AN OPTIMIZED MACHINE LEARNING PIPELINE FOR DETECTING RACIST COMMENTS WRITTEN IN SINHALA LANGUAGE

Chandrajith Priyadarshana Bandara Senadheera

189347N

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

February 2021

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: …… Date: …….

Name: C.P.B. Senadheera

The supervisor should certify the thesis with the following declaration.

The above candidate has carried out research for the Masters thesis under my supervision.

Name: Dr. Uthyasanker Thayasivam.

Signature of the supervisor: ……. Date: …….

ACKNOWLEDGEMENT

I warmly convey my gratitude to my supervisor Dr. Uthyasanker Thayasivam, for his continuous guidance and support. His patience and knowledge inspired me and helped me to improve this study. He always steered me in the right direction whenever he thought I needed it and motivated me to complete my research.

Additionally, I appreciate the support provided by all the other university lecturers of the Department of Computer Science and Engineering, who supported me from the beginning and during the Masters degree.

Finally, I express my gratefulness to my family members for supporting me and believing in me. They kept me going on and this work would not have been possible without their support. Finally I would like to thank my closest friends who helped me in many ways and encouraged me to complete this research successfully.

ABSTRACT

Text classification is one of the core area of machine learning applications and it continuously improving with time. Hate speech classification in e-content became very popular over last few years and search engines, social media were the main interested parties of this area. “Racism content” is a sub part of the “Hate speech” area and many studies were carried out in this area. But surprisingly “Racism text classification in Sinhala language” is not very much popular. This study focus on building a machine learning pipeline for detecting racism comments written in Sinhala language. Sinhala racism text classification techniques can be used to remove unnecessary texts from social media, pages, blogs and many more text sources. This study was done to provide a solution to Sinhala racist text classification problems in social media, but also this study is valid for any text source that contain Sinhala text as Unicode text.

As the initial step, previous similar studies were reviewed and documented. Used techniques and results of similar studies were documented and reviewed. One similar study was selected as the baseline and set the baseline performance measures as the lower margin of the performance target. An architecture for the pipeline was designed and a methodology was selected as the next step. In this methodology, as the initial step, dataset was extracted and preprocessed. Stemming, stop word removal and conversion of text to basic character word were the main preprocessing steps. Features were extracted next and due to less number of racism data, an oversampling method was used to increase the training data. Six machine learning classifiers were selected and those were Random forest, Naïve bayes, SVM, Logistic regression, Ada boost and XGBoost. All the classifiers were trained with oversampled data. This was a major improvement point of the results. In order to increase the performance of these classifiers further, hyperparameter tuning was performed. Ensemble techniques were also used to increase the performance. As the ensemble techniques bagging, boosting and voting was used. After selecting the best classifiers from bagging and boosting, best three classifiers were set as the input to voting classifier to get the final results.

The study shows that each preprocessing steps improves the performance of classifiers and results. Each classifier behave differently in each step. This study highlights these differences and sensitivity of each classifier to various changes. In oversampling step and hyperparameter tuning step, all the classifiers reached to a stable level. Hyperparameter tuning identified as a critical step in Sinhala text classification. Finally the best results were shown by the voting classifier and selected as the best model in the study. The proposed pipeline performed better and highlighted the each classifiers performance differences. The pipeline was able to outperform the baseline with a greater accuracy. The study proves that the racism text classification is possible with selected classifiers and accurate data. Study also proves that a better Sinhala racism text classification pipeline can be built using other classifiers with the help of ensemble techniques and enhanced preprocessing techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
BoW	Bag of Words
CNN	Convolutional Neural Network
IDF	Inverse Document Frequency
PoS	Part of Speech
ROC	Receiver Operating Characteristic
SATC	Semi-Automated Text Classifier
SMOTE	Synthetic Minority Oversampling Technique
TF	Term Frequency
UTF	Unicode Transformation Format
WWW	World Wide Web

LIST OF APPENDICES

CHAPTER 1 - INTRODUCTION

This chapter intends to provide a brief introduction to this study. Text classification and hate speech in Sinhala language, problem identification and motivation for this study and objectives which intends to achieve throughout the research will be discussed.

1.1 Text Classification

Text classification is one of the most popular classification category due to rapid growth of e-content creation. Text classification provide a better way of classifying contents based on some rules and divide the results into groups called “classes”. Text classification is involving in many areas such as sensitive information classification, grammar checking and document searching in search engines. Starting from this broad area, this study narrows down its focus on to racism content classification for Sinhala language. Sinhala text classification area is not much improved compared to English text classification area and since population that use Sinhala language is relatively much lower than English language, the generated e-content is very low.

1.2 Sinhala language

Sinhala is the native language of Sri Lanka. It is evolved from the Indo-Aryan branch of the Indo-European languages. Sinhala has a rich written alphabet, which contain 54 basic characters. Sinhala language has many ways of implying a meaning, therefore speaking and writing in Sinhala is kind of difficult compared to many other languages. Sinhala text content is rarely available on internet last few years but now Sinhala has some rich contents available on the internet and even Google and Facebook allow user to choose Sinhala as preferred language. In last few years Unicode Transformation Format (UTF) characters were emerged to content management and it was a major breakpoint to Sinhala text publishing in internet. After this emerge, people started to post Sinhala contents in the internet more often; even in the social media, this became so much popular.

1.3 Text Classification in Sinhala

There are not many text classifiers built for Sinhala language compared to English language but the quantity is not very much low. With the Unicode characters, Sinhala text classifiers were built rapidly and this was a very good opportunity for researchers. As mentioned above section, number of users who comment in Sinhala in social media has increased and social media became a rich place to collect text contents. Hence social media based studies are still improving and most of them are focuses on text classification.

1.4 Hateful content in social media

It is a common situation that people post religious conflicting and racism contents in social media. These contents can be fraud and it can carry a wrong message to the society. The worst part in this situation is that people share this content in social media without judging the accuracy of the post, and by sharing this content, it spreads across the social media at an exponential growth rate and reach many users without knowing the original generated source of the content. If there is a way to identify racism content from these posts, then it will be a help to the social media and to the people who use social media since social media is the most powerful communication method in the world at the moment.

1.5 Problem Identification and proposed solution

As discussed in the previous sections, this study mainly focus on racism text classification in Sinhala language. Even though the basic target area for this study is social media, this study can be still applicable to any Sinhala text source. Sri Lanka faced a problem in 2018 and it was a civil religious conflict among Muslim and Sinhala people.

The reason for this conflict growth is that fraud information was posted in social media and that caused more problems. Government had to ban the social media for a week to control this situation. If there was a proper way of identifying racism contents in social media at that time; then there would be a way to prevent spreading wrong information without banning social media.

Even though the Sinhala language is now generating more content in social media, there are many challenges in text collecting, especially the comments in social media. Most of these comments are really long and many words are misspelled. These misspelled texts can be a result of using text converters. Sometimes user expected character might not be there in the character set and user can use an alternative character which can make the same sound when reading. Some comments are ambiguous and reader can be confused while reading. Such comments are barely point something clearly. Since Sinhala is a rich language of words and characters there are various ways of writing a content. Some contents are complex while another gives a deep meaning from few simple words.

Some people write comments as songs and these songs also can have a deep meaning since song writing is not same as the normal text writing in Sinhala. With these challenges text classification in Sinhala language become much difficult. As a solution for these challenges and problems, this research propose a machine learning approach to identify the Sinhala racism texts from normal texts which can be used to remove the racism texts from any text source. Further, this solution can be effectively used to reduce the racism text posts in social media and reduce misunderstandings and conflicts among people.

1.6 Goals and objectives

This research focus on building a machine learning pipeline for Sinhala racism text classification. In order to provide a solution for the research question, following goals and objectives were selected as the main targets to be covered in this research.

1. Identify most suitable classification techniques which can be used to classify Sinhala racism texts.
2. Identify the actions which need to be performed on the pipeline and which are not.
3. Observe each classifier behavior and response at each phase of the pipeline and identify which techniques are most suitable for building the optimum classification pipeline by evaluating them.

4. Optimize the output of each phase of the pipeline by performing various actions and gain the best performance of the pipeline.
5. Produce the most accurate pipeline in classifying the Sinhala racism texts by using the best classifiers and improvement techniques.

CHAPTER 2 – LITERATURE REVIEW AND BACKGROUND

Literature review and background chapter discuss the similar studies which were carried out earlier by others, their difficulties and achievements and limitations and improvements which can be help this study to achieve the desired objectives. This chapter also discuss the background of some concepts which are useful for this study.

2.1 Similar studies

There are various studies were done in cyber hate speech recognition. This is a huge section and has many sub modules. Racism content classification is one of them. When considering cyber hate speech on social media, [1] Pete Burnap et al., 2015 did a study for twitter. Authors' main target was to build a classifier to classify hate speech on tweets. Tweets were extracted and those were annotated from "Crowdflower", which gives the capability to annotate data by people with their majority votes. As final data set 1901 tweets and 222 of them are offensive.

Data set was preprocessed as the next step and they have implemented the Stanford lexical parsing model which is capable of extracting typed dependencies in tweets and that also provided grammatical relationship representations in a text. All the tweets were transformed into word vectors but they have not used the BoW (Bag of Words) approach since it assigns equal weights to each n-gram and can be led to confusing results. Random Forest Decision tree approach was selected to do the classification. This classification technique combines the output of several decision trees to decide the best rule set. They have implemented a Bayesian Logistic Regression classifier as a probabilistic approach also used SVM to assess a special classification model which is based on probabilistic rules.

In high-dimensional space, feature vectors were plotted and divided into classes. They have used an ensemble voting classifier which is capable of making the final output by considering all models outputs. Results were taken as precision, recall and F-measure.

After analyzing results they have concluded that the n-gram hateful terms combined with n-gram typed dependencies approach is the best.

[2] Njagi Dennis et al., 2015; emerged into a rule based approach to detect hate speech in sentence level sentimental analysis. They have begun with training subjective sentence detector with multi-perspective question answering corps and other sources and extracted subjective sentences. After extracting those features, they used the noun patterns based on semantic classes of race and hateful verbs along with bootstrap technique. Afterwards authors' have built and tested their classifier with their annotated corps. For this experiment they have crawled 100 blog postings with a list provided in the hate directory. Learning-based and rule-based approaches were used to learn the subjectivity classifier. Finally hate verb growing algorithm was compared with classic apriori algorithm based on the annotation classification and used evaluation measures such as F-score, precision and recall for evaluation.

[3] Giacomo Berardi et al., 2015; propose a Semi-Automated approach to classify sensitive information. Semi-Automated Text Classifier (SATC) rank automatically classified documents. SATC use a human annotator who inspects and validates these automatically classified documents and this process can be specified as the process of ranking a collection of automatically classified documents. If a documents in a top-ranked set is validates by human annotator accuracy should be maximized.

Researchers were implemented an approach called utility-theoretic in order to approach SATC. Utility Theoretic capable of ranking the automatically labelled documents. Authors' also pointed out some important factors about evaluation criteria. Gains from validating true positives and True negatives were different when analyzing results F1 score. F1 score pays equal importance to precision and to recall, but a sensitive document missing is higher risk than mistakenly classifying a non-sensitive document, thus sensitivity identification can be defined as a recall priority task, which means that recall is much important than precision in this and that kind of measure should be selected.

Precision focus on "How useful the search results are" while recall focus on "How complete the results are". This factor can be directly relate to this study as well. F2 is a standard choice for recall oriented tasks, therefore; In order to measure of classification accuracy, they have used F2 measure. As the learning algorithm - SVM-light implementation was used.

Their training data set was imbalanced, therefore they have oversampled their training data set by duplicating sensitive records till they reach same balanced training data set. After applying oversampling technique, SVM model improved its performance.

[4] Irene Kwok et al., 2013; experimented racist content identification in twitter and used Naïve Bayes classifier. They have built a balanced dataset of sample tweets for training. Racist tweets were selected from Twitter accounts and they have identified that since tweets contain much offensive words, 86% were labeled as racist. Therefore, when building their vocabulary, they were considered unigram features. 10-fold cross-validation method was used to evaluate the accuracy of the classification and achieved a 76% accuracy 24% error rate. They have mentioned that their study only focus on unigrams, thus text sentiments were not used. They suggest that it should be considered and to capture the relationship between words, classifier does not use bigrams and Any tweet containing these racial characteristics may be incorrectly labeled as racist but originally it is not, thereby reducing accuracy. They have concluded that in order to correctly classify racist tweets, Bow model is not enough.

[5] William Warner et al., 2012; did a study on World Wide Web hate Speech recognition. Most of the previous studies in literature review were focused on social media and this study has focused on much larger scope. When they consider hate speech on WWW, they had to collect data from various places. A data set was provided by Yahoo! which was from their news group articles that had been considered offensive by users. They have annotated manually. They have identified seven categories to label and those were anti-immigrant, anti-asian, anti-muslim, anti-black, anti-semitic, anti-woman, or other kind of hate.

As the classification approach they have used the template-based strategy to create features using the corpus. All templates had a single word centered on and they also used PoS tagging. Their classification task is weight prone and they have removed some features and some weighted with different values to reduce the weight unbalance of feature space.

They have used SVMlight and kernel which used was linear. All classifiers were cross validated with 10-fold cross validation. As final results they have observed that full set was outperformed by unigram feature set, and the smaller set of features consisting only positive unigrams performed best. And they have also mentioned that bigram and trigram templates degraded the performance of the classifier.

Björn Gambäck and Utpal Kumar Sikdar [7] classified hate speech using neural networks. They have built a Convolutional Neural Network (CNN) model to hate speech categorization and have used word2vec which is build word vectors based on semantic information for tokens. Features were extracted and merged with the word vectors and downsized with the help of max-pooling. CNN model was fed with character n-grams and above features to categorize tweets using predictions. Their dataset includes 6555 tweets and 10-fold cross validation technique was used to validate the dataset and compared with the Logistic Regression model.

Finally, with the values of precision 85.66%, recall 72.14% and F-score 78.29%, word2vec model without character n-grams obtained the optimal results. By having high precision and F1-score, logistic regression model was outperformed by the CNN model. Logistic regression model outperformed all the models from recall.

[8] Dulan S. Dias et al., 2018; show that Sinhala racist comments can be classified using n-grams. They built a corpus by randomly capturing Sinhala language comments from social media and annotated manually as “racist” or “non-racist”. They built an N-gram dictionary from comments and as weighting function they have used TF-IDF. SVM was used as the classifier and 75% of random data was selected from dataset and model was trained with that data. 25% of data used as the test data. The Microsoft Azure Machine Learning Studio was used to tune the hyperparameters. They have mention that Hyperparameters tuning improved the accuracy.

Precision, accuracy, recall, F1 score were selected as the evaluation criteria and ROC and Precision vs. Recall graphs also considered to check the quality of the output. They have mention that they used 100 Racist comments and 138 Non-Racist comments.

Even though this dataset is balanced, size of the dataset is not sufficient to build a good model. Since this is the only study which was done to classify Sinhala racism texts, it was selected as the baseline.

When it comes to Sinhala NLP studies with word embeddings, [18] Dimuthu et al., 2020; did a study of evaluation of different types of word embeddings for Sinhala language. They have evaluated three word embedding models, Word2Vec, FastText, and Glove under two types of evaluation methods and those are intrinsic evaluation and extrinsic evaluation. In intrinsic evaluation, word relatedness evaluations were performed. As extrinsic evaluation tasks, sentiment analysis and POS tagging were conducted.Their final pre-processed corpus contained of 94,648,911 tokens. FastText model was trained based on the Skip-gram architecture and both skip-gram and CBOW models of Word2Vec were evaluated. As per their final results FastText word embeddings with 300 dimensions gave the best results. They also described that there is no perfect word embeddings technique for all NLP tasks and that can be varied according to nature of the task.

Z. Waseem and D. Hovy [9], have done a study based on predictive features for hate speech. They have collected the data source as tweets and collected 16,914 tweets, which were annotated as sexist, racist or neither of them. By using Logistic regression classifier, they checked the effect of different features on prediction performance and cross validation also done with ten folds. As researchers have found character n-gram is better than word n-gram. They mainly used the location, gender and tweet length as extra features. With the additional feature as gender, best performance as F-score 73.93% was achieved with character n-grams of 4. They stated that other features like location and length have not contributed to F1-score improvements. They also concluded that problem can be partially solved using a character n-gram based approach.

A dictionary based approach has used to detect racism in Dutch social media by St´ephan Tulkens et al [10]. They collected comments from two social media sites and annotated them and used SVM classifier to automatically classify comments, using handcrafted dictionaries. Their Annotated dataset had 5759 comments.

This dataset was had three classes of data, which were invalid, non-racist and racist. They used their own annotations as gold standard to evaluate performance. Manually cleaned dictionary was used by the best model and gained 0.46 as F-score.

When it comes to low resource language text classification studies, Md Gulzar Hussain et al. [11], did a study to detect abusive Bangla texts. They have collected 300 comments from social media and used their own classification algorithm. As features they have used unigram, bigram and trigram. They claims that two words in a sentence with their relevant relationship can be identified in the bigram structures. Finally they were able to achieve precision of hate class as 0.83 and recall as 0.67. They stated that when it comes to the Bangla language, differentiating among funny, hate and abusive speech was a challenge.

Indonesian Language hate speech detection was done by Ika Alfina et al [12]. They used Twitter streaming API to collect tweets and had 1,100 tweets to be labeled manually. Researchers found 30 volunteers who were all 17-24 year-old college students and toughly 43 percent of them were men and 57 percent were women; also they belong to many races and religions.

This annotation is bit different since researchers trying to reduce the annotator's bias. Their dataset was unbalanced and performed undersampling to balance it. After undersampling they had 520 tweets with 260 for hate and 260 for non-hate classes.

They used BoW and negative sentiment, character n-gram and word n-gram were used as feature classes. Word n-grams were used up to 4 (Quadra-grams). They have used Random forest decision tree, Naïve bayes, Bayesian logistic regression, SVM, and classifiers for classification. The best 82.6 percent F-measure was obtained by merging the RFDT and character Quadra-gram. Character trigram and RFDT as well as the combination of Unigram and BLR or RFD combinations, also performed well. They claims that these four variations are pretty much comparable.

Thomas Davidson et al. [13], did a study to detect hate speech in twitter and problem detection on offensive language using machine learning approach. The tweets are categorized in to classes, which are offensive, hate, or neither.

First, using the Twitter API they extracted 25k tweets containing terms from the lexicon and annotated by CrowdFlower (CF) workers. They have converted all tweets to lower cases and Porter stemmer was used to stem the texts and generated up to trigram features and used TF-IDF weighting.

They also added features for the words, character count in all tweets. Logistic regression with L1 regularization was used to decrease the dimensionality of the data. They have used other models as well, such as linear SVMs, random forests, decision trees, and naive Bayes. 5-fold cross validation was used to evaluate each model.

Grid-search was used and they observed that the SVM and Logistic Regression performs better than the other models. All classifiers have been trained for all labels, and each tweet is assigned a class label with the highest expected probability across all classifiers.

Result values of 0.91as precision, 0.90 as recall, and 0.90 as F1 score were achieved by best model. They mentioned that lexical methods are capable of identifying potentially offensive words but sometimes can be wrong when identifying hate speech; researchers have found that some words play major role when it comes to identifying between offensive language and hate speech.

N. Ruwandika [14] detected social media hate speech using lexicon based method. 1500 annotated comments were used as dataset and as the hate lexicon google bad word list was used. He has count of the hate words in the comment and used that count as a feature. Four classifiers and a clustering technique were used for his study.

As unsupervised learning model KMeans clustering algorithm was used and as supervised learning models Logistic Regression, Naïve Bayes, Decision tree and SVM were used. He mentioned that performance of the unsupervised learning model was significantly lower than the supervised learning models. As the best performance measure, 0.719 of F-score value was gained by Naïve Bayes Model with Tf-idf feature vectors and worst results were achieved by KMeans clustering model.

Joni Salminen et al. [15], developed a hate classifier and considered several social media platforms as their data sources. They first collected 197,566 comments from Wikipedia, Twitter, Reddit, and YouTube. They labelled 20 percent of dataset as hateful 80 percent as not-hateful. As classification algorithms Neural Networks, XGBoost, SVM, Naïve Bayes and Logistic Regression were used and as features 'BERT', 'Word2Vec', 'BoW', 'TF-IDF' and their combinations were also used. They mentioned that keyword-based baseline classifier was outperformed by all classifiers and XGBoost performed the best as F1 = 0.92. They have analyzed the importance of features and identified most impactful features for the predictions as BERT features.

Punyajoy Saha et al. [16], combined different types of features and classified hate speech against women. They have used Sentence embeddings, TF-IDF vector and Bag of words vector (BoWV) as feature vectors and Logistic Regression, XGBoost and CatBoost as the classifiers. Their training dataset had of 4000 labelled tweets and 1000 unlabeled tweets as test dataset. Dataset labels are Misogyny, Misogynistic category and Target. They have generated three types of features and concatenate them for each tweet. Researchers claimed that the combination of all the features worked the best and obtained the best result for English subtask (0.704 accuracy) from Logistic Regression classifier. Catboost and XGBoost performed next. They concluded that multiple feature vectors can be concatenated and get improved results.

Fabio Del Vigna et al. [17], also did a study on hate speech detection on Facebook using Recurrent Neural Networks and SVM classifier. Their dataset contained 17,567 Facebook comments annotated them to labels: No hate, Weak hate and Strong hate. They tested two different classifiers: RNN named Long Short Term Memory (LSTM) and SVM. LSTM are capable of capture long-term dependencies in a sentence. Lexical information in a short text can be very sparse. To overcome the issue they created two Word Embedding lexicons and trained two predict models using word2vec. As the results they have achieved best results with SVM classifier. SVM achieved Accuracy: 64.61% and F-score: 0.256 for strong hate class.

When it comes to deep learning techniques used in hate speech classification domain, Satyajit Kamble and Aditya Joshi [49] have done a study to detect hate speech from hindi-english code mixed tweets using deep learning models.

This particular study proposes three deep learning models for hate speech detection. CNN-1D, LSTM and BiLSTM. They have trained word embeddings on a large corpus of code-mixed data. Also they have created domain specific word embeddings. Finally they have compared the results of each model and they claimed CNN-1D performed better than the other models.

Miguel Romero et al. [50], evaluated neural network techniques with small datasets. In this study they have applied transfer learning and they claimed that it played an important role by reusing previously learned features. Additionally, the performance of deep CNN under different training and transfer learning methods were evaluated on small training sizes. They concluded that transfer learning play a major role when it comes to small datasets. They also claimed that if dataset is small it should be trained well and data source also should be rich from all aspects.

Prashant et al. [51], also done a study on hate speech detection in social media using deep learning approaches. They developed thirteen deep learning models and used word embedding techniques such as W2V, glove and fastText. They achieved good results with character n-grams. They also described that identification of contextual hate speech was challenging. Steven et al. [52], done some improvements for hate speech detection with deep learning ensembles. They have shown that weight initialization methods are an important factor to consider in any research using deep learning. Also they shown that ensemble method for neural network is much effective and has a significant improvement over a single model.

When reviewing the hate speech detection using neural networks, the common problem they discussed was the dataset size. Almost all the studies thoroughly mentioned that dataset should be bigger and rich. Also dataset should contain clean data. Therefore in order to get a quality output from neural networks, dataset size is important. Comparing the previous studies done with the neural networks, their datasets and difficulties, it was clear that if a study uses a neural networks, the dataset size has to be relatively higher than traditional approaches. Therefore as a conclusion applying a neural network technique with low resource languages are challenging based on the literature.

2.2 Results comparison

All the similar studies which were observed in previous section can be compared to each other. This comparison gives a better abstraction of each study and makes it easier to compare against other studies. Comparison of the similar studies which were discussed in earlier section have been shown in Appendix I, Appendix II and Appendix III.

2.3 Background

This section describes the theoretical background of some concepts which have been used in this study.

2.3.1 Oversampling

Oversampling is a statistical technique which is used to adjust the class distribution of a data set. Oversampling normally used when class imbalance problem present in the dataset. There are various oversampling techniques and SMOTE is one of them.

2.3.1.1 What is SMOTE

Synthetic Minority Oversampling Technique, is known as SMOTE for short. This statistical technique is normally use to balance the data set. SMOTE basically generate new instances from minority instances until data set is somewhat balanced.

2.3.1.2 How does SMOTE works

SMOTE creates new cases, but new cases are not only copies of current minority instances; instead, the algorithm takes samples of the feature space for each target class and its nearest neighbors, and generates new examples that combine features of the target case with features of its neighbors. This approach increases the features available to each class and makes the samples more general.

Figure 2.1 shows an example of data point distribution in feature space. After applying SMOTE oversampling, new data points can be anywhere between those lines, and those are not copies, those are new instances. Figure 2.2 shows these new instances. SMOTE initially selects an instance from minority class randomly and finds k nearest minority class neighbors for that instance.

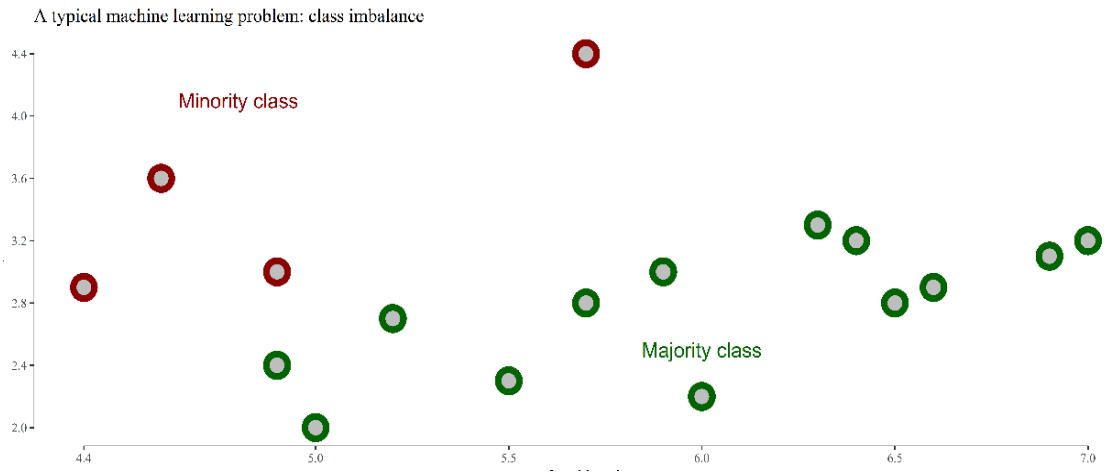


Figure 2.1: Class imbalance problem

Source: [41]

By choosing one of the k nearest neighbors, artificial instance is created randomly. Afterwards a line segment is formed in feature space using those two instances. The artificial instances are generated as a convex combination of those two chosen instances. [42]

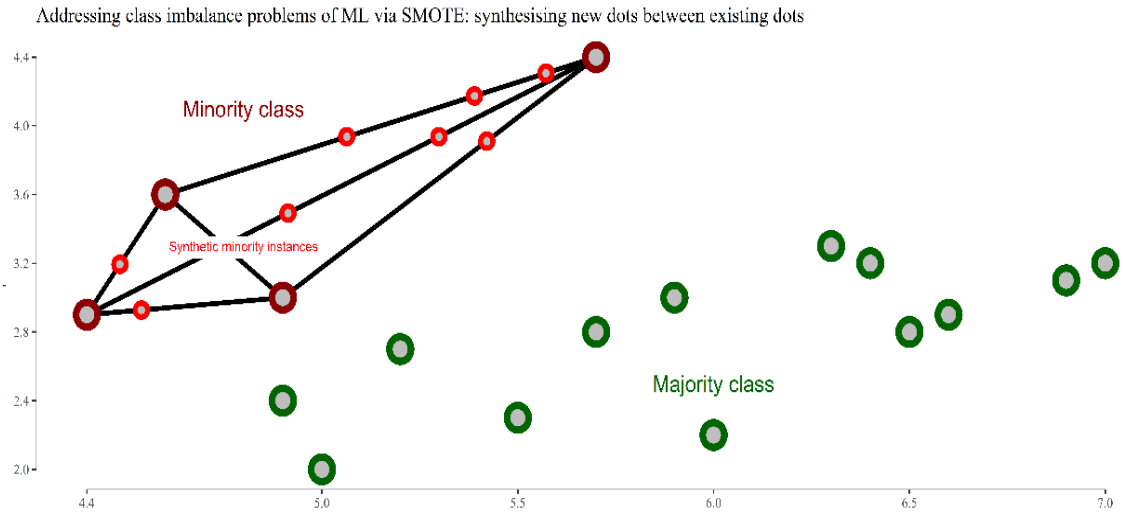


Figure 2.2: SMOTE new generated instances

2.3.2 Classifiers

Classification is basically an algorithm that maps input data to a category or class. There are many classifiers in machine learning and few of them have been described below.

2.3.2.1 Random Forest Classifier

Random forest is a sub instance of ensemble techniques. This classifier uses decision trees as estimators or weak learners and combine all the outputs from base estimators and majority output was selected by voting and consider that as the final output.

2.3.2.2 SVM Classifier

Support-Vector Machines (SVM) is a popular classifier for any kind of classification problem. Basically SVM constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space. Hyperplanes are decision boundaries that help to classify the data points. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. This hyperplane should separate the data points correctly and after that it can be used for classification. Hyperplane in feature space has been shown in Figure 2.3.

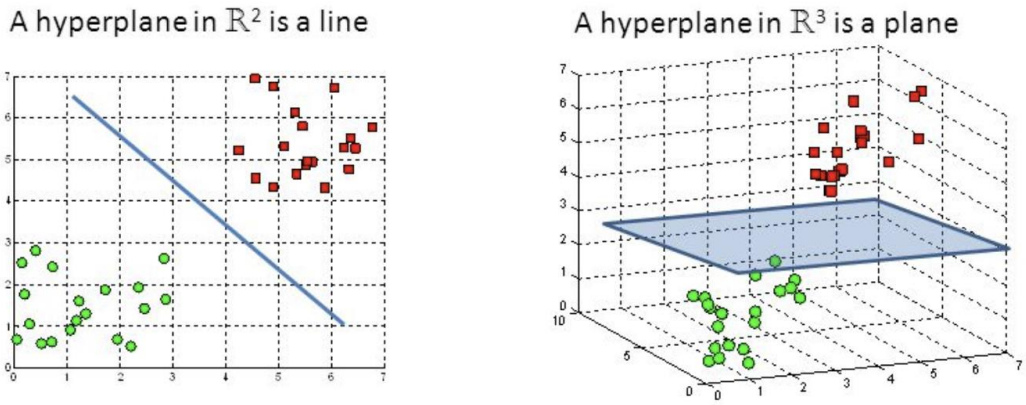


Figure 2.3: Hyperplanes in SVM

Source: [44]

Some important parameters of the svm classifer are

- C: controls the tradeoff between smooth decision boundary and classifying training points correctly.
- gamma: this parameter determines how far the influence of a single training example reaches, reach is ‘far’ when gamma is low and ‘close’ when gamma value is high. [46]
- class_weight: some problems have different class priorities and some classes are more important than other classes, this parameter can be used in such scenarios; which means this should considered when SVM use for class imbalance problems. Importance of weighted class is shown in figure 2.4.

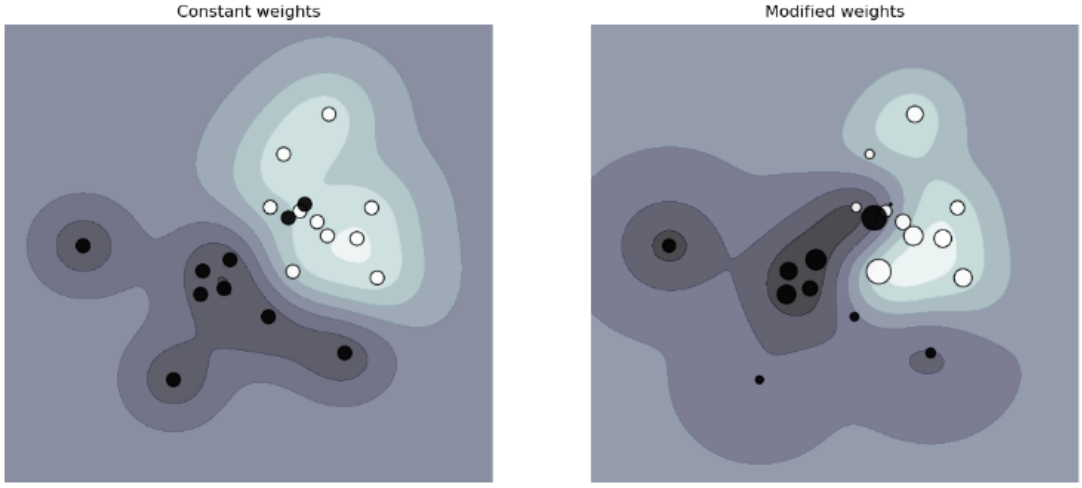


Figure 2.4: Importance of weighted class

Source: [47]

2.3.2.3 Naïve Bayes Classifier

Naïve Bayes is a commonly-used, simple, yet effective machine learning classifier. It is a probabilistic classifier that makes classifications based on the Bayes theorem.

Bayes theorem finds the probability of an event occurring given the probability of another event that has already occurred.

Bayes theorem is stated mathematically as the following equation:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Where, y is class variable and X is a dependent feature vector (of size n) where

$$X = (x_1, x_2, x_3, ..., x_n)$$

Hence, we reach to the result:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Which can be expressed as:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, as the denominator remains constant for a given input, that term can be removed:

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

The probability of given set of inputs for all possible values of the class variable y and pick up the output with maximum probability can be expressed mathematically as:

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

P(y) is also called class probability and P(xi | y) is called conditional probability.
These results can be used to calculate conditional probabilities and naïve bayes classifier mainly based on this theorem.

Multinomial Naive Bayes classifier is a specific instance of a Naive Bayes classifier which uses a multinomial distribution for each of the features.

2.3.2.4 Logistic Regression Classifier

Logistic Regression is a popular ML algorithm and it is a predictive analysis algorithm which can be used for the classification problems.

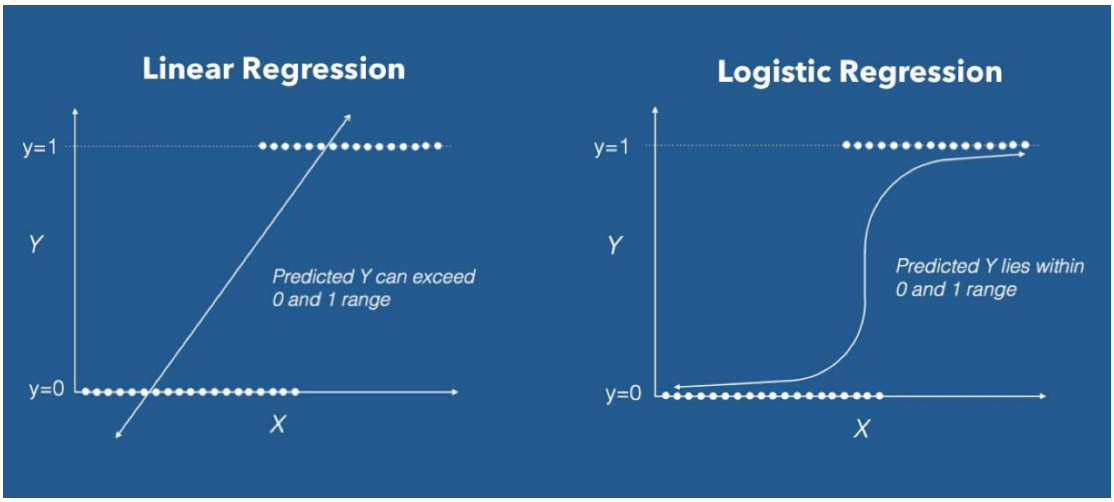


Figure 2.5: Logistic regression function

Source: [48]

This classifier use a cost function called 'Sigmoid function'. This cost function can be defined as the. This function capable of mapping any real value to another value between 0.0 and 1.0, in other words function can use to map predictions to probabilities. Figure 2.5 shows the logistic regression sigmoid function. Like SVM, "C and class_weight" parameters are present in this classifier as well.

2.3.3 Bagging

Ensemble learning is a machine learning paradigm where multiple models are trained to solve the same problem and combined to get better results. There are various techniques under ensemble concept, bagging is one of them.

2.3.3.1 What is bagging

Homogeneous weak learners, learn independently from each other in parallel and combines their results to get the final result. This is known as bagging. Bagging constructs n classification trees using bootstrap sampling of the training data and then combines their predictions to produce a final meta-prediction. [20].

In other words, Bagging fits several independent models and average their predictions in order to obtain a model with a lower variance. However, in practice, fitting fully independent models is difficult since it require too much data. Therefore, it rely on the bootstrap samples to fit models that are almost independent.

After getting the results of the model, aggregate them into some kind of averaging process in order to get an ensemble model with a lower variance.

Bagging process has been shown in Figure 2.6.

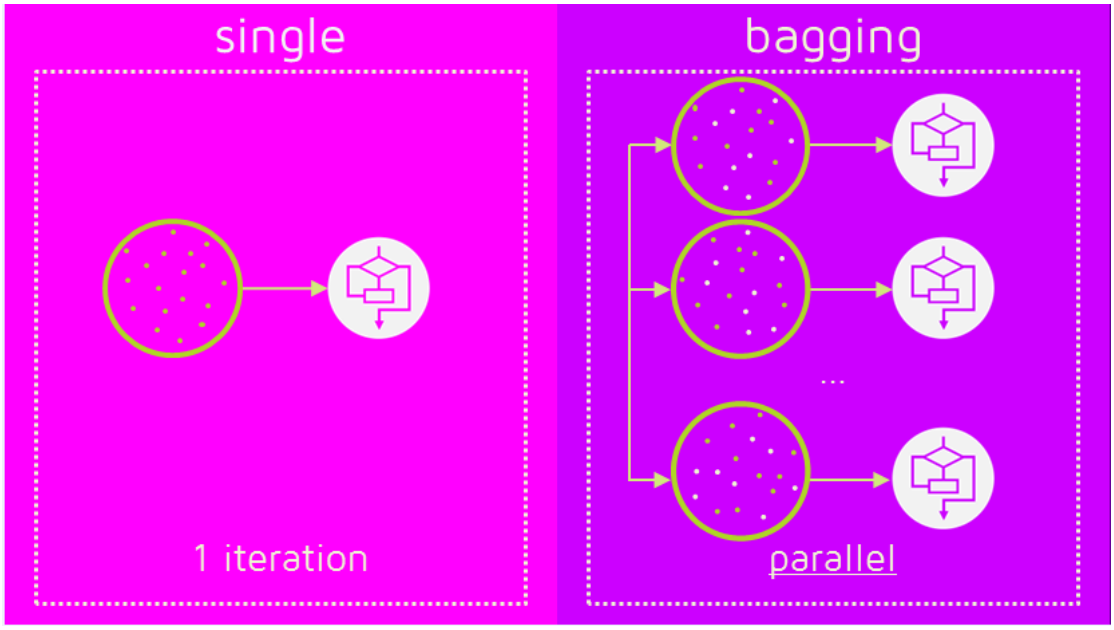


Figure 2.6: Bagging process

Source: [21]

2.3.4 Boosting

Boosting is one another ensemble technique and widely used in machine learning. Boosting, technique consider homogeneous weak learners, learns them sequentially in a very adaptive way where the base model depends on the previous ones, and combines them following a deterministic strategy. Boosting method work same as bagging methods, except few changes. Boosting also build a family of models that are aggregated to obtain a strong learner that performs better.

However, while bagging mainly aiming at reducing variance, boosting considers in fitting sequentially multiple weak learners in a very adaptive way. The special case here is each model in the sequence is fitted giving more importance to observations in the dataset that were badly handled by the previous models in the sequence. In this way, each new model focus its efforts on the most difficult observations to fit.

At the end of the process, a strong learner with lower bias gives the results. [23]. Boosting process has been shown in Figure 2.7.

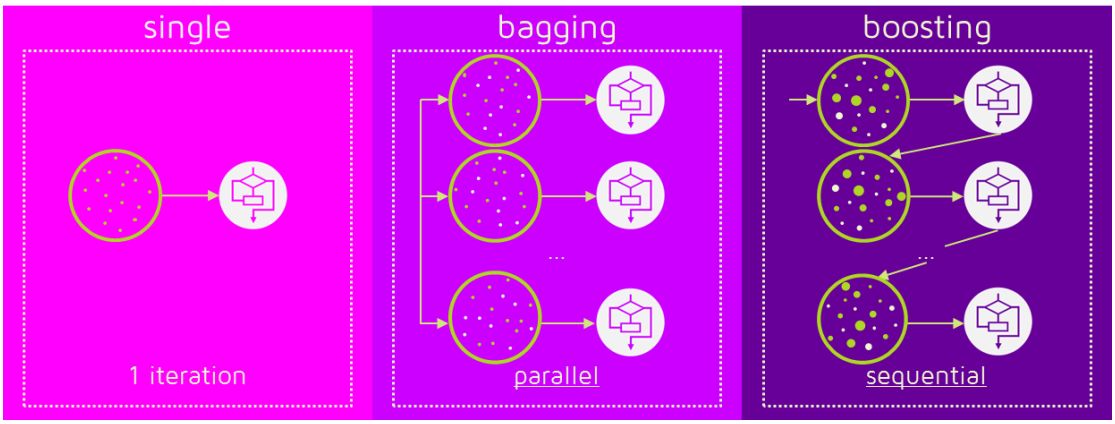


Figure 2.7: Boosting process

2.3.4.1 Ada Boost

AdaBoost, short for Adaptive Boosting, is a meta-algorithm in machine learning. AdaBoost use to boost the performance of any machine learning algorithm. To boost performance this technique uses many types of learning algorithms. This algorithm also use weak learners output and combined them with the weighted summing mechanism and produce the final output. This technique get the best from the weak learners and known as one of the best classifiers for binary classification.

3.3.4.2 How Ada Boost works

Ada boost works on a sequential way and first it trains weak learners with equal weighted training data and predict with original data. If the first learner's prediction is wrong, then it assigns higher weight to observation that was incorrectly predicted.

This way, it keep adding the learners until it reach the maximum learners count or maximum accuracy. To classify, it performs a "vote" across all of the learning algorithms which were built in the process. [24]. Decision trees with a single split in AdaBoost is known as decision stumps which are weak learners. Ada boost use this stumps. As the base estimator, this support any machine learning algorithm as long as it accepts weights on training data set.

2.3.4.3 XGBoost

eXtreme Gradient Boosting is known as xgboost. This boosting classifier is very powerful, scalable and efficient implementation of gradient boosting technique and optimized for boosted tree algorithms. The term “Gradient Boosting” is proposed by Friedman, 2001. [25]. Main goal of the xgboost is pushing machine computing limits extremely to provide scalable, compact, and reliable results for large-scale tree boosting. xgboost capable of supporting different functions such as ranking, classification and regression. xgboost is extendable for users requirement, which enable users to define and achieve their own objectives with a low effort.

Figure 2.7 shows how xgboost improve standard Gradient Boosting algorithm.

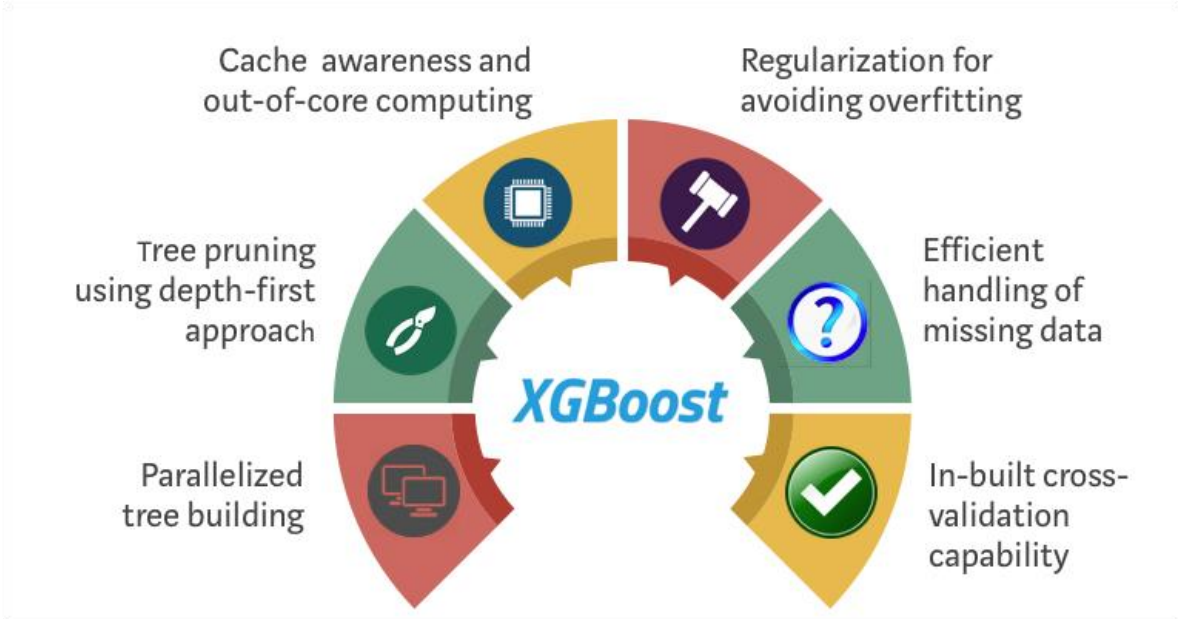


Figure 2.8: XGBoost improvement process

Source: [26]

2.3.5 Voting

Voting is another ensemble technique and widely use on machine learning. Voting classification concept will be described in following sections.

2.3.5.1 What is voting

As discussed earlier ensemble techniques use many estimators or learners and aggregate each results and get the final results by voting. Bagging, boosting use this method but they cannot have multiple types of base estimators. i.e.: Naïve Bayes and SVM. This is where the voting classifier emerges. Voting classifier has the capability to input many base estimators and get each base estimator's results and aggregate them to a final result.

2.3.5.1 Voting Classifier

Voting Classifier combine predictions of many machine learning algorithms. This classifier actually evaluate the votes and gives the final output according to votes. If multiple algorithms predict the same output for particular target, that majority vote becomes the final result.

This voting mechanism has two way of performing the voting.

1) **Hard Voting:** Hard voting can be described as the basic voting, In this case, the class that received the highest number of votes will be chosen.
2) **Soft Voting:** Soft voting consider the probability vector for each predicted class for all classifiers are summed up and averaged. The resulting class is the one corresponding to the highest value.

Figure 2.9 describes the voting classifier behavior.

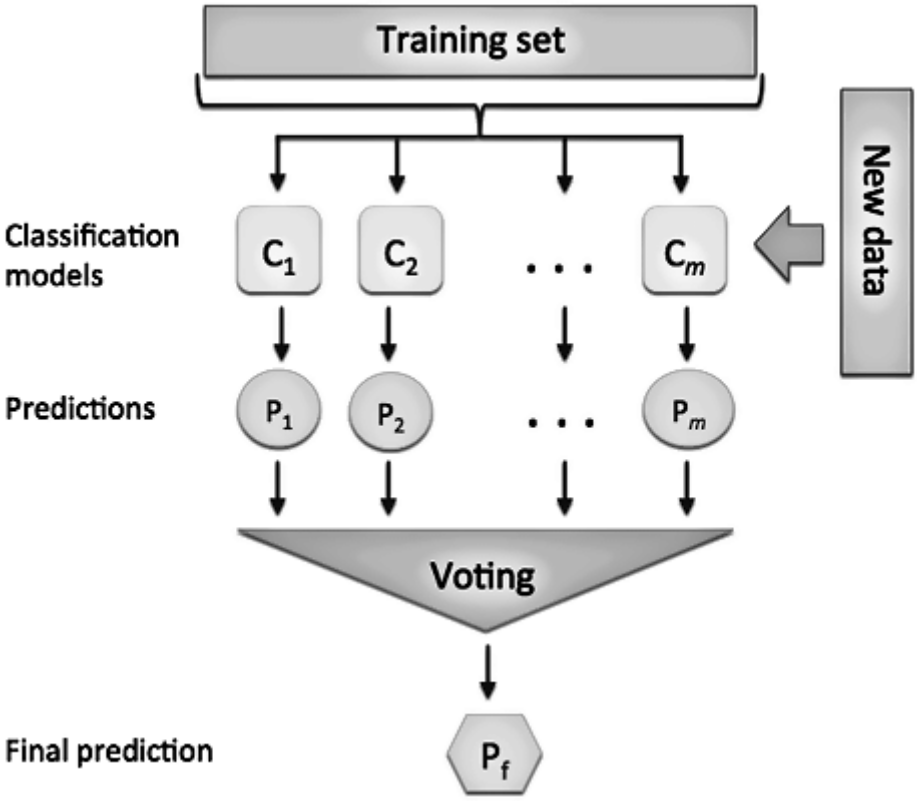


Figure 2.9: Voting classifier behavior

Source: [27]

CHAPTER 3 – METHODOLOGY AND EXPERIMENT

After reviewing similar studies, the methodology was selected. In order to select a better methodology, various actions and experiments were done and analyzed. This section describes the methodology which was used in this study and other experiments.

A pipeline for classifying racist comments written in Sinhala language, was designed and all the steps were performed with respect to this architecture. The designed pipeline has been shown in Figure 3.1.

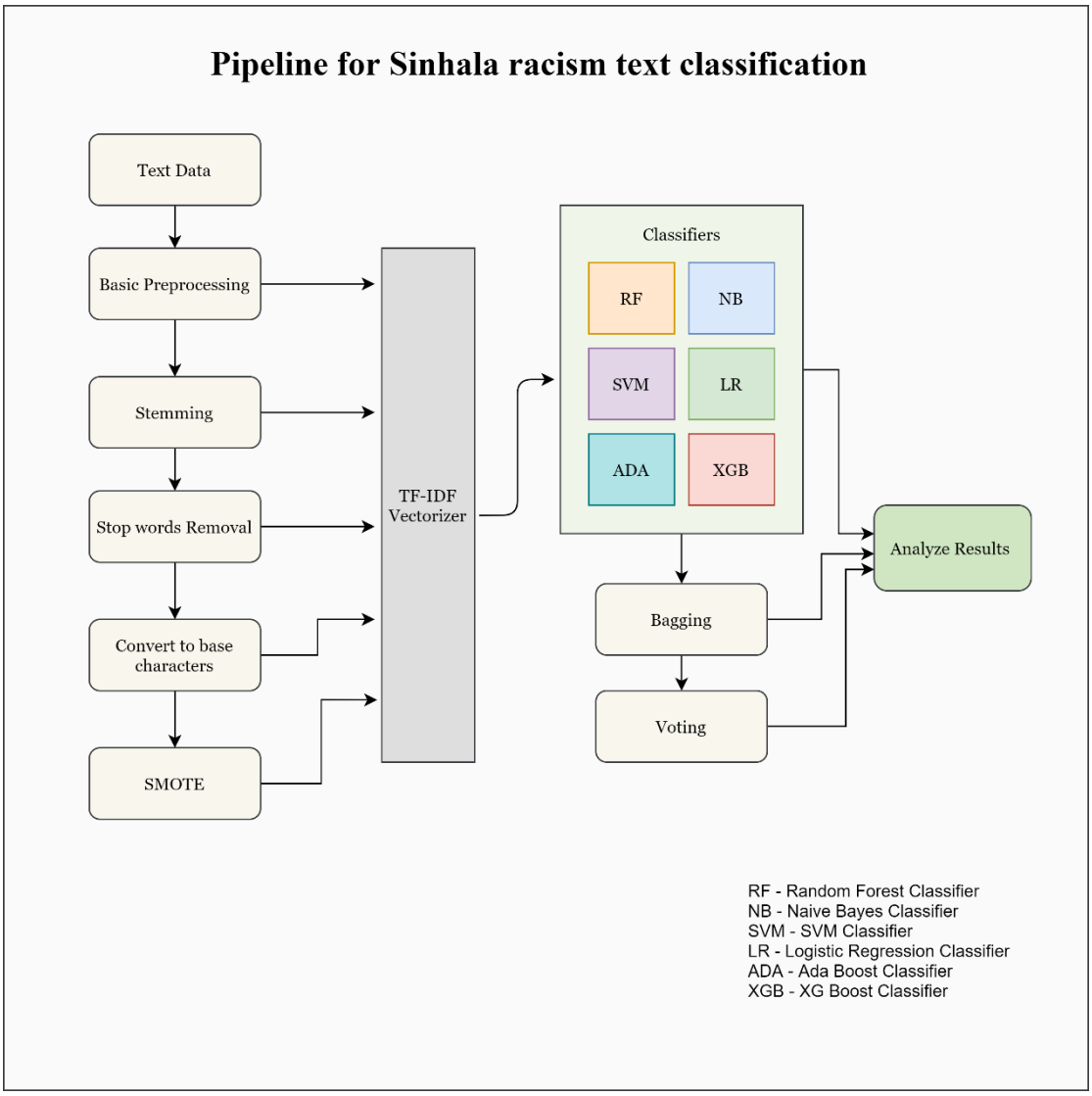


Figure 3.1: Pipeline for Sinhala racism text classification

As for the methodology, there were various steps to be completed in order to get the final results. Figure 3.2 describes the main flow of this experiment.

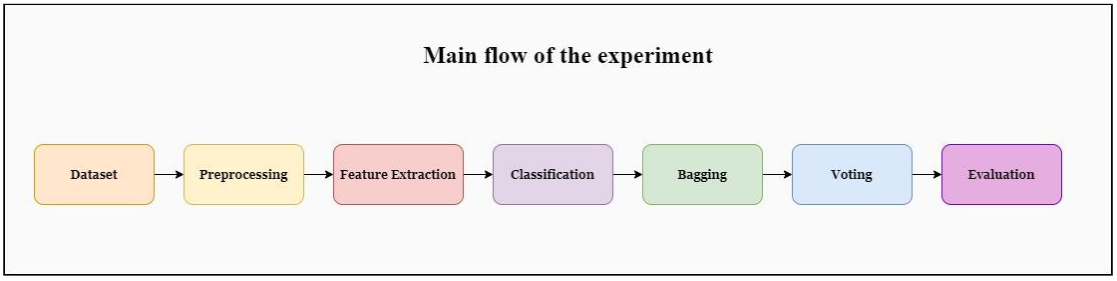


Figure 3.2: Main flow of the experiment

Following sub sections describes the details of each main step, sub tasks and challenges of each step of the experiment.

3.1 Dataset

Phase one is mainly focus on dataset. This section contains the actions or steps which were completed until data annotation finished.

3.1.1 Finding the Dataset

As the first step in proposed methodology, data was collected. For this, numerous data sources were checked and data was extracted. When extracting religious conflict and racism texts from the social media, Facebook comments, tweets from tweeter, racist articles on social media were considered first. Facebook has large active communities and pages which share racism and anti-religious posts. By subscribing these pages, latest data was collected.

Another dataset was collected from a Github. These posts and comments were extracted and stored for preprocessing. Figure 3.3 shows a sample post which was collected from Facebook.


එක් වන්න අප සමග : - සුර සරදියල්බලකාය http://www.facebook.com/surasaradiyalbalakaya
කල නිස්සේ විවිධ උපක්‍රම භාවිතා කරමින් අනවසරයෙන් පවත්වාගෙන ගිය පල්ලිය මෙවර හොර රහසේම විවෘත කිරීමට කට යුතු සංවිධානය කරන බව දැනගත් අතීත සිංහල පුතුන් කිහිප දෙනෙක් මෙයට විරුද්ධ වීම සඳහා සාක්ෂිය කරමින් සිටින විට එප ස්ථානයට ආයුධ අතැතිව පැමිණි මුස්ලිම් අන්තවාදීන් එක් තරුණයකුගේ බෙල්ලට කඩු පහරක් එල්ලකොට තිබේ ...ඔහුට බරපතල තුවාල ඇති හෙයින් මේ වනවිට ඔහුව රෝහල්ගතකොට තිබේ ....පැමිණි මුස්ලිම් අන්තවාදීන් එම ස්ථානයේ රැදිසිටි තවත් සිංහල තරුණයන් 4දෙනෙකුට අමානුෂික ලෙස පහර දී තිබේ ...මේ අවස්ථාව වනවිට සිංහල තරුණයන් 5දෙනෙක් රෝහල් ගතකොට තිබේ ...ලද සැනින් විස්තර ගෙන ඒමට අප සුදානම් එක්වී සිටින්න අපත් සමග ...

Figure 3.3: Sample hate post in Facebook

Source: [6]

There were some special cases to be considered when collecting the dataset. In other words there were many types of comments were observed in the database. Those have been listed below.

- Political hate speeches.
    - මැණික් මල්ල සහ යාලුවො ඉඩම් දිපු කතා හා සමානයි. ජනතාව මොඩයො
    - ඇමති හරි මොකද රට පාලනය කරන්නේ හරක් නේ..ඒකෙන්ම පැහැදිලි නේද ඇමතිලත් මෙවට සම්බන්දයි කියල.
- Sarcastic comments.
    - වෙන්න ඇති ඌ කියනවනේ.
    - උළමා බකමූණා කරන්න ඕනි.
    - හරි හරි බන් එල එල.
    - කඩු 59 කින් කපන්න තරම් කොළඹ කැලයක් වැවෙනකල් බලන් සිටි මහ නගර සභාව මේවට වග කියන්න ඕන.
- Normal comments.
    - යකෝ දඩේ ගෙවලා ලයිසන් එක ගන්නත් පැයක් විතර ඉන්න ඔනේ බං.
    - පිස්සුවෙන් දොඩවනව කියන්නෙ මෙන්න මේවට.

All these posts had to be converted to Unicode characters (UTF-8). More than four hundred hate speech posts about racism and religious conflicts, were chosen and also more than one thousand five hundred non hate speech posts were selected as the dataset.

3.1.2 Preprocessing the dataset

As the second step of the pipeline, dataset was preprocessed. This step was the most important and challenging step of this study. All the major preprocessing steps has been shown in Table 3.1.

Table 3.1: Preprocessing steps

Action	Before preprocessing	After preprocessing
Emojis were removed	කෑල්ල බලන්න මාතර බස් ස්ටෑන්ඩ් එකට ආවා ❤	කෑල්ල බලන්න මාතර බස් ස්ටෑන්ඩ් එකට ආවා
	අල්ලා කෑලේ මවන් නැතුව කඩු විතරක් මවලා ⁉⁉	අල්ලා කෑලේ මවන් නැතුව කඩු විතරක් මවලා
Spellings were corrected	හිත්වල තියන **තරහාෘ** පිට කරන්නේ ඒම	හිත්වල තියන **තරහා** පිට කරන්නේ ඒම
	අපේ **වුන්**	අපේ **උන්**
Unwanted mentions in comments were removed	@Nimal15700736 @cleanpoltics @NimaliWijesuri3 @ApiWenuwen @JMLPBandara @GayaniGunaseke2 @ParliamentLK @randheeraM @nilanp @astroanu @mihywun @Aqua097 @hima_flake @LankanJokes @Yawathashi @ManoGanesan @PodujanaParty මේ ගිවිසුම හරහා අපේ රටේ සේවා ටික සිංගප්පූරුවට පමණක් නොවෙයි පාවලා දීලා තිබෙන්නේ	මේ ගිවිසුම හරහා අපේ රටේ සේවා ටික සිංගප්පූරුවට පමණක් නොවෙයි පාවලා දීලා තිබෙන්නේ
Duplicate comments were deleted	බෞද්ධයො බුද්ධාගමේ චිරස්ථිතිය වෙනුවෙන් හම්බයො මරන්න, පල්ලි කඩන්න, හම්බකඩ ගිනිතියන්න බෞද්ධයො බුද්ධාගමේ චිරස්ථිතිය වෙනුවෙන් හම්බයො මරන්න, පල්ලි කඩන්න, හම්බකඩ ගිනිතියන්න	බෞද්ධයො බුද්ධාගමේ චිරස්ථිතිය වෙනුවෙන් හම්බයො මරන්න, පල්ලි කඩන්න, හම්බකඩ ගිනිතියන්න
Unwanted links were removed	එක පැත්තකින් බැලුවාම උඩතලවින්නේ තම්බි දණ ගැස්සු චරිතයක් තමා ලොහාන් කියන්නේ .. https://t.co/J1Jc6n6Z6v	එක පැත්තකින් බැලුවාම උඩතලවින්නේ තම්බි දණ ගැස්සු චරිතයක් තමා ලොහාන් කියන්නේ ..
Unnecessary full stops removed	හම්බයො මරන්න, පල්ලි කඩන්න.. ..	හම්බයො මරන්න, පල්ලි කඩන්න
Special characters removed	අඩෝ..!! #$#%#	අඩෝ

Some unwanted parts left from the original source was removed	පසුගිය දිනවල කොළඹ නගරය පුරා භීතිය පැතිරූ පප්පා කළ අමානුෂික ඝාතන එකින් එක වමරායි. රහස් තොරතුරු සියල්ල එළියට. **Read more**	පසුගිය දිනවල කොළඹ නගරය පුරා භීතිය පැතිරූ පප්පා කළ අමානුෂික ඝාතන එකින් එක වමරායි. රහස් තොරතුරු සියල්ල එළියට.
#tags were removed in the dataset.	#බෞද්ධ	බෞද්ධ

Apart from those steps, there were many steps and actions which were performed during cleaning the dataset. Some of them have been listed below.

- Unsupported characters were deleted since they do not imply anything in Sinhala language.

Rest of the preprocessing steps have been listed below.

3.1.2.1 Stop words

The common words which appear multiple times in a document are known as stop words. These words do not have direct correlation with the classification class. In this step, stop words were identified and removed from the dataset. There were 440 stop words to be removed. Some of the stop words have been listed below.

'ඉනිං','සහ','වෙනත්','මම','මං','මගේ','මට','මා','මන්','මගෙ','මචං','අපි', 'අපිව', 'අපිට','අපට', 'අපිම','අපේ', 'ඔබ','එක', 'ඔයා', 'ඔයාලා', 'ඔහේ', 'ඔහේලා', 'අරූ', 'අරුන්', 'මූ', 'මුන්', 'මුන්ගේ', 'උඹ','උන්', 'උන්ගේ', 'උගේ', 'එයා', 'ඇය', 'ඒකි','මේකී', 'ඒකිම', 'මෙච්චර', 'මෙතන', 'මෙන්', 'මෙන්න', 'මෙයා', 'මෙහෙ', 'මෙහෙම', 'මේක', 'මේකෙ', 'මේම','මේවා', 'කවුද', 'කවද්ද', 'කොහොමද', 'කොහොම', 'කොයි', 'කොයිඛ'

There were some challenges to be dealt with when removing stop words.

i.e: Some words have many meanings in Sinhala. While one meaning gives a stop word, other meaning gives a verb. These kind of words were not added to the stop word list.

Eg: ඉඳන්- meanings are “from” or “sitting”. This “from” meaning is a clearly a stop word. But not the word “staying”.

එක්ක- meanings are “with”, “accompany”.

3.1.2.2 Stemming

Basic stemmer was implemented to stem the text. This stemmer has to be more sophisticated to get accurate results since Sinhala language is very complex. Stemmer can help to reduce the complexity of many words. However, there were many scenarios where stemmer basically invert the meaning of the sentence.

i.e.: **රහත්** - the enlighten one, or monk who understood the ultimate truth or not having any bonds or wishes. If we stem **ත්**, the stemmed word is **රහ**. The meaning of stemmed word is "taste".

This is clearly gives the wrong idea of word and Sinhala language has many words like this. These are some of limitations and disadvantages of the bad stemmer, and there are alternative ways to improve the accuracy. Implementing a best stemmer was not considered in this study, but a basic stemmer was implemented. A Sinhala morphological parser was also considered as an alternative, but it was unable to produce a quality output. A lemmatizer was not implemented in this study.

Some stemming actions have been listed below with example words in Table 3.2.

Table 3.2: Stemming actions

Action	Before stemming	After stemming
Remove "ටත්"	කොහොමටත්, ඔයාටත්, කලාටත්	කොහොම, ඔයා, කලා
Remove last character "ගේ"	අම්මාගේ, ඔයාගේ, හාමුදුරුවන්ගේ	අම්මා, ඔයා, හාමුදුරුවන්
Remove last character "'ත්"'	ඔබත්, මමත්, ඔයාත්, ආවත්, කලත්	ඔබ, මම, ඔයා, ආව, කල
Remove "වත්"	කවුරුවත්, ඔයාවත්, කාටවත්	කවුරු, ඔයා, කාට
Remove last character "'ද"'	කමුද,අගමැතිද, කොහේද	කමු, අගමැති, කොහේ

3.1.2.3 Convert to base characters

Sinhala language has many letters and after combining with some characters or symbols, it gives a different meaning and pronunciation also changes. When these combined characters removed, only base characters remain in the word. The other scenario is that there are some characters which gives same sound when pronounce. Some of those characters are actually in same letter group, just like in English capital and simple letters. These characters also has a base character. This conversion feature or action help to reduce complexity of a word. A dictionary was used to implement this feature. Some items in the dictionary have been listed below.

"ණ": "න",
"ඳ": "ද",
"ඵ": "ප",
"භ": "බ",
"ඹ": "බ",
"ශ": "ෂ",
"ළ": "ල",
"ඈ": "ඇ",
"ඌ": "උ",
"ඬ": "ඩ",
"ඦ": "ඥ",
"ඕ": "ඔ",
"ඖ": "ඔ",

3.1.3 Annotate the dataset

The next main step of the study was the annotation of dataset. In order to do the data annotation in a correct manner three annotators were selected and asked them to annotate the dataset in following manner. Copy of dataset was given to three annotators and they were asked to group the data into two categories, and those were “Racist”, “Non racist”. After annotating, datasets were compared and at least if two annotators categorized a comment to a particular label, that label was assigned to that comment, in other words each comment was labelled according to majority votes. Table 3.3 shows inter rater agreement of the dataset.

Table 3.3: Inter-rater agreement

	Annotator 1	Annotator 2	Annotator 3
Racist	444	452	449
Non racist	1517	1509	1512

Cohen's Kappa Statistic score was calculated as below,

Cohen's Kappa measure (k) = $(P_o - P_e) / (1 - P_e)$

(k) = (0.9923 - 0.4708) / (1- 0.4708)

(k) = 0.9854

There were some special cases when annotating the dataset; those have been listed below.

- Sarcastic comments were observed; but annotation was hard since those comments do not really provoke racism in a direct way.
- Some comments are actually not racist when reader does not know the context. If reader or annotator know the context he would annotate that comment as a racist. These kind of comments were considered as "non-racist" by annotators.
- Some comment were typed with emojis. Emojis change the sentence meaning in a particular way. When removing emojis some original labels of comments were changed.

After annotating, there were 449 racist comments and 1512 non-racist comments.

3.1.4 Train and Test Data

After creating the corpus train data was separated. Corpus order was shuffled and 70% of data split as training data and remaining 30% of data considered as training set.

3.1.5 Baseline dataset

I was able to collect the dataset which was used for the study of Racism text classification done by Dulan S. Dias et al. This dataset size was 238 comments and it was already annotated.

3.1.6 Challenges of dataset selection and preprocessing

There were some challenges when completing this phase. Following list describe the challenges of this phase.

- Finding the dataset was very difficult. All the social media platforms have removed the racism contents; even though there are non-racism posts, all comments were deleted. Since the racism comment's main source is social media, this situation limited the data collection.
- Some comments were very long, sometimes comments contain songs; therefore understanding and annotation were somewhat difficult tasks.
- Some Sinhala comments were not typed in UTF-8 format; therefore those were converted manually to UTF-8.
- Implementing a good stemmer was a challenge since Sinhala contain many rich words and most of the words have many meanings; therefore stemming most of the time inverts the meaning of a sentence.
- Defining some stop words in Sinhala language done by manually and most of the words were matched with the English stop word list. Another stop word list was given by UCSC Language Technology Research Laboratory.

3.2 Feature Extraction

This section describes the feature extraction techniques which were used in this study.

3.2.1 Tokenizer

"Tokenizing basically divides the text into minimally relevant units, and it is also a required step which should be perform before processing" [39]. Tokenizer splits the text into sentences and then sentences into typographic tokens. In this study a normal tokenizer was used and the texts were split from spaces and full stops.

3.2.2 TF-IDF Vectorizer

TfidfVectorizer uses Term frequency – Inverse document frequency scores to generate a feature matrix, in other words it transforms text to feature vectors that can be used as input to an estimator. A sentence becomes a vector and numbers of the vector represents td-idf values.

In this study also TfidfVectorizer was used to generate feature matrix and implementation of "sklearn TfidfVectorizer" [40] was used. This Tfidfvectorizer gives some parameters to change the output. Necessary parameters were assigned with values and those are,

- Maximum features count (max_features): This parameter gives the capability to limit number of features generate from Tfidfvectorizer. I have set this to 5000.
- Minimum document frequency: This option allow the Tfidfvectorizer to omit words which have a document frequency less than the specified value when building the vocabulary. This parameter was set to 5, which means if any term does not appear 5 times in whole dataset, that word will be ignored.
- Maximum document frequency: This limits the maximum occurrences of a term while building the vocabulary. If a word appear too much in the dataset, it become a common word and become useless for the prediction; in other words it becomes a "Stop word". To address this issue maximum document frequency option is given by Tfidfvectorizer. I have set this to 0.7, which means if any word occurs more than 70% of all words, that word should be ignored.

3.2.3 N-grams

Feature vector can generated in many ways. TF-IDF and N-grams are the most popular ones. While TfidfVectorizer count TF-IDF scores for each term, n-grams count word combinations. This combinations can be adjacent words or letters of length n in text, which means if n equals two (bigrams) it check for 2 adjacent words with all combinations in a sentence. Only bigrams were used in this study.

3.2.3 fastText

FastText is an open-source library which provides the capability learn text representations and text classifiers created by Facebook. This library is for learning of word embeddings and text classification. The model allows users to create a supervised or unsupervised learning algorithm for obtaining vector representations for words. In this study fastText library used for create sentence vectors and feature vector was created from those.

3.2.4 Choose the best feature generation technique

TF-IDF, N-grams and fastText techniques were used for feature matrix generation. All the classifiers were trained with these feature vectors and results were observed. At this stage stemming, stop word removing or base character conversion was not performed on the dataset. Best feature vector which supports the classifiers to achieve overall best results, was selected as the feature generation technique for the next steps.

3.2.5 SMOTE Oversampling

When considering the dataset, there were 449 racist and 1512 non-racist comments. This clearly leads to a poor results since the dataset is highly imbalanced. To overcome this problem "SMOTE Oversampling" method was used.

After applying SMOTE implementation provided by "imblearn" [43], the train data set size was increased from 1372 to 2130. This implementation let the user to select oversample class as well. I have set this to "minority", which means it oversample only minority class and it does not oversample majority class. This was done since our majority class instances are sufficient for this study relative to minority class; hence minority class oversampled. Improvements of results will be discussed in a latter chapter.

3.2.6 Challenges of feature extraction

Major challenges faced during feature extraction phase have been listed below.

- It was unable to use default tokenizer which was given by "sklearn Tfidfvectorizer" since it was not performing well with Sinhala UTF-8 characters and sentences; Therefore new tokenizer had to be implemented.
- Selection of best feature generator was difficult since those had to be tested for results to check for the best feature generator and which approach is best for this study.
- Solutions had to be found to address the class imbalance problem.

3.3 Classification

One of the most important step of this study is classification. Since the training data is available and those are correctly annotated, it is possible to use supervised learning method. There are many classifiers which can be used to build a good model, but these studies (Pete Burnap et al., 2015, Giacomo Berardi et al., 2015, Irene Kwok et al., 2013, William Warner et al., 2012, Dulan S. Dias et al., 2018), proved that this kind of problems can be solve more accurately with following classifiers.

Those classifiers are SVM classifier, Naïve Bayes classifier, Random Forest classifier, and Logistic Regression classifier. Addition to that another two classifiers were selected as boosting classifiers. These boosting classifiers will be described in latter sections. All these 6 classifiers were selected as final classifiers and each of those trained with train data.

Following libraries were used as the implementations of each classifier.

- "sklearn RandomForestClassifier" implementation [33] was used as the Random Forest classifier.
- "sklearn svm" implementation [45] of SVM was used as the SVM classifier.
- "sklearn MultinomialNB" implementation [34] was used as the Naïve bayes classifier.
- "sklearn LogisticRegression" [19] implementation was used as the Logistic Regression classifier.

3.3.1 K-Fold cross validation

5-Fold cross validation technique was used to validate the model. Training dataset was used as the validation dataset and the model which gives the best results was selected as the best model for specific stage. Each best model was then trained on full training data set afterwards and tested with testing dataset in order to get the best results of each stage.

3.3.1 Hyperparameter tuning

Classifiers are not built to solve every classification problem in optimum way. Problems are different and their solutions and way of achieving the solutions should be different.

Handling different classification problems cannot be achieved in a single predefined classifier; therefore classifiers have various parameters and those parameters gives the capability to change the way of the classifier work. Some parameters can be changed. According to this values classification results might be changed. To get the best outcome or best result, classifier has to be tested with various parameter values. Testing these classifiers with various parameter values and getting the optimum result known as hyperparameter tuning.

All the four classifiers were hyperparameter tuned with some range of values. "sklearn GridSearchCV" [35] was used to select the best model. Grid search generates a model for hyperparameter combinations listed, and examines each model. In GridSearchCV, it is possible to set a scoring function. According to scoring criteria GridSearchCV find the best estimator (model). I have set the scoring function as "neg_mean_absolute_error" since this will reduce the error by comparing results. This will be discussed on results chapter.
There were main parameters to be tuned for each classifier. Parameters and applied values have been listed below.

3.3.1.1 Random Forest Classifier tuned parameters

Random Forest classifier was tuned with following possible values.

- Parameter set:
  'max_depth': [80, 90, 100, 110],
  'min_samples_leaf': [1, 2, 3, 4, 5],
  'min_samples_split': [2, 4, 8, 10, 12],
  'n_estimators': [100, 1000, 1500]

After performing grid search, the best estimator was found.

- Best Model with parameters

  RandomForestClassifier(bootstrap=True,
  ccp_alpha=0.0, class_weight =None,
  criterion='gini', max_depth = None, max_features = 'auto',
  max_leaf_nodes = None, max_samples = None,
  min_impurity_decrease = 0.0, min_impurity_split = None,
  min_samples_leaf = 1, min_samples_split = 2,
  min_weight_fraction_leaf = 0.0, n_estimators = 1000,
  n_jobs = None, oob_score = False, random_state = 0, verbose =0,
  warm_start = False)

3.3.1.2 SVM Classifier tuned parameters

SVM classifier was tuned with following possible values.

- Parameter set:

  kernel: [poly, linear, rbf],
  C : [0.1, 1, 10, 100, 1000],
  class_weight : [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}],
  gamma : [1, 0.1, 0.05, 0.01, 0.001, 0.0001]

- Best Model with parameters

  svm.SVC(C=10, break_ties =False, cache_size =200,
  class_weight ={0: 0.3, 1: 0.7}, coef0 =0.0,
  decision_function_shape ='ovr', degree =3, gamma =0.05, kernel ='rbf',
  max_iter =-1, probability = True, random_state =None, shrinking =True,
  tol =0.001, verbose =False)

3.3.1.3 Naïve Bayes Classifier tuned parameters

MultinomialNB classifier was tuned with following possible values.

- Parameter set:

  alpha : [0.1, 0.3, 0.5, 0.9, 1.0, 1.2, 1.5],
  fit_prior : [True, False]

- Best Model with parameters

  MultinomialNB(alpha=0.9, class_prior =None, fit_prior =False)

3.3.1.4 Logistic Regression Classifier tuned parameters

Logistic Regression classifier was tuned with following possible values.

- Parameter set:

  C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

  class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]

  solver = ['liblinear', 'saga']

- Best Model with parameters

  LogisticRegression(C =10, class_weight ={0: 0.3, 1: 0.7},
  dual =False, fit_intercept =True, intercept_scaling =1, l1_ratio = None, max_iter =100, multi_class = auto , n_jobs = None, penalty =l2 , random_state =None, solver = liblinear , tol =0.0001, verbose =0, warm_start =False)

3.3.2 Challenges of classification

Major challenges faced during phase 3 have been listed below.

- Classifiers were behave in different ways. Some classifiers process data very quickly while some others take longer time. Understanding these classifiers were one of a major challenge.
- Hyperparameter tuning was somewhat difficult since it take longer time to give the results. Some classifiers were not given the expected lift of recall after tuning.
- High processing power and performance were needed to classify and hyperparameter tuning.

3.4 Bagging

This section describes the bagging step of the experiment.

3.4.1 Bagging classifier

"sklearn BaggingClassifier " [22] implementation was used as the Bagging classifier. This classifier allow user to adjust the parameters and parameters defines how the bagging classifier would behave. Most important parameters of the bagging classifier have been listed below.

- base_estimator: base estimator or model creates from this. This can be any classifier, if none specified decision tree considers as the base estimator.
- n_estimators: number of estimators should be generated i.e. how many decision tree models should be generated to get the final result.

In this study optimum n_estimators value was 1000. Base estimator was changed to get various results.

3.4.2 How bagging classifier applied to this study

Bagging normally improve the accuracy by averaging majority results from each model. In this study all six tuned classifies were set as the base estimator and bagging classification was performed.

3.4.3 Challenges of bagging

Major challenges faced during bagging phase have been listed below.

- Bagging classifier improves the accuracy of some classifiers and not in some classifiers. This fact showed that bagging classifier cannot be used every time.
- Bagging classifier take more time to process since it has to train all the base estimators. When n_estimators value increased, time taken by the bagging classifier was increased.
- High processing power was needed to classify using bagging classifier.

3.5 Boosting

Boosting is done under classification since boosting classifiers also act as normal classifiers. This section discuss the boosting techniques which were used in this study.

3.5.1 Boosting Classifiers

There are many boosting classifiers available for use. In this study I have selected only two boosting classifiers. Since these are another type of classifiers, those were used with the other classifiers from the beginning and results were discussed at later chapters. The selected boosting techniques were Ada Boost and XGBoost.

3.5.1.1 Ada Boost Classifier

"sklearn AdaBoostClassifier" implementation [36] was used as the classifier. There are some important parameters to adjust the classifier.

- n_estimators: Number of estimators or the weak learners count.
- learning_rate: this rate defines how the weak learners help in the final combination.
- base_estimators: estimator or classifier. Different machine learning algorithms can be assign as base estimator.

3.5.1.2 XGBoost Classifier

"xgboost XGBClassifier" library for python [37] was used as the XGBoost implementation.

There are several important parameters given by the classifier.

- n_estimators: Number of estimators.
- learning_rate: this rate defines how the weak learners help in the final combination.
- min_child_weight: Minimum sum of weights required in a child.
- max_depth: Maximum depth of a tree.
- gamma: Minimum loss reduction need to make a node split.
- colsample_bytree: Denotes the fragment of columns to be randomly samples for each tree.

3.5.2 Hyperparameter tuning

Adaboost and XGBoost classifiers both have the tunable parameters or hyperparameters. These parameters can take different values and according to those values performance of the classifier changes. In order to find the best classification results, hyper parameter tuning was applied to both classifiers.

Same as the previous classifiers tuning, "sklearn GridSearchCV" was used to select the best parameters and best performing model or the estimator. After getting the results of GridSearchCV, best model was selected, trained and evaluated.

Tuned parameter values of classifiers have been listed below.

- Ada boost parameter set:

  learning_rate= [0.01, 0.02, 0.05, 0.08, 0.1]

  n_estimators= [100, 1000, 1500]

- Ada boost best model

  AdaBoostClassifier(algorithm = SAMME.R,

  base_estimator =None, learning_rate =0.08,

  n_estimators =1000, random_state =0)

- XGBoost parameter set:

  learning_rate = [0.01, 0.02, 0.05, 0.08, 0.1]

  n_estimators = [100, 1000, 1500]

  max_depth = [3, 4, 5, 6, 8]

  colsample_bytree = [0.8, 0.9, 1]

  gamma = [0, 1, 3, 5]

- XGBoost best model:

  XGBClassifier(base_score = 0.5, booster = gbtree ,

  colsample_bylevel =1,

  colsample_bynode =1, colsample_bytree = 0.8, gamma = 0,

  learning_rate = 0.05, max_delta_step =0, max_depth = 5,

  min_child_weight = 1, missing =None, n_estimators =1000, n_jobs = 1,

nthread =None, objective = binary:logistic , random_state = 0, reg_alpha = 0.3, reg_lambda =1, scale_pos_weight = 1, seed =None, silent = False, subsample = 0.8, verbosity = 1)

3.5.3 Challenges of boosting

- Boosting techniques take more time than the general machine learning classification techniques.
- Tuning the classifier was hard since the possible values and combinations were high.

3.6 Voting

This is the final and critical step to be performed before reaching to final results evaluation. "sklearn VotingClassifier" implementation [38] was used as the voting classifier and "Soft Voting" was selected as the voting mechanism. Top three best performing classifiers were selected for voting and considered as base estimators. These classifieirs were random Forest, naïve bayes and ada boost. Since all the classifiers were not performed at same level, all the classifiers should not consider as equal. Therefore weights were applied for each classifier. In order to find the best weights, "sklearn GridSearchCV" was used.

Selected weights as follows,

- Random Forest Classifier: **3**
- Naive Bayes Classifier: **3**
- XGBoost Classifier: **2**

As the next step, best weights were set for the voting classifier and final output was taken.

3.6.1 Challenges of voting

There were challenges while performing the voting phase. Some of the challenges have been mentioned below.

- Voting classifier takes the largest time comparing to other classifiers, since it runs all the classifiers separately.

- Weight adjustment was a complex task since there were many combinations and underweighting or overweighting can reduce the final output quality.
- Creating the hyperparameter grid for weight parameter was difficult since all the weights are manually defined.

3.7 Evaluation

As the next step, quality of the outputs were measured by evaluating and comparing the results. This evaluation process was performed in each phase and results were measured. Evaluation criteria have been described below.

3.7.1 Four main evaluators

There are four main values when evaluating the classification problem. All the evaluation criteria based on these four values. Table 3.4 describes those values.

Table 3.4: Main evaluators

		Predicted	
		Positive	**Negative**
Actual	**Positive**	**True Positive(TP)**	**False Negative(FN)**
	Negative	**False Positive(FP)**	**True Negative(TN)**

3.7.1.1 True Negative (TN)

Correctly identified negative values are known as true negatives. In other words the actual class value is negative and predicted class value is also negative.

3.7.1.2 True Positive (TP)

Correctly identified positive values are known as true positives. In other words the actual class value is positive and predicted class value is also positive.

3.7.1.3 False Negative (FN)

If the classified result is negative but actual result is positive, those are count as false negatives.

3.7.1.4 False Positive (FP)

If the classified result is positive but actual result is negative, those are count as false positives.

3.7.2 Accuracy

Accuracy is the first performance measure for any study, but it is not actually correct at all times. Accuracy indicates the ratio of correctly predicted observations to total observations. Following formula give the accuracy measure.

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

If highly imbalanced dataset is present, and minority class does not have many instances and also those are incorrectly classified, still the accuracy measure gives a high value. Therefore; measuring only accuracy does not gives a correct idea about classifiers in this study, since the class imbalance problem is present.

3.7.3 Precision

The ratio of accurately identified positive observations to the all predicted positive observations is known as precision. In other words precision measures number of predicted items are relevant.

$$precision = \frac{TP}{TP + FP}$$

3.7.4 Recall

The ratio of accurately identified positive observations to the actual positive class total observations is known as recall. In other words recall measures number of relevant items selected.

$$recall = \frac{TP}{TP + FN}$$

In this study recall is more important than the precision. It is dangerous to classify racist comment as a non-racist than mistakenly classifying a non-racist comment as racist. Therefore; racism classification is a recall-oriented task, which means recall over precision needs to be accepted.

3.7.5 F Measure

As discussed previously recall and precision are capable of measuring different kind of errors which can be impact for the positive class. If FPs are needed to be minimized, precision needed to be maximized. Likewise FNs can be minimized by increasing recall. F1-Score gives a better approach to capture both recall and precision into a single measure. F1-Measure given in following equation.

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

This is the harmonic mean of the two fractions. The result is a value between 0.0 for the worst F-measure and 1.0 for a perfect F-measure. The important part of the F-measure is that both measures are balanced in importance and that only a good precision and good recall together result in a good F-measure.

3.7.6 F-beta Measure

"*The F-beta-measure measure is an abstraction of the F-measure where the balance of precision and recall in the calculation of the harmonic mean is controlled by a coefficient called beta*" [28]. In some cases, F-measure is important, as an example, when FPs should be minimized, but FNs also important. On the other hand in some scenarios, F-measure is very important but recall also need more attention. In such situations FNs are more critical and should be minimized, but FPs also important. In these cases F-beta measure can be used. In simple words, F-beta apply weights to precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

The beta value can have a several integer values. If beta is 1, it mentioned as F1-measure and if beta is 2, it can be mentioned as F2-score.

Main values used for F-beta measures are,

- F0.5-score: put less weight to recall, more to precision, here beta is 0.5.
- F1-score: put equal weights to recall and precision, beta value is 1.
- F2-score: put less weight to precision, more to recall.

3.7.7 F2 Measure – Main Performance Measure

As discussed earlier this study is a recall oriented task, hence main performance measure was selected as the F2 score.

3.7.8 ROC Curve

A **Receiver Operating Characteristic Curve**, is a diagram that illustrates binary classifier system ability at various thresholds settings. The ROC curve plots the false alarm rate versus the hit rate. In other words it illustrates the TP rate against the FP rate at different threshold values between 0 and 1.

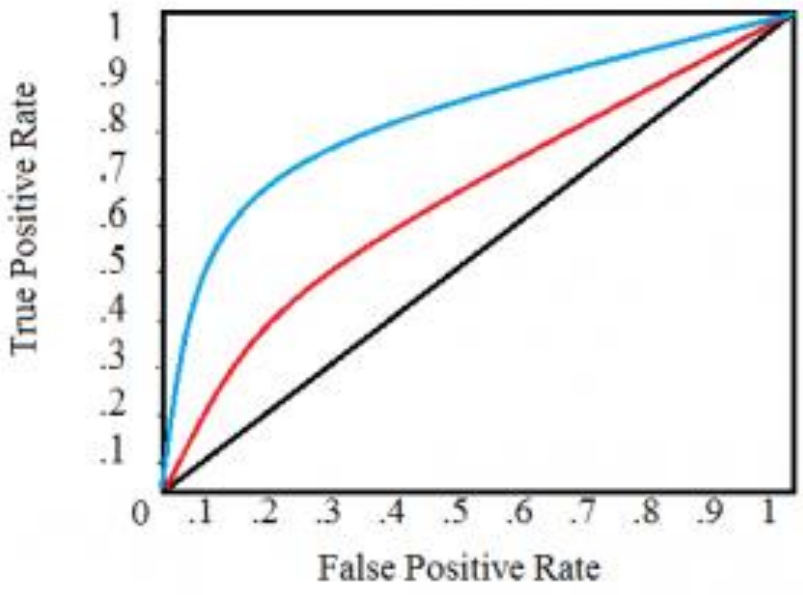


Figure 3.4: Sample ROC curve

Source: [29]

True positive rate (TPR) can be calculated as the TPs divided by the total number of TPs and FNs. TPR also known as sensitivity.

$$TPR = TP/(TP+FN)$$

False positive rate (FPR) can calculated as the number of FPs divided by the total number of FPs and TNs.

$$FPR = FP/(FP+TN)$$

ROC curve can be used to evaluate different kind of models. This performance measure is mostly compared with AUC score in other words Area Under Curve value. This value is capable of representing the skill of the model. If any curve bow up and left side of the plot, it is then referred as skilful model. Diagonal line from coordinates (0,0) to (1,1) is known as no-skill line and has an AUC of 0.5. A good model should perform better than this no-skill model and should draw a ROC curve having higher AUC. Figure 3.4 describes a sample ROC curve.

3.7.9 Precision-Recall Curve

Precision-Recall curve is another performance evaluator which can be used to evaluate or compare models. In Precision-Recall curve, y-axis represent the precision and x-axis represent the recall. Precision-Recall curve also can be evaluated with Area Under the Curve (AUC). If the AUC value is high, then the model performs well.

Figure 3.5 shows sample Precision-Recall curve.

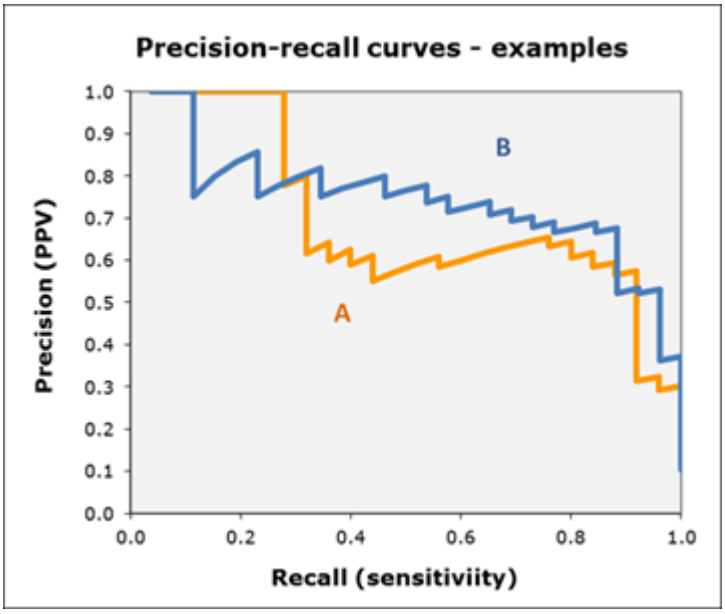


Figure 3.5: Sample precision recall curve

Source: [30]

3.7.10 ROC curve vs. Precision-Recall Curve

As discussed earlier ROC and Precision-Recall curve (PRC) are two main performance measures for binary classification. However Takaya Saito and Marc Rehmsmeier, 2015 [31] claims that PRC is the best performance measure for studies which are having imbalance class problems. Moreover, they are claiming that when it comes to interpretation on ROC with imbalanced datasets, it can interpret ambiguous visual results and also when it comes to classification performance reliability, there can be wrong interpretations of specificity. Also they suggests Precision Recall plots, can provide accurate predictions of classification performance since the fragment of TPs among positive predictions are measured by PRC.

Additionally, Jesse Davis and Mark Goadrich, 2006 [32] propose a similar idea of using PRC instead of ROC when there is a class imbalance situation presence. They report that ROC curve may provide an excessively optimistic results of an algorithms when there is a major class distribution skew. They also suggest that PRCs can be used as an alternative to ROC curves if there is a large class imbalance is involved in the dataset.

By studying these backgrounds and findings it is clear that using the ROC curve when there is a class imbalance situation, will not give the best performance measure. As for the findings, if there are roughly same numbers of data for each class in dataset, ROC curves should be used, but PRC can be used in situations where there is a large class imbalance or medium class imbalance problem is present in the dataset. Therefore Precision-Recall Curve was selected as the final performance measure to evaluate and compare models performance.

3.8 Baseline experiments

It was possible to collect the baseline dataset from the researchers which was used for their study. This dataset contained 238 comments. Two experiments were performed with this dataset to test trained pipeline performance.

3.8.1 Experiment 1

In this experiment trained voting classifier was tested with the whole baseline data set. The purpose of this experiment is to check the performance of the voting classifier with the baseline dataset. This also test the performance of the classifier with unseen data. After performing this step results were analyzed.

3.8.2 Experiment 2

As the next experiment, testing the quality of the pipeline was selected. In order to perform this experiment baseline dataset was split into 75% training data and 25% testing data sets. This was done since baseline experiment also split their dataset with this percentages. After splitting the datasets, training data was fed into pipeline and best model which is voting classifier was trained. As the next step of the experiment, testing data was tested with the voting classifier. This experiment implies the quality of the pipeline. By using the same dataset which was used for the baseline experiment, if the voting classifier performs better than the baseline, it confirms that pipeline process is better than the baseline experiment. This experiment is done several times with dataset shuffling. Results were analyzed at the end of the experiment.

CHAPTER 4 – RESULTS

This chapter contains the results of the study. Results will be carefully analyzed and differences, improvements, trends and performance measures will be observed in this section.

4.1 Results after major steps

Results were evaluated under the evaluation criteria after each major step and differences of the classifier achievements were observed and listed. All the observed results have been listed below. Note that, “Class 1” refers to “Racist” class and “Class 0” refers to “Non-Racist” class. Macro average was taken in f2 score.

4.1.1 Results after applying TF-IDF Vectorizer

As the first step, first result set were observed after applying TF-IDF Vectorizer and models as un-tuned classifiers. Results has been listed in Table 4.1.

Table 4.1: Results of TF-IDF vectorizer

After Applying TF-IDF Vectorizer										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8353	0.83	0.87	0.98	0.37	0.94	0.42	0.6845	439	8
									89	53
Naïve Bayes	0.7606	0.76	0.60	1.00	0.02	0.93	0.02	0.4821	445	2
									139	3
SVM	0.8387	0.84	0.81	0.97	0.43	0.94	0.47	0.7074	433	14
									81	61
Logistic Regression	0.7707	0.77	0.82	1.00	0.06	0.94	0.07	0.5090	445	2
									133	9
Ada Boost	0.8370	0.85	0.74	0.95	0.49	0.92	0.52	0.7275	423	24
									72	70
XGBoost	0.8387	0.84	0.83	0.97	0.42	0.94	0.46	0.7024	435	12
									83	59

After features were generated by TF-IDF Vectorizer, all six classifiers were trained and tested with data. At this stage stop words were not removed and stemming was not applied as well. After observing the results achieved by each classifier in this stage, Ada boost was scored the best F2 score as 0.7275 and maximum recall on class 1 as 0.49. Ada boost was able to get the maximum TN (True Negative) count (70) in confusion matrix as well.

The second highest scores were achieved by SVM classifier. As the third performing classifier, XGBoost was selected. Even though it perform closely to Random forest, XGBoost has the higher F2 score and accuracy than the Random forest. All the ensemble techniques were performed well in this stage and SVM also performed well. Logistic Regression and Naïve bayes were not performed as expected and Naïve bayes has the lowest F2 score 0.4821. The recall for class 1 and TN (True Negative) values of these both classifiers were very low.

4.1.2 Results after stemming

Results were observed and analyzed after stemming. Table 4.2 shows the results after performing stemming.

As result shows all classifiers were able to improve overall performance by stemming, but naïve bayes and logistic regression have gained good performance improvements compared to others. Random Forest, SVM and Ada Boost were also performed as the best classifiers by gaining highest Accuracies, F2 scores and class 1 recalls. The important observation is that these three classifiers perform at some higher level while others performing differently at this stage. SVM was able to gain 432 TP and 75 TN values. Highest TP value was achieved by naïve bayes classifier but still the TN value is 38, which is very low compared to other classifiers except Logistic regression classifier. Logistic regression classifier also gained 437 TP count and TN count as 38. Naïve bayes and Logistic regression classifiers are having lowest TN counts as well as high FP (False Positive) count which made their performance scores low.

Table 4.2: Results after stemming

After Stemming										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8692	0.88	0.82	0.96	0.58	0.94	0.62	0.7813	429	18
									59	83
Naïve Bayes	0.8132	0.81	0.86	0.99	0.27	0.94	0.31	0.6277	441	6
									104	38
SVM	0.8607	0.87	0.83	0.97	0.53	0.94	0.56	0.7571	432	15
									67	75
Logistic Regression	0.8064	0.81	0.79	0.98	0.27	0.93	0.30	0.6233	437	10
									104	38
Ada Boost	0.8353	0.87	0.70	0.93	0.55	0.91	0.57	0.7439	414	33
									64	78
XGBoost	0.8387	0.84	0.82	0.97	0.42	0.94	0.46	0.70	434	13
									82	60

4.1.3 Results after removing stop words

Stop words removal is a common strategy to make the feature vector richer. By removing stop words, corpus removes common words which are not very relevant to final result. Table 4.3 lists the results of each classifier after removing stop words.

After removing stop words all classifiers were able to improve their performances. Naïve Bayes was able to increase its F2 score from 0.6277 to 0.6510. This is a considerable lift compared to others. Logistic Regression was able to achieve F2 score 0.6549, previously it was 0.6233. Even though the TP counts are same, Logistic regression was able to increase its TN count by 8 and that particular change itself led to improve the performance. While Naïve bayes was lifting its performance, Random forest, Ada boost and SVM classifiers were able to protect their places as the best classifiers at this stage. Random forest classifier was able to increase its TN value by 2 (83 to 85).

Table 4.3: Results after removing stop words

After removing stop words										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8726	0.88	0.83	0.96	0.60	0.94	0.63	0.7883	429	18
									57	85
Naïve Bayes	0.8183	0.82	0.82	0.98	0.32	0.94	0.36	0.6510	437	10
									97	45
SVM	0.8624	0.87	0.81	0.96	0.56	0.93	0.59	0.7695	428	19
									62	80
Logistic Regression	0.8200	0.82	0.82	0.98	0.32	0.94	0.36	0.6549	437	10
									96	46
Ada Boost	0.8522	0.88	0.75	0.94	0.58	0.92	0.61	0.7678	419	28
									59	89
XGBoost	0.8404	0.85	0.81	0.97	0.44	0.93	0.48	0.7135	432	15
									79	63

4.1.4 Results after text convert to base character

As discussed in previous chapters, Sinhala language contain many letters, same word can be written using different letters while keeping the meaning and sound same. People use these letters differently and most of the time it varies on their writing style. By removing these various characters and replace them with a single letter can reduce many redundant words from feature vector.

Results after converting text to base characters has been shown in Table 4.4. After converting special letters to base letter, there were noticeable changes in the results. The interesting fact appear while observing the results is only Ada boost classifier was unable to increase its performances but all the other classifiers improved their performances due to this change. SVM, Logistic Regression and XGBoost were able to increase both TP and TN counts and reduce its FP and FN counts, but Ada boost classifier reduced its TN and FN counts and increased TP and FP counts.

Table 4.4: Results after text convert to base character

After convert to base character										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8794	0.89	0.83	0.96	0.63	0.94	0.66	0.8041	428	19
									52	90
Naïve Bayes	0.8200	0.82	0.81	0.98	0.33	0.94	0.37	0.6577	436	11
									95	47
SVM	0.8743	0.88	0.84	0.96	0.59	0.94	0.62	0.7875	431	16
									58	84
Logistic Regression	0.8302	0.83	0.85	0.98	0.36	0.94	0.40	0.6756	438	9
									91	51
Ada Boost	0.8522	0.87	0.78	0.95	0.54	0.93	0.57	0.7550	425	22
									65	77
XGBoost	0.8573	0.86	0.85	0.97	0.50	0.94	0.54	0.7454	434	13
									71	71

4.1.5 Results after SMOTE oversampling

SMOTE (Synthetic Minority Oversampling Technique) is a statistical technique for increasing the number of classes in the dataset. Table 4.5 shows the results after SMOTE oversampling. By observing these results, it becomes clear that all the classifiers have increased their performances after this step. Performance of the Random forest classifier was at the expected level and performance increase rate was not considerably larger compared to other classifiers.

Naïve bayes and logistic regression classifiers were the low performing classifiers of previous steps, but after applying SMOTE there was a major lift of the performance. Naïve bayes achieved 0.7913 as F2 score and previously it was 0.6577. The highest lift of the naïve bayes was observed in recall of class1, which increased from 0.33 to 0.80.

Also naïve bayes was able to gain more TNs (47 to 113) which led to huge performance improvement of the classifier. Naïve Bayes classifier was able to outperform Ada boost, logistic regression and SVM as well. Logistic regression also improved its performance, by increasing F2 score from 0.6756 to 0.7769 and recall of the class 1 from 0.36 to 0.80.

Other classifiers were also able to lift their performances, especially SVM and XGBoost. The noticeable fact is that all these classifiers were able to lift their recall of class 1 after applying this SMOTE technique. Random forest was still performing at the top of all these classifiers and was able to achieve the highest F2 score at this stage (0.8225). As the second highest performing classifier, XGBoost was able to gain 0.8081 as the F2 score.

Table 4.5: Results after SMOTE oversample

After SMOTE oversample										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8675	0.92	0.72	0.91	0.74	0.90	0.73	0.8225	406	41
									37	105
Naïve Bayes	0.8149	0.93	0.59	0.82	0.80	0.84	0.74	0.7913	367	80
									29	113
SVM	0.7724	0.91	0.52	0.78	0.76	0.79	0.69	0.7478	347	100
									34	108
Logistic Regression	0.7979	0.92	0.56	0.80	0.80	0.82	0.73	0.7769	357	90
									29	113
Ada Boost	0.8387	0.89	0.66	0.89	0.67	0.89	0.66	0.7805	399	48
									47	95
XGBoost	0.8794	0.89	0.81	0.95	0.65	0.94	0.67	0.8081	426	21
									50	92

4.1.6 Results after applying N-grams

Results of each classifier were observed with N-grams and listed in Table 4.6.

Table 4.6: Results after applying N-grams

N-grams										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.7589	0.76	0.50	0.99	0.04	0.93	0.05	0.4920	441	6
									136	6
Naïve Bayes	0.7538	0.76	0.00	0.99	0.00	0.95	0.00	0.4675	444	3
									142	0
SVM	0.7606	0.76	0.57	0.99	0.03	0.93	0.03	0.4857	444	3
									138	4
Logistic Regression	0.7589	0.76	0.00	1.00	0.00	0.94	0.00	0.4701	447	0
									142	0
Ada Boost	0.7589	0.77	0.50	0.98	0.05	0.93	0.06	0.4955	440	7
									135	7
XGBoost	0.7572	0.76	0.00	1.00	0.00	0.93	0.00	0.4692	446	1
									142	0

Bigrams were used as the features and results were observed after classification. All the classifiers were unable to perform with the bigrams and performances of the classifiers were very low. Even though all the classifiers were able to achieve high accuracies, F2 scores and recall score of class 1 were very low. This result set was rejected since it did not give any quality output and did not have any support for the research questions.

4.1.7 Results after applying fastText

Feature matrix was generated with fastText and tested with each classifier. Results has been listed in Table 4.7.

Table 4.7: Results after applying fastText

After fastText										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8505	0.86	0.82	0.97	0.49	0.94	0.52	0.7355	432	15
									73	69
SVM	0.8268	0.83	0.83	0.98	0.35	0.94	0.39	0.6705	437	10
									92	50
Logistic Regression	0.8200	0.82	0.83	0.98	0.32	0.94	0.36	0.6522	438	9
									97	45
Ada Boost	0.8404	0.85	0.78	0.96	0.47	0.93	0.51	0.7231	428	19
									75	67
XGBoost	0.8319	0.83	0.83	0.98	0.38	0.94	0.42	0.6847	436	11
									88	54

At this stage some classifiers performed well with fastText features but some did not. Random forest classifier was able to achieve F2 score as 0.7355 and it was the highest score compared to other classifiers. Random forest classifier was also able to gain 0.49 class 1 recall value, which is really good at initial stage.

The other classifier which performed well was logistic regression classifier. It was able to gain F2 score 0.6522 which is better than the logistic regression classifier with TD-IDF features. All the other classifiers was unable to achieve more than they achieved with TD-IDF features.

4.1.8 Results after hyperparameter tuning

Hyperparameter tuning was performed and results were observed as the next step. As for the results shown in Table 4.8, all the classifiers were improved their performances.

Table 4.8: Results after hyperparameter tuning

After hyperparameter tuning										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8658	0.92	0.71	0.90	0.75	0.90	0.74	0.8246	403	44
									35	107
Naïve Bayes	0.8149	0.93	0.58	0.82	0.81	0.83	0.75	0.7943	365	82
									27	115
SVM	0.8047	0.92	0.57	0.81	0.79	0.83	0.73	0.7811	362	85
									30	112
Logistic Regression	0.8047	0.93	0.57	0.81	0.80	0.82	0.74	0.7842	360	87
									28	114
Ada Boost	0.8471	0.90	0.69	0.90	0.67	0.90	0.67	0.7874	404	43
									47	95
XGBoost	0.8743	0.90	0.77	0.93	0.69	0.92	0.70	0.8155	417	30
									44	98

All the classifiers increased their recall of class 1 and F2 score after applying tuning, but all the classifiers except SVM classifier and Ada boost were unable to improve the accuracies and recall of the class 0. SVM was able to gain the highest performance lift compare to others at this stage.

Random forest was perform as the best classifier at this stage as well. It was able to achieve the highest F2 score at this stage and it was 0.8246. Recall of class 1 was 0.75 and accuracy was 0.8658. As the second performing classifier, XGBoost classifier was selected. Naïve bayes classifier was performed at the third place and outperformed Ada boost, SVM and logistic regression classifiers.

4.1.9 Results after bagging

Table 4.9 shows the results which were observed after performing bagging.

Table 4.9: Results after bagging

After Bagging										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8743	0.92	0.73	0.91	0.75	0.91	0.74	0.8318	408	39
									35	107
Naïve Bayes	0.8115	0.94	0.58	0.81	0.83	0.82	0.76	0.7959	360	87
									24	118
SVM	0.8149	0.93	0.58	0.82	0.80	0.83	0.74	0.7928	366	81
									28	114
Logistic Regression	0.7945	0.93	0.55	0.79	0.81	0.81	0.74	0.7770	353	94
									27	115
Ada Boost	0.8641	0.91	0.73	0.92	0.70	0.91	0.70	0.8089	410	37
									43	99
XGBoost	0.8743	0.90	0.77	0.94	0.68	0.92	0.69	0.8117	419	28
									46	96

After applying ensemble bagging technique for each classifier and use them as the base estimator, results were observed. XGBoost classifier was unable to improve any performance measure but others were able to improve performance after this step. SVM and Ada boost were able to increase all performance measures. Random forest only increased F2 score while naïve bayes increased accuracy and recall of the class and maintained same F2 score. Logistic regression only increased its recall of the class 1 and was not able to increase accuracy or F2 score.

4.1.10 Results after voting

Voting ensemble technique was used to get a voted results and results were listed in Table 4.10.

Table 4.10: Results after voting

After Voting										
Classifier	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
Random Forest	0.8743	0.92	0.73	0.91	0.75	0.91	0.74	0.8318	408	39
									35	107
Naïve Bayes	0.8115	0.94	0.58	0.81	0.83	0.82	0.76	0.7959	360	87
									24	118
SVM	0.8149	0.93	0.58	0.82	0.80	0.83	0.74	0.7928	366	81
									28	114
Logistic Regression	0.8047	0.93	0.57	0.81	0.80	0.82	0.74	0.7842	360	87
									28	114
Ada Boost	0.8641	0.91	0.73	0.92	0.70	0.91	0.70	0.8089	410	37
									43	99
XGBoost	0.8743	0.90	0.77	0.93	0.69	0.92	0.70	0.8155	417	30
									44	98
Voting	0.8794	0.93	0.73	0.91	0.79	0.91	0.77	0.8447	406	41
									30	112

Voting classifier was the last step to perform and results were observed. Voting classifier was able to achieve the best results of all results. As the table describes, voting classifier was able to gain the best F2 score, best accuracy compared to other classifiers best results. Voting classifier gives the majority output by ensembling. Therefore Voting classifiers often perform as the best classifier since there are different types of classifiers and majority voting comes as the final result.

4.1.11 Result comparison with precision recall graph

As the final evaluation method, precision recall graph was selected. Precision recall graph of each classifier were plot in one graph and performances were compared. Appendix V shows the precision recall graph comparison of each classifier.

As the precision recall graph describes, voting classifier achieved the score value of 0.771, which is the highest score of precision recall curve AUC values. XGBoost performed as the second highest performing classifier and as the third performing classifier, random forest classifier can be observed. Naïve bayes can be identified as the next performing classifier. Ada boost classifier performance is lower than the Naïve bayes but higher than the Logistic regression classifier. As precision recall curve shows, the lowest performing classifier can be identified as the SVM classifier and it is having an AUC value of 0.652.

4.1.12 Result comparison with ROC curve.

Even though the best result evaluator of this study is precision recall graph, ROC curve also plot to compare the performance but not considered as the final result evaluator. This ROC curve compare all the classifiers by AUC score.

As the Appendix IV shows, best performing classifier can be identified as the voting classifier. The second highest performing classifier can be observed as the naïve bayes classifier and random forest classifier becomes the third highest performing classifier. As the fourth highest performance classifier, XGBoost classifier can be observed. SVM and Logistic regression performance can be list after the XGBoost and as the lowest performing classifier Ada boost was observed.

4.2 Result improvements of each classifier

This section discuss how each classifier increase or decrease their performance after major steps.

4.2.1 Random forest classifier

All the results of random forest classifier was listed and compared. This comparison has been shown in Table 4.11. Results of this analysis show that the random forest classifier achieved average results in early steps and better results in latter steps. According to results, classifier was able to gain 0.6845 as F2 score from the beginning and highest F2 score as 0.8318 was achieved after performing bagging. Accuracy also increased throughout the steps.

At the first step accuracy was at 0.8353 and after bagging it became 0.8743. Recall of the class1 increased after performing each action and this results directly had an impact on other performance measures as well.

Table 4.11: Result comparison of random forest classifier

Random forest classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.8353	0.83	0.87	0.98	0.37	0.94	0.42	0.6845	439	8
									89	53
Stemming	0.8692	0.88	0.82	0.96	0.58	0.94	0.62	0.7813	429	18
									59	83
Remove stop words	0.8726	0.88	0.83	0.96	0.60	0.94	0.63	0.7883	429	18
									57	85
Convert to base character	0.8794	0.89	0.83	0.96	0.63	0.94	0.66	0.8041	428	19
									52	90
SMOTE oversample	0.8675	0.92	0.72	0.91	0.74	0.90	0.73	0.8225	406	41
									37	105
Hyperparameter tuning	0.8658	0.92	0.71	0.90	0.75	0.90	0.74	0.8246	403	44
									35	107
Bagging	0.8743	0.92	0.73	0.91	0.75	0.91	0.74	0.8318	408	39
									35	107

The next important observation of the results is that FP (False Positive) and FN (False negative) values decreased after each step. Which means classifiers ability to classify between classes became high after performing these actions.

Figure 4.1 shows the random forest classifier performance levels of each major step. After analyzing previous results, it became clear that the best classifier before apply voting was random forest classifier since classifier was able to achieve highest performance measures at bagging stage.Random forest classifier was able to perform at a very good level in each step. By observing this graph, it is easier to conclude that performance improvement was happened in each step and classifier continuously improved its performance.

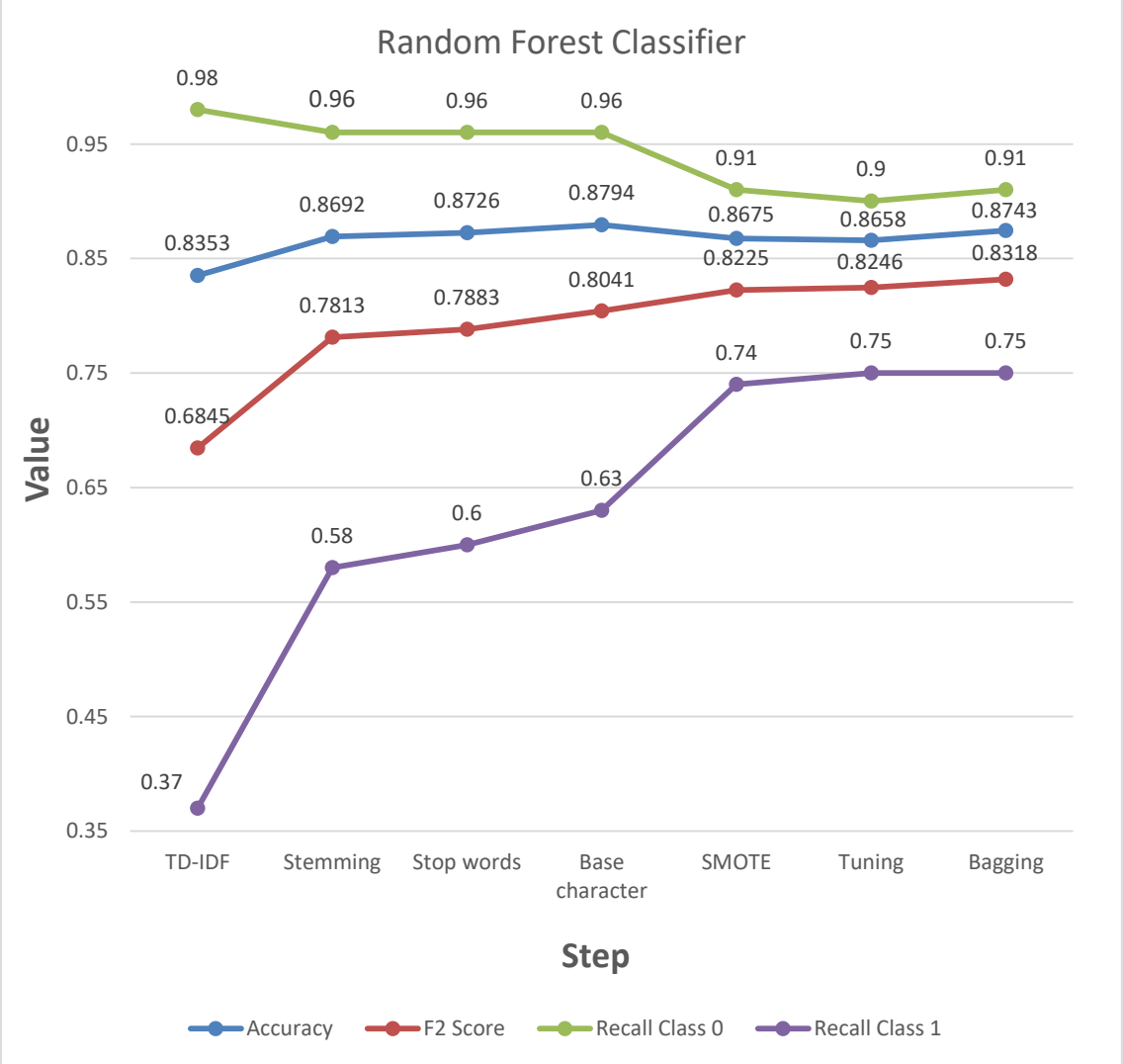


Figure 4.1: Random forest classifier performance

As appears in the graph random forest classifier was sensitive to two changes, which are text stemming and SMOTE oversampling. These steps have improved the recall of class 1 and F2 scores drastically. The other important observation is that the classifier maintained its accuracy levels at a higher and stable level. After stemming the accuracy measure was stabled and decreased at the SMOTE step.

4.2.2 Naïve Bayes Classifier

All results of naïve bayes classifier after major steps have been listed in Table 4.12.

Table 4.12: Result comparison of naive bayes classifier

Naïve Bayes Classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.7606	0.76	0.60	1.00	0.02	0.93	0.02	0.4821	445	2
									139	3
Stemming	0.8132	0.81	0.86	0.99	0.27	0.94	0.31	0.6277	441	6
									104	38
Remove stop words	0.8183	0.82	0.82	0.98	0.32	0.94	0.36	0.6510	437	10
									97	45
Convert to base character	0.8200	0.82	0.81	0.98	0.33	0.94	0.37	0.6577	436	11
									95	47
SMOTE oversample	0.8149	0.93	0.59	0.82	0.80	0.84	0.74	0.7913	367	80
									29	113
Hyperparameter tuning	0.8149	0.93	0.58	0.82	0.81	0.83	0.75	0.7943	365	82
									27	115
Bagging	0.8115	0.94	0.58	0.81	0.83	0.82	0.76	0.7959	360	87
									24	118

As the results suggest, naïve bayes classifier increased its performance in each step. Initially F2 score of the classifier was at 0.4821 and as the best score it reached to 0.7959. Accuracy also increased and gained the best score at the hyperparameter tuning and it indicates as 0.8149.

The next important performance measure is recall of class 1 and initially it was at unacceptable level of 0.02. But in SMOTE oversampling step recall increased by 0.47 and reached to 0.80. 0.83 can be observe as the highest recall of class 1. FP count of the naïve bayes classifier was 139 at the first step, but as the lowest count it reached to 24. TN count was initially show as the 3 and finally increased to 118. As result shows, bagging step gives the best classification output of the naïve bayes classifier.

As shown in the Figure 4.2, this classifier was not perform well in the early steps but improvement continued. However the drastic improvement points were stemming and the SMOTE oversampling step.

After these steps classifier increased its class 1 recall by a huge values. F2 scores were also increased. Hyperparameter tuning and bagging steps also made a minor improvement on the classifier performance. It is clear that naïve bayes classifier is sensitive to data points, without having considerable data points, the classifier might not able to perform at its best.

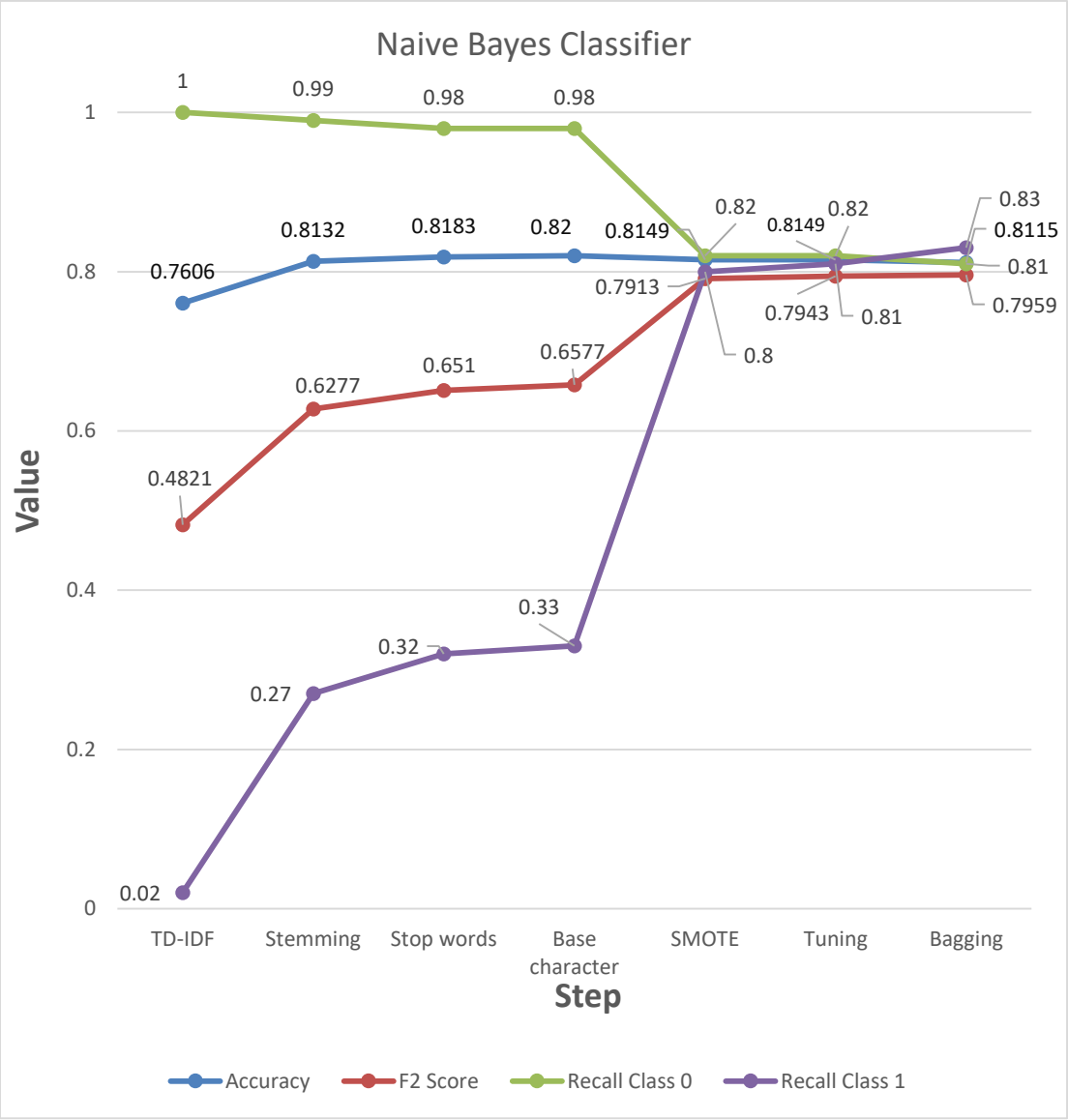


Figure 4.2: Naive bayes classifier performance

4.2.3 SVM Classifier

Result comparison of SVM classifier has been shown in Table 4.13. As results show, classifier was able to increase its performance throughout all the steps. SVM was able to gain 0.8353 accuracy score at the beginning but decreased its accuracy to 0.8047 at the hyperparameter tuning step (step 6). Bagging step can be identified as the best performing level of the classifier since it achieved the F2 score as 0.7928 while having the class 1 recall score as 0.80. Even though the second highest F2 score of 0.7875 can be seen after convert to base character step, the recall of the class 1 is at comparatively low score which is 0.59. FP count was at 89 at the beginning and reduced to 28 after bagging. TN count was at 53 at the beginning and at the best performance level, it reached to 114.

Table 4.13: Result comparison of SVM classifier

SVM Classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.8353	0.83	0.87	0.98	0.37	0.94	0.42	0.6845	439	8
									89	53
Stemming	0.8607	0.87	0.83	0.97	0.53	0.94	0.56	0.7571	432	15
									67	75
Remove stop words	0.8624	0.87	0.81	0.96	0.56	0.93	0.59	0.7695	428	19
									62	80
Convert to base character	0.8743	0.88	0.84	0.96	0.59	0.94	0.62	0.7875	431	16
									58	84
SMOTE oversample	0.7724	0.91	0.52	0.78	0.76	0.79	0.69	0.7478	347	100
									34	108
Hyperparameter tuning	0.8047	0.92	0.57	0.81	0.79	0.83	0.73	0.7811	362	85
									30	112
Bagging	0.8149	0.93	0.58	0.82	0.80	0.83	0.74	0.7928	366	81
									28	114

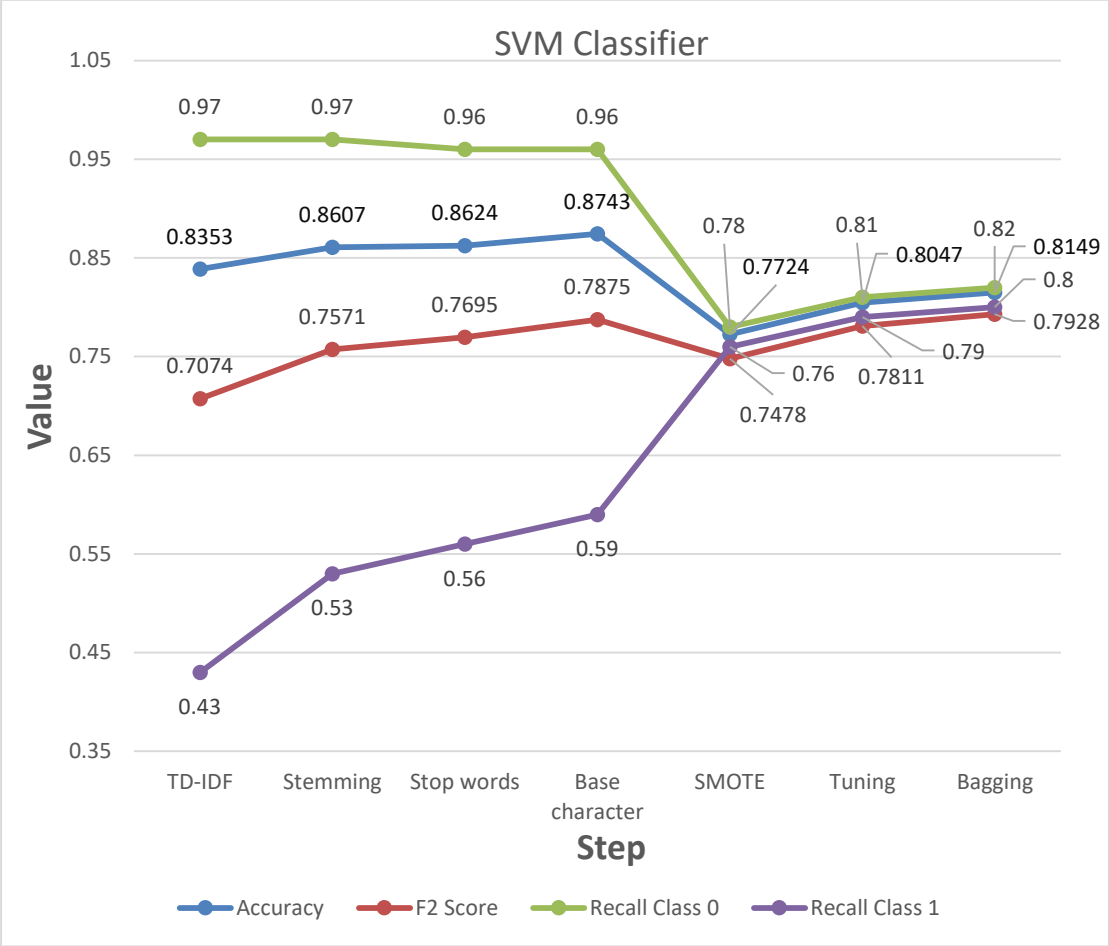


Figure 4.3: SVM classifier performance

As Figure 4.3 indicates, SVM classifier was able to increase its performance after some steps. Even though the F2 score and recall of class 1 were increasing from the beginning, after applying SMOTE oversampling technique, F2 score was decreased. But classifier improved its recall of class 1 by a considerable amount and that was a valuable change for this study. Hence SMOTE oversampling considered as the turning point of the SVM classifier. However after hyperparameter tuning, classifier was able to increase all performance measures. Therefore tuning step also considered as an improvement point for the classifier. Bagging is also helped the classifier to gain some good performance score. Therefore bagging was identified as a plus point of performance for SVM classifier.

4.2.4 Logistic Regression Classifier

Table 4.14 shows the result comparison of logistic regression classifier. As results show, highest performance of logistic regression classifier can be seen after applying hyperparameter tuning (step 6). At this step F2 score reached to 0.7842 which is the highest F2 score of the classifier and recall of the class1 (0.80) is higher than the earlier steps. After step 6 recall of the class 1 increased by 0.01 but F2 score decreased to 0.7770. Therefore step 6 considered as the best performing level of the classifier. F2 score of the first step was at 0.5090 and recall of the class 1 was at 0.06. At the step 6 recall increased to 0.80.

At the first step TN count was 9 but after performing various steps throughout the study, it reached to 114 at the best stage. At this stage FP count can be identify as 28 which is relatively much lower than the initial step count 133. Therefore after hyperparameter tuning, classifier reached to maximum performing level.

Table 4.14: Result comparison of logistic regression classifier

Logistic Regression Classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.7707	0.77	0.82	1.00	0.06	0.94	0.07	0.5090	445	2
									133	9
Stemming	0.8064	0.81	0.79	0.98	0.27	0.93	0.30	0.6233	437	10
									104	38
Remove stop words	0.8200	0.82	0.82	0.98	0.32	0.94	0.36	0.6549	437	10
									96	46
Convert to base character	0.8302	0.83	0.85	0.98	0.36	0.94	0.40	0.6756	438	9
									91	51
SMOTE oversample	0.7979	0.92	0.56	0.80	0.80	0.82	0.73	0.7769	357	90
									29	113
Hyperparameter tuning	0.8047	0.93	0.57	0.81	0.80	0.82	0.74	0.7842	360	87
									28	114
Bagging	0.7945	0.93	0.55	0.79	0.81	0.81	0.74	0.7770	353	94
									27	115

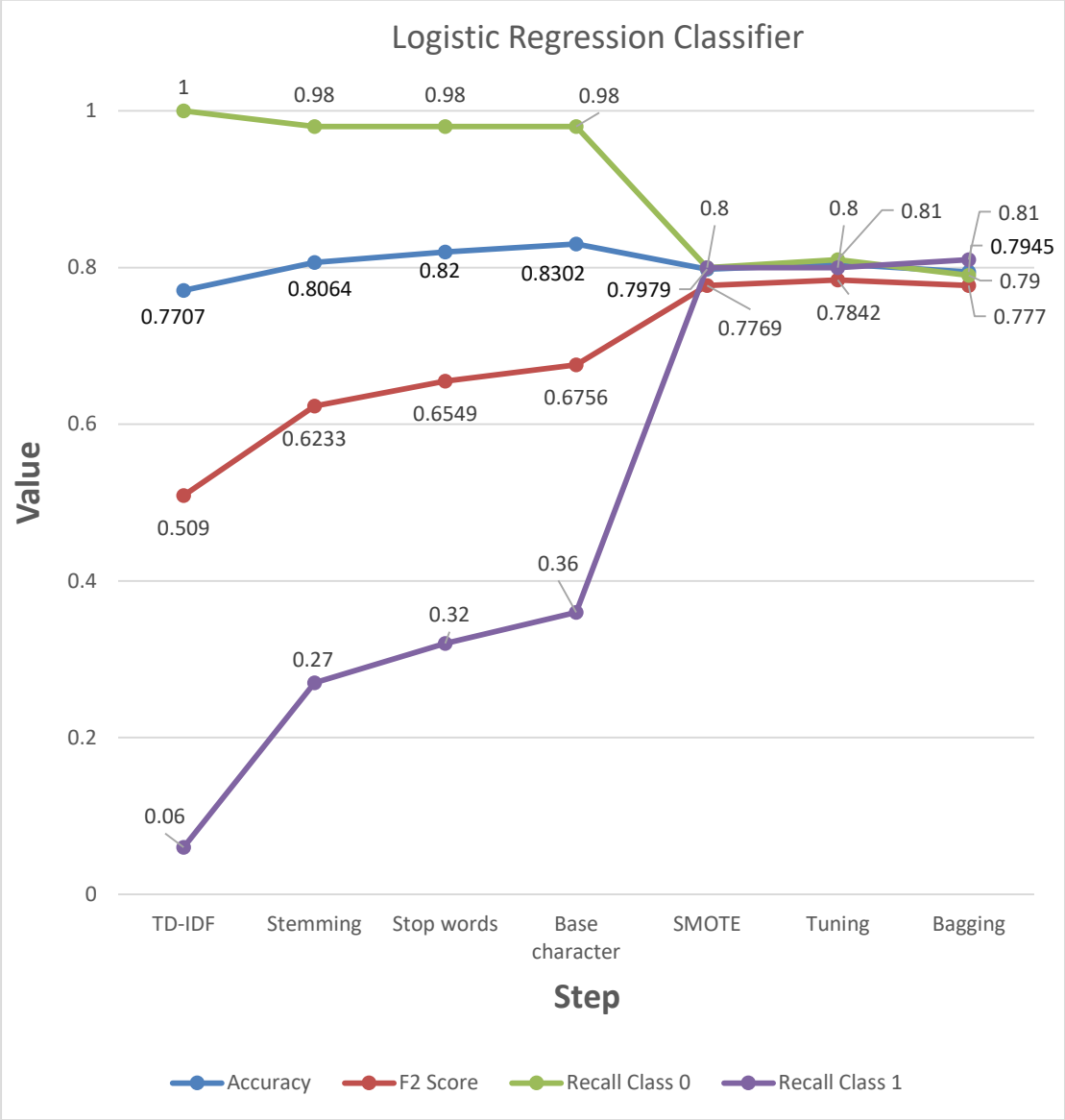


Figure 4.4: Logistic regression classifier performance

As Figure 4.4 describes, logistic regression classifier improved its performance throughout all the steps. Classifier was underperforming at the earlier steps but drastic improvement can be seen after applying stemming and SMOTE oversampling. These particular steps have changed the performance of the classifier in a major level. There was a minor performance decrease after applying bagging. Hyperparameter tuning was also helped the classifier to improve its class 1 recall and considered as a positive change of results. After performing these steps there was a balanced recall for both class 1 and class 0.

4.2.5 Ada Boost Classifier

Result comparison of Ada boost classifier has been shown in Table 4.15. As results describes, ada boost classifier shows multiple increases and single decrease of performance throughout the study. Ada boost classifier started at 0.7275 of F2 score and finally reached to 0.8089. At this best F2 score, recall of the class1 was at 0.70, which is also the best F2 score. Best accuracy also can be observed at this stage.

Since this study more focus F2 and recall scores, bagging step identified as the best outcome of the classifier. FP count initially was at 72 and reduced to 43, which was a great achievement of the classifier. TN count also increased from 70 to 99. Therefore bagging can be identified as the best performing level of the ada boost classifier.

Table 4.15: Result comparison of ada boost classifier

Ada Boost Classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.8370	0.85	0.74	0.95	0.49	0.92	0.52	0.7275	423	24
									72	70
Stemming	0.8353	0.87	0.70	0.93	0.55	0.91	0.57	0.7439	414	33
									64	78
Remove stop words	0.8522	0.88	0.75	0.94	0.58	0.92	0.61	0.7678	419	28
									59	89
Convert to base character	0.8522	0.87	0.78	0.95	0.54	0.93	0.57	0.7550	425	22
									65	77
SMOTE oversample	0.8387	0.89	0.66	0.89	0.67	0.89	0.66	0.7805	399	48
									47	95
Hyperparameter tuning	0.8471	0.90	0.69	0.90	0.67	0.90	0.67	0.7874	404	43
									47	95
Bagging	0.8641	0.91	0.73	0.92	0.70	0.91	0.70	0.8089	410	37
									43	99

As shown in Figure 4.5, Ada boost classifier behave differ than the other classifiers in a single step, which is converting to base character. While other classifiers increasing their performances at this stage, ada boost decreased its performance. SMOTE oversampling step is the largest performance lift of the classifier. All the other steps helped ada boost classifier to gain more performance and highest performance can be seen at bagging step.

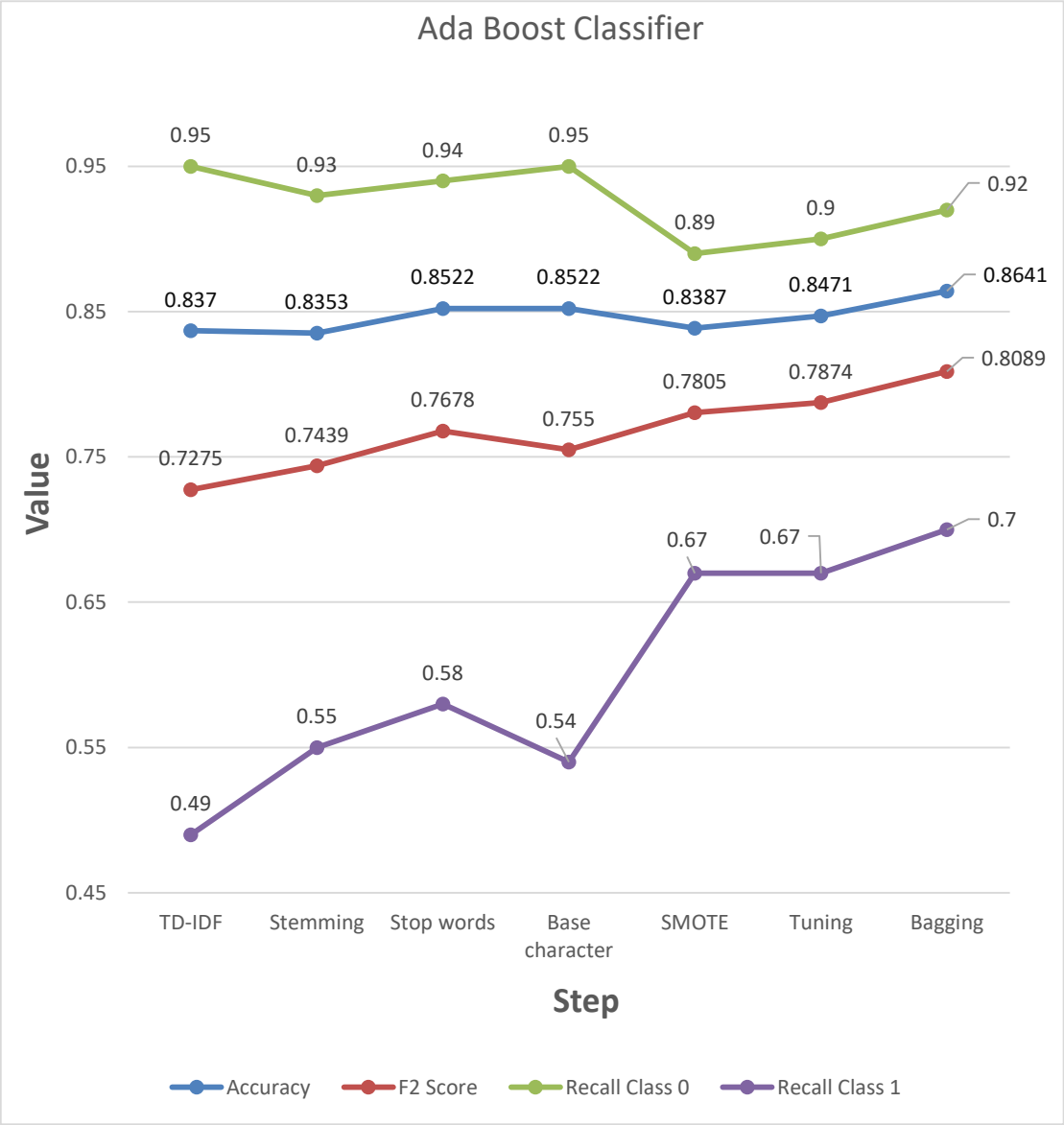


Figure 4.5: Ada boost classifier performance

However after hyperparameter tuning, classifier improved its performance, but recall of the class 1 remained same. The next improvement was observed after bagging. Even though boosting algorithms normally do not combine with the bagging algorithms since both are ensemble techniques and both are perform in somewhat similar way. But this results shows that even boosting classifiers can be combine with bagging techniques and still get the best results.

4.2.6 XGBoost Classifier

Table 4.16 shows result comparison of XGBoost classifier.

Table 4.16: Result comparison of XGBoost classifier

XGBoost Classifier										
Step	**Accuracy**	**Precision**		**Recall**		**F2-Score**			**Confusion Matrix**	
		Class 0	**Class 1**	**Class 0**	**Class 1**	**Class 0**	**Class 1**	**F2 Average**	**TP**	**FN**
									FP	**TN**
TF-IDF Features	0.8387	0.84	0.83	0.97	0.42	0.94	0.46	0.7024	435	12
									83	59
Stemming	0.8387	0.84	0.82	0.97	0.42	0.94	0.46	0.70	434	13
									82	60
Remove stop words	0.8404	0.85	0.81	0.97	0.44	0.93	0.48	0.7135	432	15
									79	63
Convert to base character	0.8573	0.86	0.85	0.97	0.50	0.94	0.54	0.7454	434	13
									71	71
SMOTE oversample	0.8794	0.89	0.81	0.95	0.65	0.94	0.67	0.8081	426	21
									50	92
Hyperparameter tuning	0.8743	0.90	0.77	0.93	0.69	0.92	0.70	0.8155	417	30
									44	98
Bagging	0.8743	0.90	0.77	0.94	0.68	0.92	0.69	0.8117	419	28
									46	96

As the result shows, XGBoost classifier initially showed average results and step by step results were improved. At the initial step F2 score was indicated as the 0.7024 and after bagging it can be seen at the score level of 0.8117.

While increasing the F2 score, XGBoost was able to increase the other performance measures as well. Recall of class 1 was increased from 0.42 to 0.68 and accuracy was increased from 0.8387 to 0.8743.

XGBoost classifier was able to increase the performance in all steps except bagging step. Therefore hyperparameter tuning can be identified as the best performing step of the classifier. When observing the confusion matrices of the results, it becomes clear that FP count decreased in each step and it decreased from 83 to 44 at the best stage. Also TN count increased from 59 to 98. At the last step F2 score is decreased to 0.8117, therefore bagging can be identified as performance decreasing step of the classifier.

If a classifier maintained a stable results, not gave any unexpected results, it was undoubtedly the XGBoost classifier. XGBoost known as Xtream Gradient Boost. This is a well-known classifier to use at almost every classification problem. As indicates in Figure 4.6, XGBoost decreased its performance only once and that also a very small decrease compare to other lifts. Even in the initial steps, this classifier was able to produce good results and continued to do so while improving the results.

There were two main steps which made major impacts on XGBoost classifier. The first one is convert the text characters to base character. This gave a considerable lift to F2 score and recall of the class 1. The second step which gave a good lift of performance to classifier is SMOTE oversampling. This was the largest lift in XGBoost.

Hyperparameter tuning also played a major role of increasing performance of the XGBoost classifier. Bagging on the other hand was unable to do much to increase the performance but as shown in the graph, there is a minor improvement of some performance measures such as recall of class 0.

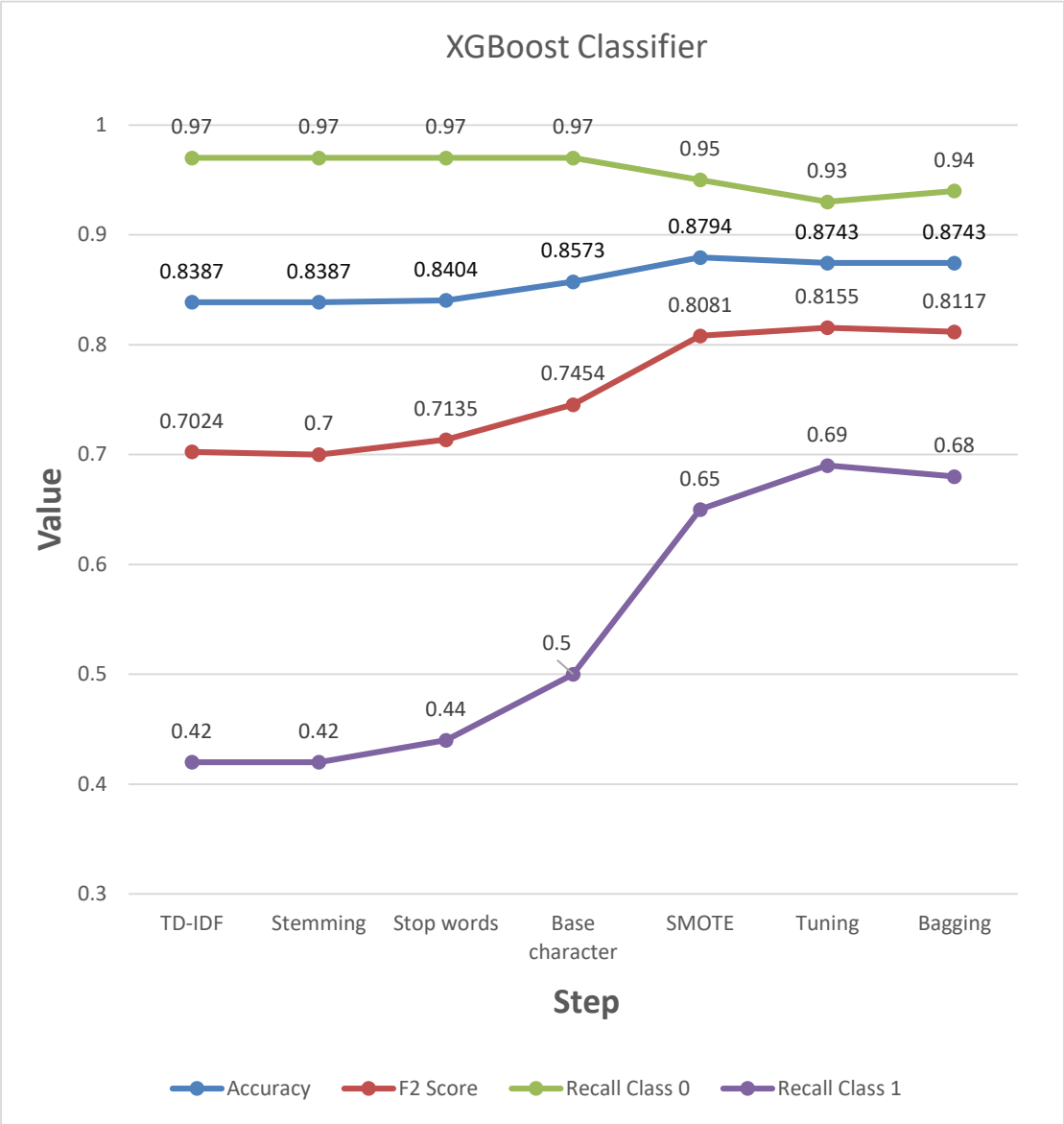


Figure 4.6: XGBoost classifier performance

4.3 Result comparison with baseline

Finally pipeline of classifying racism text was reached to very stable level, which is very good comparing to the baseline results. Table 4.17 compares the baseline and pipeline results.

Table 4.17: Result comparison with baseline

Experiment	Accuracy	Precision	Recall
Baseline	0.5760	1.00	0.138
Pipeline	0.8794	0.83	0.850
Baseline experiment 1 (tested with baseline dataset)	0.7172	0.72	0.690
Baseline experiment 2 (trained and tested the pipeline with baseline dataset)	0.8500	0.86	0.815

According to Table 4.17, baseline experiment shows an accuracy of 0.57, which is relatively low compare to other performance measures. Pipeline on the other hand scored very high accuracy, recall compare to baseline results. Pipeline has been able to reach 0.85 which is a very good level when comparing to baseline score of 0.138. This shows that pipeline easily outperforms the baseline results. However when trained pipeline tested with unseen data set which is whole baseline dataset, it showed some good level of results, compared to baseline. These results also higher than the baseline results. This means pipeline classify unseen data very well especially better than the baseline. As results implies, when pipeline trained with baseline dataset and tested with testing partition, it has given better results than the baseline. This proves that pipeline is much effective and accurate than the baseline even when using same dataset.

CHAPTER 5 – DISCUSSION

This chapter intends to discuss the abstract summary of the study and the final outcomes of this study. Also, it reviews the future research areas that can be done based on this study.

5.1 Summary

Many of the studies have been focused on identifying racism on text based sources. Nowadays social media create more contents than other text sources, therefore studies have been more focusing on social media than the other text sources. Motivation for this study was that social media spread wrong and racism provoking information and no one has the control of limiting these unnecessary information. Building a text classifier which can help to identify racist texts written in Sinhala would be a solution for this and selected that solution as the main objective.

After reviewing the similar studies, necessary results and findings were documented and scope of the study was carefully defined. With the help of the similar studies, four main classifiers were selected for the Sinhala text classification. In addition to those, another two boosting classifiers were also selected. The idea was to build an optimized pipeline from all these classifiers while understanding performance levels of each classifier.

UTF-8 formatted Sinhala text data was gathered from social media and other text sources. This data set was formatted and preprocessed. After preprocessing was finished, annotation was performed by three annotators. Final annotated corpus was selected as the dataset to be fed to pipeline and stemming was performed as the first step in pipeline. After stemming stop words were removed and each character of a word converted to basic character of Sinhala language.

SMOTE oversampling technique was applied due to low data was present in racism category. Bag of Words (BoW) approach was used to create the features and all the data was fed into TF-IDF vectorizer to perform this action. Six classifiers were used for the classification and after observing results, hyperparameter tuning was done for each classifier to achieve the best output and bagging was performed as next step using each classifier.

After each of these major steps, results were carefully monitored and best performing classifiers were selected as the base estimators for voting classifier. These best classifiers were random forest, naïve bayes and XGBoost. As the final step of the pipeline voting classifiers results were evaluated. All the results have been listed in chapter 4. After analyzing all the results and as the final result, voting classifier was identified as the best classifier for the classifying racist texts written in Sinhala language.

5.2 Findings

After observing the results there were important findings which need to be discussed further. Following sub sections discuss the major findings after each major steps.

5.2.1 Apply TF-IDF vectorizer

First result set was observed after applying tf-idf vectorizer which generated the feature vector. All classifiers were trained with this feature vector and stemming or stop words removal was not performed at this stage. By observing the results of this stage, it was easy to identify that ada boost was the best performing classifier comparing to other classifiers results. The reason for this can be the ensemble techniques are minimizing the errors and perform better in many situations while other classifiers fail to achieve that level. Ada boost normally use decision trees as the base estimator, therefore an assumption can be made that decision trees give the best support to increase the accuracy at this stage.

While studying the similar studies, many of the studies used SVM and have achieved the best results. But there was no study for compare boosting techniques with SVM. In this results Ada boost classifier has outperformed the SVM.Logistic Regression and Naïve bayes were not performed as expected and Logistic regression achieved the lowest F2 scores. The reason for this can be lack of data available to train and test. Recall and the TN (True Negative) values were also at the lowest .This can be caused by low training data for one class and classifiers do not have the capability to classify between classes. Unlike ensemble techniques Logistic regression and naïve bayes do not use large number of estimators to minimize errors and maximize the accuracy.

5.2.2 Stemming

As results implies after stemming the texts, performance of each classifier was improved. An effective stemmer convert derived words to one word, hence feature count can be reduced by stemming and the words which are having same meanings can be identified as a one feature. This helps feature vector to become more compact and meaningful. Sinhala language use many derived words, which are derived from its base word. By observing the results after stemming, it becomes clear that stemming improves the classification results and can conclude that stemming is a major step in pre-process when it comes to Sinhala text classification. This step made some huge impact on some classifiers such as random forest, but made average impact on some classifiers like SVM and ada boost. This implies that some classifiers are sensitive to stemming than the others.

5.2.3 Removing stop words

Removing stop words improved the all the classifiers performance. All the classifiers were very responsive to this change and the performance increase also somewhat same in all the classifiers. This proves the facts that removing stop words made an impact in feature vector and it affects all the classifiers in a similar way, therefore stop word removal in Sinhala language text classification is very important. Naive bayes and logistic regression also were able to gain a noticeable performance lift. Even though they are underperforming before this stage, this step improved their performances. It is clear that even though some classifiers not performing well due to lack of data, preprocessing steps like stop word removal helps to increase the performance. After analyzing these results, it is apparent that stop words removal is a key step for text classification and it improves the performance of many classifiers.

5.2.4 Text convert to base character

Conversion of word characters to base character is reduce the complexity of the Sinhala language and it helps to reduce redundant features from feature vector. After performing this step only ada boost classifier was decreased its performance. The reason of decreasing the performance of this classifier is that feature vector might not got rich as expected and results might include some error.

Also converting to base character might change the weights assign for some racist feature words. In other words this particular step might have changed the features distribution in feature space.

5.2.5 SMOTE oversample

After applying this oversampling technique training dataset size got increased and that gave a chance to classifiers to train with many data points. This step caused a major impact on many classifiers. Some classifiers like naïve bayes was able to gain a huge performance lift from this step but some classifiers gain was at an average level.

By analyzing these results, it is easier to observe that class imbalance or lack of data in classes can reduce the performance of classifiers and balancing the dataset can help classifiers to train with more data points and improve the models. In addition to that results show that there are some classifiers which are showing average response to this action while some are showing great results. The reason for this can be that newly generated instances and correlation between them are considered by the classification algorithm in different ways.

5.2.6 N-grams

N-gram is another feature vector generation technique and some of the studies have suggested n-grams as a best technique to use when it comes to text classification. Surprisingly n-gram (Bi-grams in this study) technique was failed in this study since the results were at a very low level. This can be caused by having a low powerful feature vector which contain features that are not occurring much in other texts and does not have a direct impact on final result. N-grams consider all combinations of words of a sentence to generate the feature vector. Sinhala language is full of rich words and one word can have many different meanings.

One sentence can be written in many ways and writing style can be different from person to person. If there is low correlation between words n-grams normally fails. This can be the reason for this to fail. As discussed previously, Sinhala language has many stop words, but sometimes it does not act as a stop word. By removing stop words can revert or destroy the meaning.

After removing stop words and while considering bigrams, the important observation was that the generated features are not rich, in other words generated features not likely to be shown in the text again. This can be led to low n-gram count and became feature vector useless. Therefore N-grams feature generation technique was rejected by considering the results.

5.2.7 fastText

fastText is a library for learning of word embeddings it is capable of obtaining vector representations for words. fastText library was used for create sentence vectors in this study and feature vector was created. This feature vector was used to train and test the classifiers. As for the results given by fastText feature vector, it is clear that classifiers have achieved good results. Even though fastText has given good results, those results were not the best results at the first stage. At this stage TF-IDF, N-grams and fastText were tested with same data set and results of all the classifiers were evaluated. After evaluation it was clear that even though fastText was able to gain better results with two classifiers, classifiers with TF-IDF were able to achieve better results from four classifiers in this stage. Therefore TF-IDF was selected to proceed with this study. fastText contains pre-trained models for each language. These trained models can be include many kind of words and sentences, but may be it does not have much training data which contains racism texts in Sinhala language. As a result model may not have given a rich output with racism context in Sinhala language. This can be the reason for fastText is underperforming at this stage.

5.2.8 Hyperparameter tuning

Hyperparameter tuning is a technique to find the best parameters of a classifier, which are help model to perform at its best level. In tuning, the main object is to improve the model performance according to specific problem. In some cases this improvement can be the accuracy, while recall is a key improvement in another study. In this study all the classifiers tuned to get the best recall of class 1 and best f2 score. Results imply that the tuning improves all classifiers performance. Hence hyperparameter tuning is essential task to be performed in any classification problem. Also necessary parameters should be selected according to classification problem.

5.2.9 Apply bagging

As results suggests, all classifiers were able to increase their performance measures except XGBoost. XGBoost use boosting ensemble technique in an optimized way and bagging process might not support that optimization very well or it might decrease its performance. This performance decrease is a very minor by scale. Bagging use multiple estimators to get the final output. Most accurate models can have a better results since many accurate estimators can point the final output to right direction. Bagging reduce errors since individual estimator's error does not affect other estimators. Low accurate or low performing estimators cannot improve performance by bagging. Results also claim this and show that not all classifiers work with bagging.

5.2.10 Apply voting

Final step of the pipeline was the evaluation of voting classifier. This classifier was used with the intention of get the best out of best classifiers, in other words combine the best classifiers to get the best out of best results.

This goal was achieved and as expected and voting classifier was able to show the best classification results of all the results. ROC Curve and Precision-Recall graph also describes this fact. Voting classifier has the highest area under curve in Precision-Recall graph. Hence voting classifier was selected as the final and best classifier of this study.

Voting classifier was made with the best three classifiers selected from previous steps. As described earlier, voting classifier was fed with these three main classifiers, which were random forest, naïve bayes and XGBoost. The reason for this was the voting classifier's performance highly based on the feeding classifiers vote, therefore quality output should be maintained and fed for voting. Based on the classifiers results, voting classification weighs were adjusted. These weights played a major role when performing the voting step. Since not all classifiers perform at same rate, based on their performance, weights were assigned. As a final thought on voting, it is good to have a voting classifier when there are many good classifiers since voting classifier can combine them and get the best results.

5.3 Evaluation

As describe in the background section, F2 score and precision recall graph were used as the final evaluation criteria. However, other performance measures such as class 1 recall, accuracy and ROC curve also observed to get a better understand about the results. Final results were very clear and there were no conflicts regarding selecting best classifier after voting and selecting best three classifiers before voting.

However, there are some differences in results of precision recall curve and ROC curve comparisons. The main difference is the best performing classifier. As ROC curve suggests best performing classifier is Naïve bayes but as precision recall graph shows, best performing classifier is XGBoost classifier. Also SVM classifier is the lowest performing classifier according to precision recall graph but ROC curve listed SVM as the fourth highest performing classifier. These contradiction again shows that ROC curve and precision recall curves produce difference results in class imbalance problems. Therefore; with help of other performance measures such as accuracy and class1 recall, final output was selected as the precision recall graph scores.

5.4 Wrong classifications

After analyzing the classified comments, some interesting facts were identified. Following Table 5.1 describes expected and predicted labels of some comments. Below few comments describe common problems which were identified during the error analysis. When analyzing the results, the first problem identified was some offensive words were taken as racist when perform the classification, therefore some sentences which include offensive words were classified as racist.

The reason for this is, most of the racist comments contains offensive words; therefore a normal sentence with offensive word can be identify as racist since this experiment uses BoW technique. As described in the table, first sentence predicted as non-racist. The reason for this is that comment contains racist word but it is misspelled.

Table 5.1: Wrong classifications

Comment	Expected label	Predicted label	Reason
පර හම්බයො	Racist	Non Racist	Wrong spelling, “හම්බයො”. It should be “හම්බයො”
අනිවාර්යයෙන් හම්බ කරලා කාලා බීලා ආතල් එකේ ඉන්න ඕන සැපේ	Non Racist	Racist	The word “හම්බ” has two meanings. In this context it gives the meaning of “earn”
මුස්ලිම් ජනගහනයෙන් ඉක්මවූ විට ඔවුන් නොයෙකුත් හේතූන් සඳහා චෝදනාකරමින් නීතිය අතට ගැනීමට උත්සහ කරති	Racist	Non Racist	Deep meaning sentences
ෂරි ෂරි නානා	Non Racist	Racist	Word “නානා” or “ෂරි” considered as racist word.
තම්බි යේක ලස්සන පොටෝ එක තමා ෂරියා	Non Racist	Racist	“තම්බි”, “යේක”, “ෂරියා” words considered as racist words in other comments

Misspelled words are one of the major problem in Sinhala text classification. The second comment contains an offensive word but in that context it means another meaning. It is clear that classifiers cannot clearly maps the meaning of the sentence, especially when use the BoW technique.

The third comment is having some deep meaning and that comment does not contain any offensive words as well. This kind of comments are hard to classify by a classifier. Fourth and fifth comments contain words which are not racist but most of the racist comments contain those words; therefore, these words might identify as racist words by classifiers.

These common problems were identified by analyzing the results and future work also can be built to overcome these problems.

5.5 Baseline comparison

As described in previous section, baseline is outperformed by pipeline. This is clearly proves that these techniques give best even with the same data set which is used for baseline experiment. Pipeline is a combination of many techniques and it gives the final output by voting classifier. By using these techniques and rather than using single classifier, this study shows that multiple classifiers capabilities can be combined with this pipeline and it still gives better results than the baseline. This study focus on many areas such as feature generation, preprocessing steps and ensembling. Baseline study did not use some of them and some techniques used very lightly. As an example, number of preprocessing steps can be increased. Stemming and converting to base character steps were not there in baseline study. They have not used any ensembling technique as well. This study has proven those techniques increase the accuracy of the model.

5.5 Limitations

This study mainly focus on the racist text classification on Sinhala language using effective pipeline. When it comes to selecting the best pipeline, it is almost impossible since this study does not apply all the techniques exist in the domain and does not compare all the classifiers and their behaviors. Hence this study only focus on limited techniques based on the scope. The dataset was relatively small and this study also used limited number of classifiers and those were selected by reviewing similar study recommendations. These classifiers were compared by performing eight major different steps. Cause of the classifiers performance increase or decrease was not considered deeply since it is out of the scope of this study.

5.6 Future work

Future studies can be done with the help of the findings and techniques which are used in this study. As discussed in the above section error analysis, this technique has few limitations. Apart from those classification problems, this pipeline only contains limited number of steps. Extending these steps and improving these steps can be improved the output. Feature generation with different techniques and enhancing them with new techniques will be a future work.

CHAPTER 6 – CONCLUSION

This study focused on build a classification pipeline for racism texts written in Sinhala. In order to build an effective pipeline various techniques were used. As the baseline, Sinhala racism text classifier proposed by Dulan S. Dias et al was selected. As the next step, six classifiers were identified as main classifiers to include for this pipeline for the classification part. Bag of Words (BoW) approach was selected to generate features. Data preprocessing was identified as a major step in this pipeline and steps such as stemming, stop word removal and converting to base character were improved the quality of the classification output. SMOTE oversampling technique was used in this study due to class imbalance problem and it gave great results. Hyperparameter tuning was improved each classifiers performance and ensembling techniques also improved the performance of some classifiers. A Voting classifier was used and final output of voting classifier was considered as the best output of the pipeline. This voting classifier was able to outperform baseline and proved it can classify racism texts in an optimum way.

Main conclusion of this study is classification of racism text written in Sinhala language, is possible with BoW technique and all the six classifiers which were used in this study. A better model can be built using these classifiers and voting classifier can be used to get the best output from all these classifiers. Study showed that preprocessing steps boost and improve classification results in Sinhala language. Also this study showed that oversampling techniques can be used effectively with the Sinhala NLP problems. Finally it can be concluded as an effective pipeline for Sinhala racism text classification can be built using the techniques used in this study.

REFERENCES

[1] Pete Burnap and Matthew L. Williams.*Cyber Hate Speech on Twitter: An Application of Machine Classification and Statistical Modeling for Policy and Decision Making*

[2] Njagi Dennis Gitari1, Zhang Zuping, Hanyurwimfura Damien and Jun Long. *A Lexicon-based Approach for Hate Speech Detection*

[3] Giacomo Berardi, Andrea Esuli, Craig Macdonald,Iadh Ounis, Fabrizio Sebastiani. *Semi-Automated Text Classification for Sensitivity Identification*

[4] Irene Kwok and Yuzhou Wang. *Locate the Hate:Detecting Tweets against Blacks*

[5] William Warner and Julia Hirschberg. *Detecting Hate Speech on the World Wide Web*

[6] Shilpa Samaratunge and Sanjana Hattotuwa. *Liking violence, A study of hate speech on Facebook in Sri Lanka.*

[7] Björn Gambäck and Utpal Kumar Sikdar.*Using Convolutional Neural Networks to Classify Hate-Speech*

[8] Dulan S. Dias, Madhushi D. Welikala, Naomal G.J. Dias. *Identifying Racist Social Media Comments in Sinhala Language Using Text Analytics Models with Machine Learning*

[9] Z. Waseem and D. Hovy, "Hateful Symbols or Hateful People? *Predictive Features for Hate Speech Detection on Twitter*", Proc. NAACL Student Res. Work., pp. 88–93, 2016.

[10] St´ephan Tulkens, Lisa Hilte, Elise Lodewyckx, Ben Verhoeven, and Walter Daelemans, *A Dictionary-based Approach to Racism Detection in Dutch Social Media*.

[11] Md Gulzar Hussain, Tamim Al Mahmud, and Waheda Akthar, *An Approach to Detect Abusive Bangla Text*.

[12] Ika Alfina, Rio Mulia, Mohamad Ivan Fanany, and Yudo Ekanata, *Hate Speech Detection in the Indonesian Language: A Dataset and Preliminary Study*.

[13] Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber, *Automated Hate Speech Detection and the Problem of Offensive Language*.

[14] N.D.T. Ruwandika, *Identification of Hate Speech in Social Media*.

[15] Salminen, J., Hopf, M., Chowdhury, S.A. et al, *Developing an online hate classifier for multiple social media platforms*. Hum. Cent. Comput. Inf. Sci. 10, 1 (2020). https://doi.org/10.1186/s13673-019-0205-6

[16] Punyajoy Saha, Binny Mathew, Pawan Goyal, and Animesh Mukherjee, *Hateminers: Detecting Hate speech againstWomen*.

[17] Fabio Del Vigna, Andrea Cimino, Felice Dell'Orletta, Marinella Petrocchi, and Maurizio Tesconi, *Hate me, hate me not: Hate speech detection on Facebook*.

[18] Dimuthu Lakmal, Surangika Ranathunga, Saman Peramuna and Indu Herath, *Word Embedding Evaluation for Sinhala.*

[19] "LogisticRegression" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (Last accessed Jan. 6, 2020).

[20] Dave Sotelo, "Using Bagging and Boosting to Improve Classification Tree Accuracy" medium.com. [Online]. Available: https://towardsdatascience.com/using-bagging-and-boosting-to-improve-classification-tree-accuracy-6d3bb6c95e5b (Last accessed Jan. 8, 2020).

[21] "bagging-boosting" github.io. [Online]. Available: https://swalloow.github.io/bagging-boosting (Last accessed Jan. 8, 2020).

[22] "BaggingClassifier" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html (Last accessed Jan. 9, 2020).

[23] Joseph Rocca, "Ensemble methods: bagging, boosting and stacking" medium.com. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (Last accessed Jan. 9, 2020).

[24] Sunil Ray, "Quick Introduction to Boosting Algorithms in Machine Learning" [Online]. Available: https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning (Last accessed Jan. 9, 2020).

[25] Friedman JH (2001). *Greedy function approximation: a gradient boosting machine.*

[26] Vishal Morde, “XGBoost Algorithm: Long May She Reign!” medium.com. [Online]. Available: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d (Last accessed Jan. 11, 2020).

[27] “Majority voting algorithm” [Online]. Available: https://www.researchgate.net/figure/Majority-voting-algorithm_fig2_324014302 (Last accessed Jan. 15, 2020).

[28] Jason Brownlee, “A Gentle Introduction to the Fbeta-Measure for Machine Learning” machinelearningmastery.com. [Online]. Available: https://machinelearningmastery.com/fbeta-measure-for-machine-learning/ (Last accessed Jan. 17, 2020).

[29] “ROC” [Online]. Available: https://www.statisticshowto.datasciencecentral.com/c-statistic/ (Last accessed Feb. 20, 2020).

[30] “Precision-recall curves” [Online]. Available: https://acutecaretesting.org/en/articles/precision-recall-curves-what-are-they-and-how-are-they-used (Last accessed Feb. 20, 2020).

[31] Takaya Saito and Marc Rehmsmeier. *The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets*

[32] Jesse Davis and Mark Goadrich, 2006. *The relationship between Precision-Recall and ROC curves*

[33] “RandomForestClassifier” scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Last accessed Feb. 24, 2020).

[34] “MultinomialNB” scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html (Last accessed Feb. 24, 2020).

[35] “GridSearchCV” scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (Last accessed Feb. 24, 2020).

[36] “AdaBoostClassifier” scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html (Last accessed Feb. 24, 2020).

[37] “Python API Reference” xgboost.readthedocs.io. [Online]. Available: https://xgboost.readthedocs.io/en/latest/python/python_api.html (Last accessed Feb. 24, 2020).

[38] "VotingClassifier" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html (Last accessed Feb. 24, 2020).

[39] "NLP-What-does-it-mean-Tokenizing" quora.com. [Online]. Available: https://www.quora.com/NLP-What-does-it-mean-Tokenizing (Last accessed Dec. 29, 2019).

[40] "TfidfVectorizer" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Last accessed Dec. 30, 2019).

[41] "SMOTE_explained" rikunert.com. [Online]. Available: http://rikunert.com/SMOTE_explained (Last accessed Jan. 2, 2020).

[42] Haibo He, "Imbalanced Learning: Foundations, Algorithms, and Applications", 2013, pp. 47-48

[43] "SMOTE" imbalanced-learn.readthedocs.io. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html (Last accessed Jan. 2, 2020).

[44] Rohith Gandhi, "Support Vector Machine-Introduction to Machine Learning Algorithms" medium.com. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (Last accessed Jan. 5, 2020).

[45] "SVC" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (Last accessed Jan. 5, 2020).

[46] "RBF SVM parameters" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html (Last accessed Jan. 5, 2020).

[47] "Support Vector Machines" scikit-learn.org. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html# (Last accessed Jan. 5, 2020).

[48] Ayush Pant, "Introduction to Logistic Regression" medium.com. [Online]. Available: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148 (Last accessed Jan. 6, 2020).

[49] Satyajit Kamble, Aditya Joshi. *Hate Speech Detection from Code-mixed Hindi-English Tweets Using Deep Learning Models.*

[50] Miguel Romero, Yannet Interian, Timothy Solberg and Gilmer Valdes. *Training Deep Learning models with small datasets.*

[51] Prashant Kapil, Asif Ekbal and Dipankar Das. *Investigating Deep Learning Approaches for Hate Speech Detection in Social Media.*

[52] Steven Zimmerman, Chris Fox and Udo Kruschwitz. *Improving Hate Speech Detection with Deep Learning Ensembles.*

APPENDICES

[Appendix – I: Literature review summary part 1]

Reference	Domain	Method	Dataset Size	Features	Evaluation
High resource language and single classifier					
[2]	Hate speech detection in web sites.	Rule based classifier	500 paragraphs.	semantic and subjective word features	Precision and Recall slightly lower than 70%.
[3]	Sensitive document identification	SVM	1111 records.	Term frequency with words	Detailed results were described in the research paper.
[4]	Racist content identification in twitter	Naïve Bayes classifier	24582 Tweets.	Unigram	Accuracy of 76%, Error rate of 24%
[5]	Hate Speech on the World Wide Web	SVM	1000 paragraphs.	part-of-speech windows as features	Accuracy 0.910.
[9]	Predictive Features for Hate Speech	Logistic regression	16,914 tweets	character n-gram and word n-gram	F-score 73.93% with character n-grams
[10]	Dictionary-based Approach to Racism Detection in Dutch Social Media	SVM	5759 comments	Their own dictionary and word2vec	F-score as 0.46
Low resource languages					
[8]	Identifying Racist Social Media Comments in Sinhala Language Using Text Analytics Models with Machine Learning	SVM	238 comments with 100 racist comments	N-grams with TF-IDF	Accuracy: 0.576 Precision: 1.000 Recall: 0.138 F1 Score: 0.242
[11]	Detect Abusive Bangla Text	Their own classification algorithm	300 comments	Unigram, bigram and trigram	Precision of hate class: 0.83 Recall: 0.67.
[12]	Hate Speech Detection in the Indonesian Language	Random Forest, Bayesian Logistic Regression, Naïve bayes, SVM, , Decision Tree	520 tweets	Character n-gram, Word n-gram, and negative sentiment	F-score of 82.6% with Quadra-gram and RFDT

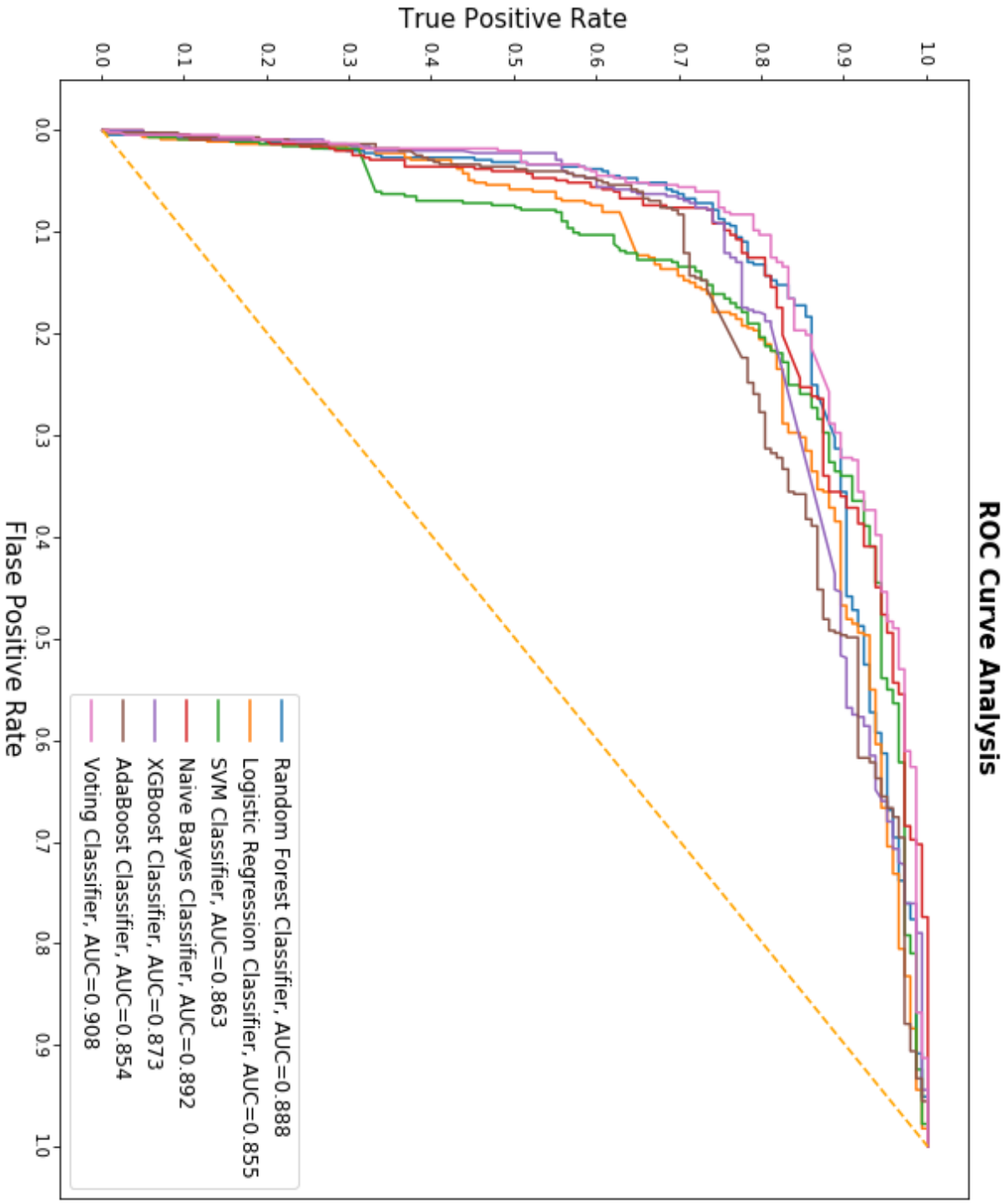
[Appendix – II: Literature review summary part 2]

Reference	Domain	Method	Dataset Size	Features	Evaluation
High resource language and multiple classifiers					
[1]	Hate speech identification in tweets.	Bayesian Logistic Regression, Random Forest, Decision Trees, SVM	1901 tweets with 222 offensive tweets.	N-grams	Using Voted meta classifier, Precision – 0.83, Recall – 0.60 F measure – 0.77
[13]	Automated Hate Speech Detection and the Problem of Offensive Language	Logistic regression, Naive Bayes, Decision trees, Random forests, Linear SVM	25k tweets	Unigram, bigram and trigram. each weighted by its TF-IDF	Precision:0.91 Recall: 0.90 F1-score: 0.90 With LR and SVM
[14]	Identification of Hate Speech in Social Media	Naïve Bayes, Logistic Regression, SVM, Decision Tree, K-Means clustering	1500 comments	TF-IDF	F-score: 0.719
[15]	Hate classifier for multiple social media platforms	Logistic Regression, Naïve Bayes, Support Vector Machines, XGBoost, Neural Networks	197,566 comments	TF-IDF, BoW, Word2Vec, BERT combinations.	F-score: 0.92 with XGBoost
[16]	Detecting Hate speech against Women	Logistic Regression, XGBoost and CatBoost	5000 tweets	Sentence embeddings, TF-IDF vector and Bag of words vector (BoWV)	Accuracy: 0.704 with Logistic Regression classifier.

[Appendix – III: Literature review summary part 3]

Reference	Domain	Method	Dataset Size	Features	Evaluation
Deep learning techniques					
[7]	Hate-Speech classification using Convolutional Neural Networks	Convolutional Neural Networks	6,555 tweets	continuous-bags-of-words (CBoW) and skip-grams	Precision of 86.61% Recall of 70.42% F-score of 77.38%.
[17]	Hate speech detection on Facebook	Recurrent Neural Networks and SVM classifier.	17,567 Facebook comments	word2vec	SVM achieved Accuracy: 64.61% and F-score: 0.256 for strong hate class.
[49]	Hate Speech Detection from Code-mixed Hindi-English Tweets Using Deep Learning Models	CNN-1D, LSTM, BiLSTM	Dataset1: 255,309 and Dataset2: 3849 tweets	Domain specific word embeddings	Precision of 83.34% Recall of 78.51% F-score of 80.85%. With CNN-1D.
[50]	Training Deep Learning models with small datasets	CNN	14,863. (Detailed dataset were described in paper)	CNN as feature extractor.	Detailed results were described in the paper.
[51]	Investigating Deep Learning Approaches for Hate Speech Detection in Social Media	CNN, LSTM, BiLSTM, and Character-CNN	15,476 tweets	W2V, GloVe and fastText	Detailed results were described in the paper.
[52]	Improving Hate Speech Detection with Deep Learning Ensembles	CNN.	16,883 comments	Their own word embedding technique	Detailed results were described in the paper.

[Appendix – IV: ROC curves comparison]



[Appendix – V: Precision recall curves comparison]