid_clf.best_index_  
cv_results = grid_clf.cv_results_  
  
#print("best parameters: {}".format(clf_params))  
#print("best score:     {:0.5f} (+/-{:0.5f})".format(clf_score, clf_stdev))  
if display_all_params:  
    import pprint  
    pprint.pprint(clf.get_params())  
  
# pick out the best results  
# =====  
scores_df = pd.DataFrame(cv_results).sort_values(by='rank_test_score')  
  
best_row = scores_df.iloc[0, :]  
if negative:  
    best_mean = -best_row['mean_test_score']
```

```

else:
    best_mean = best_row['mean_test_score']
    best_stdev = best_row['std_test_score']
    best_param = best_row['param_' + param_name]

    # display the top 'num_results' results
    # =====

#display(pd.DataFrame(cv_results).sort_values(by='rank_test_score').head(num_results)
)

# plot the results
# =====
scores_df = scores_df.sort_values(by='param_' + param_name)

if negative:
    means = -scores_df['mean_test_score']
else:
    means = scores_df['mean_test_score']
stds = scores_df['std_test_score']
params = scores_df['param_' + param_name]

# plot
if graph:
    plt.figure(figsize=(8, 8))
    plt.errorbar(params, means, yerr=stds)

    plt.axhline(y=best_mean + best_stdev, color='red')
    plt.axhline(y=best_mean - best_stdev, color='blue')
    plt.plot(best_param, best_mean, 'or')

    plt.title(param_name + " vs Score\nBest Score {:.5f}".format(clf_score))
    plt.xlabel(param_name)
    plt.ylabel('Score')
    plt.show()

def roundUp(test_dict):
    K = 3
    res = dict()
    for key in test_dict:
        # rounding to K using round()
        res[str(key)] = round(test_dict[key], K)
    return res

```

```

def plotAccuracy(all_accuracy):
    label = []
    scores = []

    for accuracy in all_accuracy:

        label.append(accuracy.get("imputation"))
        scores.append(round(accuracy.get("score"), 3))
    plt.figure(figsize=(10, 10))
    plt.bar(label, scores)
    plt.title("Compare accuracy for all datasets")
    plt.xlabel("Dataset")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

def classifier(imputation, max_depth,X_train,y_train,X_test, y_test, class_names):
    print("=====")
    print(imputation)
    print("=====")
    model = None
    cb = TimingCallback()
    es =
EarlyStopping(monitor="loss",min_delta=0,patience=3,verbose=1,mode="auto",baseline
=None,restore_best_weights=False)
    estimator = KerasClassifier(build_fn=baseline_model, batch_size=5, verbose=0)
    #history = model.fit(X_train,
y_train,validation_split=0.33,epochs=1,verbose=1,callbacks=[cb, es])
    #predictions = model.predict(X_test)
    history = estimator.fit(X_train,
y_train,validation_split=0.33,epochs=20,verbose=1,callbacks=[cb, es])
    predictions = estimator.predict(X_test)
    timing.append(cb.logs)
    # list all data in history
    test_acc = accuracy_score(y_test, predictions)
    test_predictions = estimator.predict(X_test)
    cm = ConfusionMatrix(actual_vector=y_test, predict_vector=test_predictions)

    plot_confusion_matrix(predictions, y_test, imputation, class_names)
    print("\n Overall Accuracy Score \t", "%.3f" %(test_acc))
    all_accuracy.append({"score": test_acc, "imputation": imputation})

    metric = [ {"Dataset": imputation,
                "Overall Accuracy":test_acc,

```

```

        "Accuracy": roundUp(cm.ACC),
        'F1': roundUp(cm.F1),
        'True Positives': cm.TP,
        'False Positives':cm.FP,
        'False Positives Rate':roundUp(cm.FPR),
        'Recall': roundUp(cm.TPR) ,
        'Precision':roundUp(cm.PPV)}}

    all_metric.append(metric)
    display(pd.DataFrame(metric))

print("=====
=====")

    epoch_range = range(1,len(history.history['loss']) + 1)
    plt.plot(epoch_range, history.history['accuracy'])
    plt.plot(epoch_range, history.history['val_accuracy'])
    plt.title('Model_accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train','val'], loc='upper left')
    plt.show()

    #plot training and validation loss values
    plt.plot(epoch_range, history.history['loss'])
    plt.plot(epoch_range, history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train','val'], loc='upper left')
    plt.show()

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation +
Confusion matrix')
    plt.show()

def generate_confusion_matrix(cnf_matrix, classes, normalize=False, title='Confusion
matrix'):
```

```

if normalize:
    cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion matrix, without normalization')

plt.imshow(cnf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
plt.title(title)
plt.colorbar()

tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cnf_matrix.max() / 2.

for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
    plt.text(j, i, format(cnf_matrix[i, j], fmt), horizontalalignment="center",
            color="white" if cnf_matrix[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel("True label")
plt.xlabel("Predicted label")

return cnf_matrix

all_metric = []
all_accuracy = []
for subdir, dirs, files in os.walk("./"+DATA_DIR) :
    for file in files:
        data = arff.loadarff("./"+DATA_DIR + file)
        dt = pd.DataFrame(data[0])

        if file == "MICE_Imputation.arff":
            FEATURES_COUNT = 26
            imputation = "Mice Imputation"
            X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
            INPUT_SHAPE = X_train.shape[0]
            classifier(imputation, 1, X_train, y_train, X_test, y_test, classMapping)

        elif file == "MF_Imputation.arff":
            FEATURES_COUNT = 26

```

```

    imputation = "MF Imputation"
    X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 10, X_train, y_train, X_test, y_test, classMapping)

elif file == "KNN_Imputation.arff":
    FEATURES_COUNT = 26
    imputation = "KNN Imputation"
    X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train, y_train, X_test, y_test, classMapping)

elif file == "combine-postchar.arff":
    FEATURES_COUNT = 71
    imputation = "POSIT + MICE"
    X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train, y_train, X_test, y_test, classMapping )

elif file == "combine-posit.arff":
    FEATURES_COUNT = 53
    imputation = "POSIT + CHAR"
    X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 14, X_train, y_train, X_test, y_test, classMapping )

elif file == "POSIT.arff":
    FEATURES_COUNT = 27
    imputation = "POSIT"
    X_train, X_test, y_test, y_train, classMapping =
splitData(FEATURES_COUNT, dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train, y_train, X_test, y_test, classMapping )

df = pd.DataFrame(all_metric)
print(df.to_string())

```

```
plotAccuracy(all_accuracy)
```

MLP Implementation

```
# -*- coding: utf-8 -*-
```

```
"""MLP -Better model.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

```
https://colab.research.google.com/drive/1EcN1XvG0Kj375UvVe6Q0I5AznIByz86k
"""
```

```
!pip install pycm
```

```
!pip install keras-tuner
```

```
!pip install -U imbalanced-learn
```

```
from sklearn import preprocessing, svm
```

```
import pandas as pd
```

```
import numpy as np
```

```
from keras.utils import np_utils
```

```
from keras.wrappers.scikit_learn import KerasClassifier
```

```
from keras.preprocessing.text import Tokenizer
```

```
from keras.models import Sequential
```

```
from keras.layers import Activation, Dense, Dropout
```

```
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder
```

```
import sklearn.datasets as skds
```

```
from pathlib import Path
```

```
from scipy.io import arff
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
from sklearn.model_selection import KFold
```

```
from sklearn.model_selection import cross_val_score
```



```

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from sklearn.metrics import plot_confusion_matrix
import tensorflow as tf
from tensorflow import keras
from time import time
import os.path
from pycm import *
import keras_tuner as kt
from kerastuner import HyperModel
import itertools
from tensorflow.keras import models, layers
from sklearn.model_selection import validation_curve

""""MLP Implementation""""

from google.colab import drive
drive.mount('/content/drive')

timing = []
DATA_DIR = "drive/MyDrive/dataset/"
all_metric = []
all_accuracy = []
FEATURES_COUNT = 0
class_names={ }

class TimingCallback(keras.callbacks.Callback):
    def __init__(self):
        self.logs=[]
    def on_epoch_begin(self, epoch, logs={ }):
        self.starttime=time()
    def on_epoch_end(self, epoch, logs={ }):
        self.logs.append(time()-self.starttime)

def baseline_model():
    num_labels = 3
    input_shape = (FEATURES_COUNT,)
    model = keras.Sequential()

    model = keras.Sequential()
    model.add(Dense(30, input_shape=(FEATURES_COUNT,)))
    model.add(Activation('relu'))

```

```

model.add(Dropout(0.2))

model.add(Dense(20))
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Dense(num_labels))
model.add(Activation('softmax'))
#model.summary()
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
return model

def splitData(FEATURES_COUNT,dt):
    classMapping = {label: idx for idx, label in enumerate(np.unique(dt["CLASS"]))}
    dt["CLASS"] = dt["CLASS"].map(classMapping)

    dt_label_class = dt['CLASS'].astype(float)
    dt_features = dt.iloc[:, 0:FEATURES_COUNT].apply(np.ceil)
    train_x, test_x, y_train, y_test = train_test_split(dt_features, dt_label_class,
test_size=0.3)
    scaler = StandardScaler()
    scaler.fit(train_x)
    X_train = scaler.transform(train_x)
    X_test = scaler.transform(test_x)
    class_names = classMapping
    return (X_train, X_test, y_test, y_train, classMapping)

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation +
Confusion matrix')
    plt.show()

def generate_confusion_matrix(cnf_matrix, classes, normalize=False, title='Confusion
matrix'):
    if normalize:
        cnf_matrix = cnf_matrix.astype('float') / cnf_matrix.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")

```

```

else:
    print('Confusion matrix, without normalization')

plt.imshow(cnf_matrix, interpolation='nearest', cmap=plt.get_cmap('Blues'))
plt.title(title)
plt.colorbar()

tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cnf_matrix.max() / 2.

for i, j in itertools.product(range(cnf_matrix.shape[0]), range(cnf_matrix.shape[1])):
    plt.text(j, i, format(cnf_matrix[i, j], fmt), horizontalalignment="center",
             color="white" if cnf_matrix[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel("True label")
plt.xlabel("Predicted label")

return cnf_matrix

def plot_confusion_matrix(predicted_labels_list, y_test_list, imputation, class_names):
    cnf_matrix = confusion_matrix(y_test_list, predicted_labels_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generate_confusion_matrix(cnf_matrix, classes=class_names, title=imputation + '
Confusion matrix')
    plt.show()

def classifier(imputation, max_depth, X_train, y_train, X_test, y_test, class_names):
    print("=====-----")
    print(imputation)
    print("=====-----")
    model = None
    cb = TimingCallback()
    es =
EarlyStopping(monitor="loss", min_delta=0, patience=3, verbose=1, mode="auto", baseline

```

```

=None,restore_best_weights=False)
    estimator = KerasClassifier(build_fn=baseline_model, batch_size=7, verbose=0)
    #history = model.fit(X_train,
y_train,validation_split=0.33,epochs=1,verbose=1,callbacks=[cb, es])
    #predictions = model.predict(X_test)
    history = estimator.fit(X_train,
y_train,validation_split=0.33,epochs=20,verbose=1,callbacks=[cb, es])
    predictions = estimator.predict(X_test)
    timing.append(cb.logs)
    # list all data in history
    test_acc = accuracy_score(y_test, predictions)
    test_predictions = estimator.predict(X_test)
    plot_confusion_matrix(test_predictions, y_test, imputation, class_names)
    cm = ConfusionMatrix(actual_vector=y_test.values,
predict_vector=test_predictions)
    print("\n Overall Accuracy Score \t", "%.3f" %(test_acc))
    all_accuracy.append({"score": test_acc, "imputation": imputation})

    metric = [ {"Dataset": imputation,
                "Overall Accuracy":test_acc,
                "Accuracy": roundUp(cm.ACC),
                'F1': roundUp(cm.F1),
                'True Positives': cm.TP,
                'False Positives':cm.FP,
                'False Positives Rate':roundUp(cm.FPR),
                'Recall': roundUp(cm.TPR) ,
                'Precision':roundUp(cm.PPV)}]

    all_metric.append(metric)
    display(pd.DataFrame(metric))

print("=====")
=====)

    epoch_range = range(1,len(history.history['loss']) + 1)
    plt.plot(epoch_range, history.history['accuracy'])
    plt.plot(epoch_range, history.history['val_accuracy'])
    plt.title('Model_accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train','val'], loc='upper left')
    plt.show()

```

```

    #plot training and validation loss values
    plt.plot(epoch_range, history.history['loss'])
    plt.plot(epoch_range, history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train','val'], loc='upper left')
    plt.show()

def plotAccuracy(all_accuracy):
    label = []
    scores = []

    for accuracy in all_accuracy:

        label.append(accuracy.get("imputation"))
        scores.append(round(accuracy.get("score"), 3))
    plt.figure(figsize=(10, 10))
    plt.bar(label, scores)
    plt.title("Compare accuracy for all datasets")
    plt.xlabel("Dataset")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

def roundUp(test_dict):
    K = 3
    res = dict()
    for key in test_dict:
        # rounding to K using round()
        res[str(key)] = round(test_dict[key], K)
    return res

all_metric = []
all_accuracy = []
for subdir, dirs, files in os.walk("./"+DATA_DIR) :
    for file in files:
        data = arff.loadarff("./"+DATA_DIR + file)
        dt = pd.DataFrame(data[0])

        if file == "MICE_Imputation.arff":
            FEATURES_COUNT = 26
            imputation = "Mice Imputation"
            X_train, X_test, y_test,y_train, classMapping =

```

```

splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train,y_train, X_test, y_test, classMapping)

elif file == "MF_Imputation.arff":
    FEATURES_COUNT = 26
    imputation = "MF Imputation"
    X_train, X_test, y_test,y_train, classMapping =
splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 10, X_train,y_train, X_test, y_test,classMapping)

elif file == "KNN_Imputation.arff":
    FEATURES_COUNT = 26
    imputation = "KNN Imputation"
    X_train, X_test, y_test,y_train, classMapping =
splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train,y_train, X_test, y_test, classMapping)

elif file == "combine-postchar.arff":
    FEATURES_COUNT = 71
    imputation = "POSIT + MICE"
    X_train, X_test, y_test,y_train, classMapping =
splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train,y_train, X_test, y_test,classMapping )

elif file == "combine-posit.arff":
    FEATURES_COUNT = 53
    imputation = "POSIT + CHAR"
    X_train, X_test, y_test,y_train, classMapping =
splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 14, X_train,y_train, X_test, y_test,classMapping )

elif file == "POSIT.arff":
    FEATURES_COUNT = 27
    imputation = "POSIT"
    X_train, X_test, y_test,y_train, classMapping =
splitData(FEATURES_COUNT,dt)
    INPUT_SHAPE = X_train.shape[0]
    classifier(imputation, 1, X_train,y_train, X_test, y_test,classMapping )

```

```
df = pd.DataFrame(all_metric)
print(df.to_string())
plotAccuracy(all_accuracy)
```