Data Mining Project (CSE-362)
Report

Hate Speech / Toxic Comment Detection

Report submitted in fulfillment of the requirements
for the Data Mining Course of

Third Year B. Tech.
in
Computer Science and Engineering

Submitted by

Roll No	Names of Students
18074019	Harshit Agrawal
18075068	Ashish Kumar
18075070	Sachin Srivastava

Under the guidance of
Dr. Bhaskar Biswas

Department of Computer Science and Engineering
Indian Institute of Technology (BHU) Varanasi
Varanasi, Uttar Pradesh, India – 221005

Semester V - Nov, 2020

Dedicated to

Our beloved parents, teachers,

Our laptops and Varanasi

Declaration

We certify that

1. The work contained in this report is original and has been done by ourself and the general supervision of our supervisor.

2. The work has not been submitted for any project.

3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: November 25, 2020

Harshit Agrawal (18074019 - IDD)
Ashish Kumar (18075068 - B.Tech.)
Sachin Srivastava (18075070 - B.Tech.)

Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Department of Computer Science and Engineering
Indian Institute of Technology (BHU) Varanasi
Varanasi, INDIA 221005.

Certificate

This is to certify that the work contained in this report entitled **"Hate Speech / Toxic Comment Detection"** *being submitted by* **Harshit Agrawal (18074019)**, **Ashish Kumar (18075068)** *and* **Sachin Srivastava (18075070)**, *carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of my supervision.*

Place: IIT (BHU) Varanasi
Date: November 25, 2020

Dr. Bhaskar Biswas
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

It is a great pleasure for us to express respect and deep sense of gratitude to our supervisor Dr. Bhaskar Biswas, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, as a convener of this course for his wisdom, vision, expertise, guidance, enthusiastic involvement and persistent encouragement during the planning and development of this work.

We also gratefully acknowledge his painstaking efforts in thoroughly going through and improving the manuscripts without which this work could not have been completed. We are also highly obliged to Prof. Pramod Kumar Jain, Director, Indian Institute of Technology (BHU) Varanasi, and Prof. Rajeev Srivastava, Head of Department, CSE for providing all the facilities, help and encouragement for carrying out this project work. We are also obliged to our parents for their moral support, love, encouragement and blessings to complete this task.

Finally, we are indebted and grateful to the Almighty for helping us in this endeavor.

Place: IIT (BHU) Varanasi
Date: November 25, 2020

Harshit Agrawal
Ashish Kumar
Sachin Srivastava

Contents

List of Figures

List of Tables

Data Mining Project on Hate Speech / Toxic Comment Detection

Harshit Agrawal, harshitagrawal.cse18@itbhu.ac.in
Ashish Kumar, ashishkumar.cse18@itbhu.ac.in
Sachin Srivastava, ssrivastava.cse18@itbhu.ac.in

under the guidance of Dr. Bhaskar Biswas

November 25, 2020

Abstract

With the push towards the decline in data rates and growth of telecommunication networks in the last few years, there has been a huge spike in the number of internet users. Penetration of high bandwidth connection into the remotest part of the world has enabled quite a few users to come up online and express their opinions and thoughts. However, the another side of coin is that it has lead to increase in instances of hate speech and bullying which not only holds back certain users from opening up freely with ingenious ideas but also spreads negativity, depression, communal/racial hatred and at times can trigger riots. So it becomes imperative for social media ventures to ensure that their platform is free from toxicity to encourage free flow of information and opinions.

1 Introduction

1.1 Overview

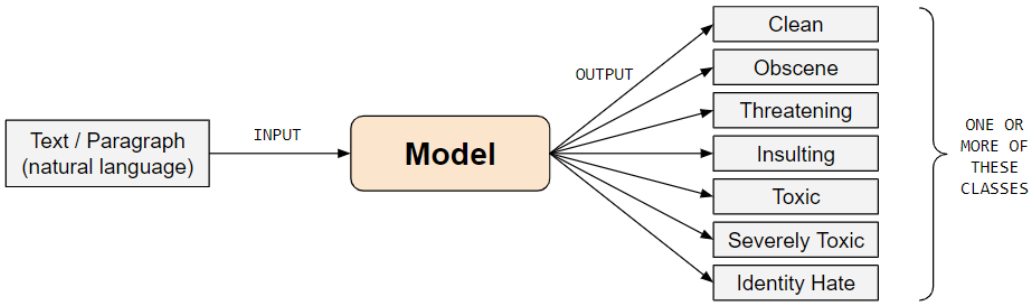


Figure 1: Introduction

Social networking sites have evolved to become an effective medium of communication all over the world. People can freely share their opinions and ideas with anyone. However, many people are found spreading verbal violence in the form of hate speeches and toxic comments. Thus, it has become a serious challenge for the social networking sites to control the spread of toxic comments. The task of hate speech or toxic comment detection can be posed as a multi-class and multi-label classification task. Companies are building models to automatically flag such content on the sites.

In this paper, we take a collection of posts obtained from various social media sites and build models to classify those posts into six categories namely, toxic, severe toxic, threat, identity hate, obscene and insult. The advantage of this type of data is that these comments represent a true sample of the content present on the social media sites. We began by performing analysis and visualization of the dataset. Since the dataset contained the real comments posted on social media, the data contains noise. We had to preprocess the data to remove any outliers or noise that were present in the dataset. We initially tested the performance of classical models namely, support vector machines, and logistic regression on this task. We then experimented with newer techniques like tree based ensembling methods. We further tested the performance of more sophisticated models, like, recurrent neural networks on this task. We then applied pretrained embeddings, namely, word2vec, fasttext, and glove in our model and performed the classification. We compared the performance of all the models using the mean AUC ROC score as the performance metric.

1.2 Report Structure

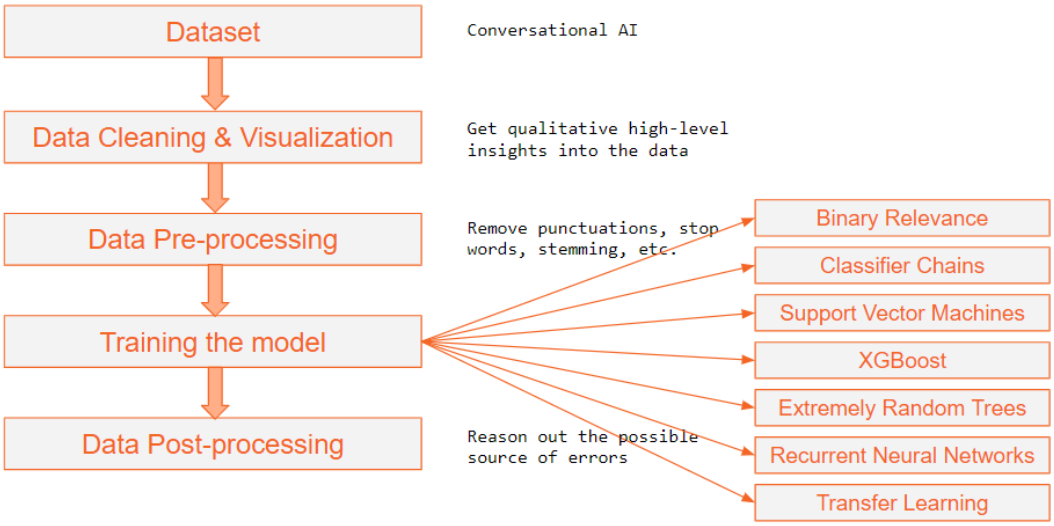


Figure 2: Flowchart

The structure of the report is as follows: In Section 1, we describe the problem statement, and give the description of the dataset. We describe the visualization and analysis of the data in section 2. In Section 3, we describe the various methods used in cleaning and preprocessing the dataset. We explain the performance of various models on this task in section 4. In Section 5, we give a comparative analysis of the results obtained from the models. Finally, we give the conclusion of our report in section 6.

1.3 Problem Description

We are given a set of tweets/comments/posts which may have been collected from a variety of websites including facebook, twitter, instagram, etc. The task involves detecting and flagging the comments which may involve display of hate or vulgarity. In particular the model classifies each of the toxic comments into one of the six categories- toxic, severely toxic, threatening, insulting, obscene, identity hate. In practice, if the comment is not toxic, then it is labeled as clean which is added as an additional class label.

The problem involves processing of comments which are written in Natural Language, hence belong to the category of NLP problems. Further the source of data is internet open platforms and social media websites so the comments are from the general public and hence are highly noisy, unclean and unstructured. This necessitates rigorous text cleaning and preprocessing pipeline to obtain useful information and to structure the data.

As we can see the above description this problem is a multiclass classification as well as multilabel classification problem.

- **Multiclass classification problem**: This type of classification involves putting each data point or example into one of several possible categories as opposed to a binary classification problem in which each example can be classified into only one of the two categories.
- **Multilabel classification problem**: This type of classification involves examples such that each example can belong to multiple categories and not necessarily only one category. A multi label classification problem can be viewed as a generalised version of the multiclass classification problem in which there is no restriction over how many classes can a training example belong to.

Figure 3: Multi-class as well as a multi-label classification problem

Now according to our problem description, a comment can belong to any of the seven categories. That's why the problem is a multiclass classification problem. Secondly we note that a comment may be offensive in multiple ways. A comment which is toxic may also be obscene. So a comment can belong to several categories all at the same time and hence, it's a mutlilabel classification problem too.

One possible variation of our problem is to classify each comment as just being toxic or not which pretty much reduced it to a binary classification problem. But

this is not a permissible reduction as it severely restricts the domain of application of our model. We may, in practice, desire a website in which some degree of insult (say) is allowed but identity hate is not allowed at all. If we restrict our model to binary classifiers, then it will not be usable in these cases. We therefore explore some other problem transformation methods described in forthcoming sections.

1.4 Dataset Description

```
train.head(10)
```

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0

Figure 4: Dataset

The dataset for the problem has been made publicly available by Conversational AI. In the snapshot of the dataset we can see it has approximately 160k training examples. Corresponding to each example we have a set of labels and each label can take two values 0 or 1 depending on whether that particular comment should belong to the class label or not.

The preparation of the dataset has been done using crowdsourced annotation platforms. This means different examples have been labeled by different users across the internet and hence there is an element of subjectivity in labeling. What may be offensive for me may not be offensive for someone else! This makes the dataset particularly vulnerable to judgement bias.

On further exploring the dataset we realise that toxicity is influenced by some other factors too. One such example is the way a comment is rendered. If rendered in a particular way on a particular screen width, the comment takes the form of offensive symbols which are obviously toxic. But if the annotator had been using

some other screen size then it won't be detected by him. In such cases he may think the comment is spam, but it will be classified as clean nonetheless.

As another example, we have some comments which are very long and detailed, possibly expressing someone's opinion on some topic. Some of these comments have been classified as toxic without any reason.

Thus it is impossible to classify 100% examples correctly as some labels may not be correct themselves.

1.5 Applications

1. **Social Networking**: With increase in user base of the popular sites it has become practically infeasible for humans to read every post and comment to decide what is toxic and what not and take down the offensive posts. Machine learning models can be put to use by social networking websites to flag the offensive posts. The process of taking down may happen in a two step process in which the model first flags the toxic comment and a human only analyses the flagged content and decides if the post should be removed from the website. This certainly makes the task a lot more easier than to analyse every post by human only.

2. **Online meetings / webinars**: The COVID era has seen a dramatic increase in the meetings and classes being conducted over the internet platforms. Unlike physical meetings where it is highly unlikely that someone might abuse the speaker, it is quite common to see people abusing on the online meeting platform as it gives them assurance of being 'hidden' by using anonymous accounts.

3. **Chatbot Training**: Chatbots like Google Assistant, Alexa and Bixby can be put to use in a wide range of applications. These chatbots collect the user data which are then again used to train them and customise to user specific data. But this also opens the possibility of abusing the system by training them on toxic data and in turn making them learn and replicate the insulting remarks. Toxic comment analysis can be used to filter out any toxic data input by the user thereby preventing the model from learning from toxic data.

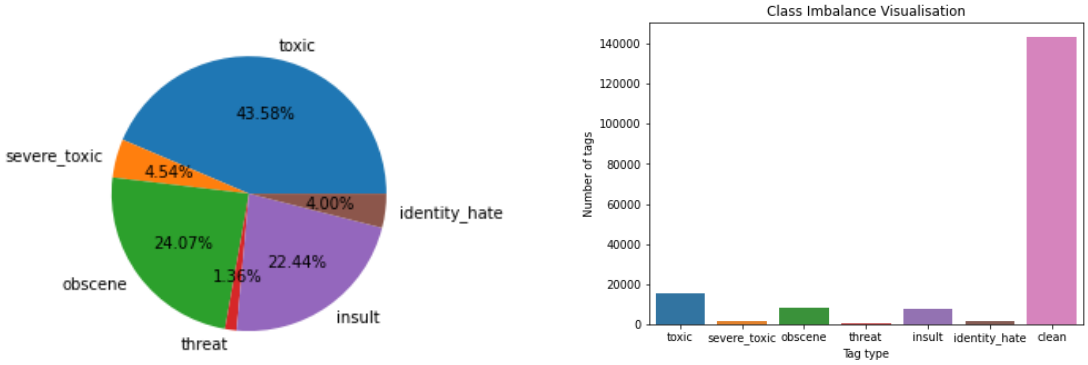
4. **Threatening messages**: If the model perfects the art of detecting offensive comments, then it is possible to directly report threatening messages to

concerned authorities taking quick and timely action. As an example, if a kidnapper demands a ransom from someone, this message can be automatically reported to the Police triggering corrective measures.

2 Data Visualization

Data Visualization is an important part of Data Mining. It helps us get visual insights about the dataset, such as some striking features or patterns in the dataset, which can help us choose the appropriate machine learning algorithms to apply.

We first plotted a pie chart, showing the relative amount of hate tags present in the dataset, and found a massive class imbalance. "Toxic" tags were the highest, whereas tags labeled with "threat" had the lowest count. Also, plotting a bar graph after adding a column for "clean" tags, it was found that a significantly large amount of comments were clean.



(a) Pie chart (b) Bar graph

Figure 5: Relative amount of hate tags present in the dataset

2.1 Visualization of the length of comments

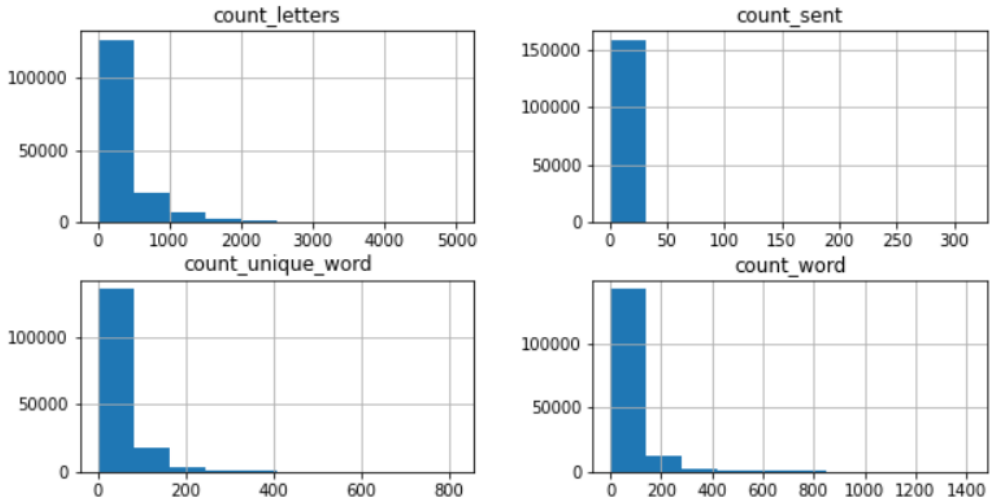
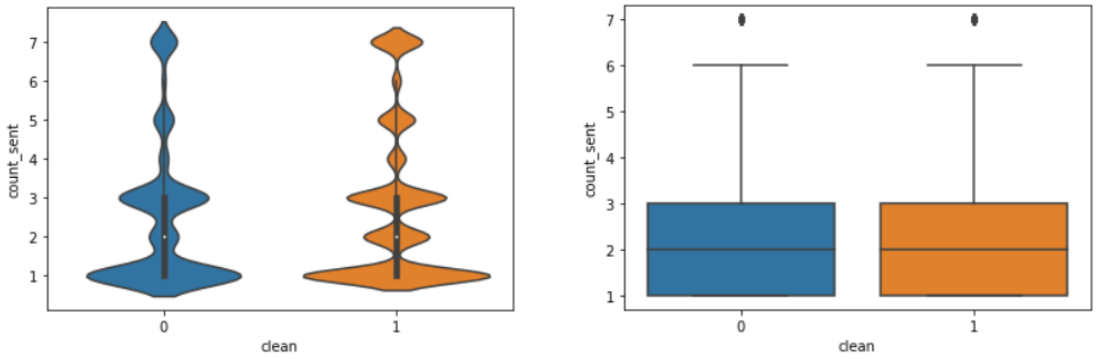


Figure 6: A lot of comments have very less count of sentences, words, or letters

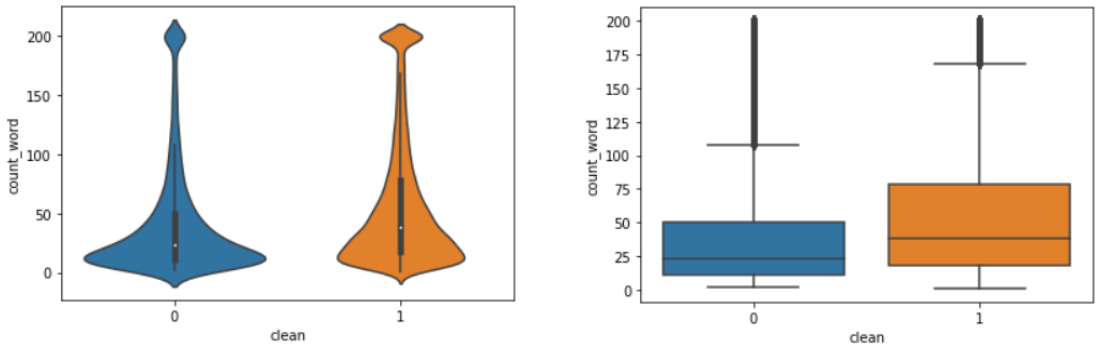
2.2 Correlation between length of comments and toxicity



(a) Violin plot

(b) Box plot

Figure 7: There is not much correlation between the count of sentences and the comment's toxicity



(a) Violin plot (b) Box plot

Figure 8: There is a slight correlation between the count of words and toxicity - less is the count of the word, the more is the comment's toxicity.

2.3 Correlation between spam comments and toxicity

A cursory look at the comments revealed that a lot of comments were spam. In order to classify the spam comments, as an approximate measure, we classified comments with a lot of repeated words, and very less percentage of unique words as spam.

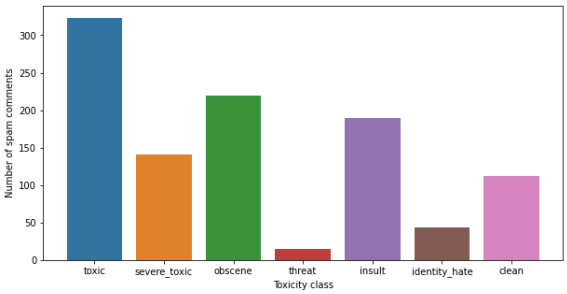


Figure 9: Bar plot - The spam comments (i.e. the comments with less percentage of unique words) increase the chance of toxicity.

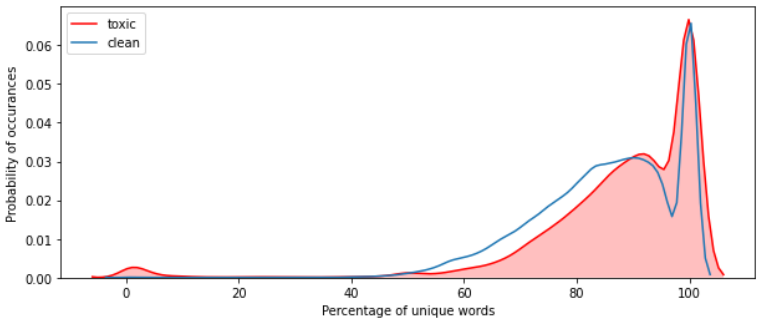


Figure 10: KDE plot - less percentage of unique words do not decrease the chance of toxicity.

2.4 Visualization of the Multi-class classification

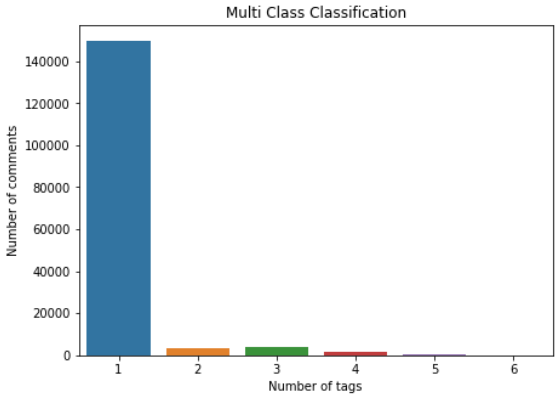


Figure 11: The bar plot shows that a large amount of comments have only a single tag. However, there do exist the comments which have 5 or 6 hate tags.

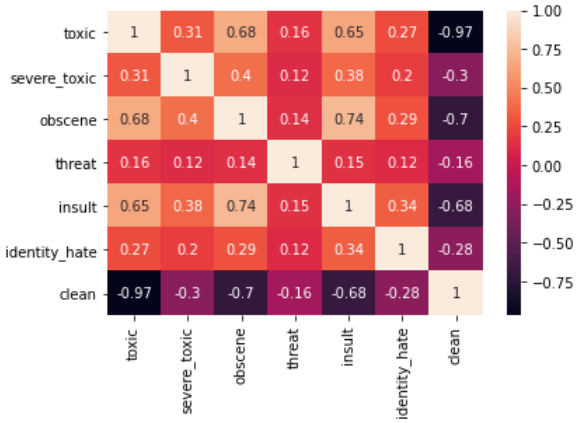


Figure 12: The heatmap of the correlation between the tags show a negative correlation between the clean comments and the toxic comments. Also, a strong correlation can be observed between certain tags, such as toxic-obscene, and toxic-insult.

3 Data Cleaning and Preprocessing

During this stage of the Data Mining task, we clean the data, so as to remove any outliers or noise that may be present in the dataset. If any data is missing, then either that data needs to be filled (manually or automatically) or that tuple needs to be ignored completely. Fortunately, we checked for missing values in the dataset and found that none of the values were missing.

Since the dataset contains the comment text from the internet, it may also consist of the URLs/hyperlinks, IP Addresses, user handles, trailing newline marks, etc. which aren't of any use in classifying the text. Therefore, we also removed these from the dataset. Lastly, as the case of the word does not matter, we reduced all the letters to the lowercase.

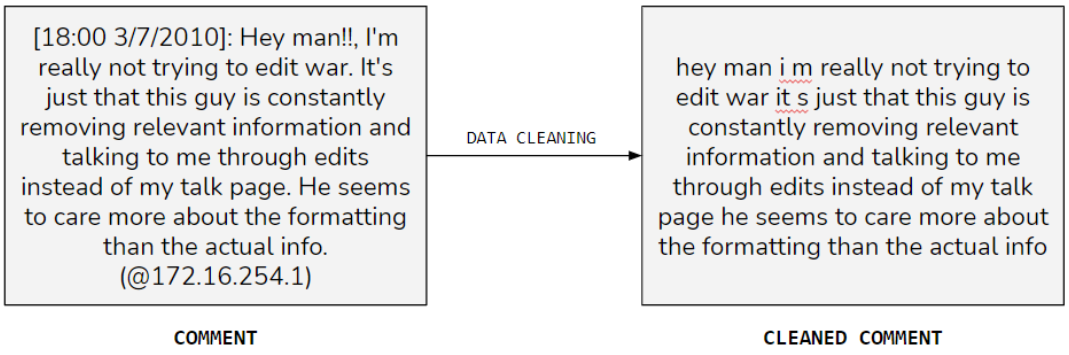


Figure 13: Data Cleaning

Then, the next step involves preprocessing the cleaned data, so that the data in the natural language form can be converted into machine readable form, on which the machine learning algorithms can be applied to get the result. Data Preprocessing mainly consists of two steps - removing the stop words, and either stemming or lemmatization of the comment text.

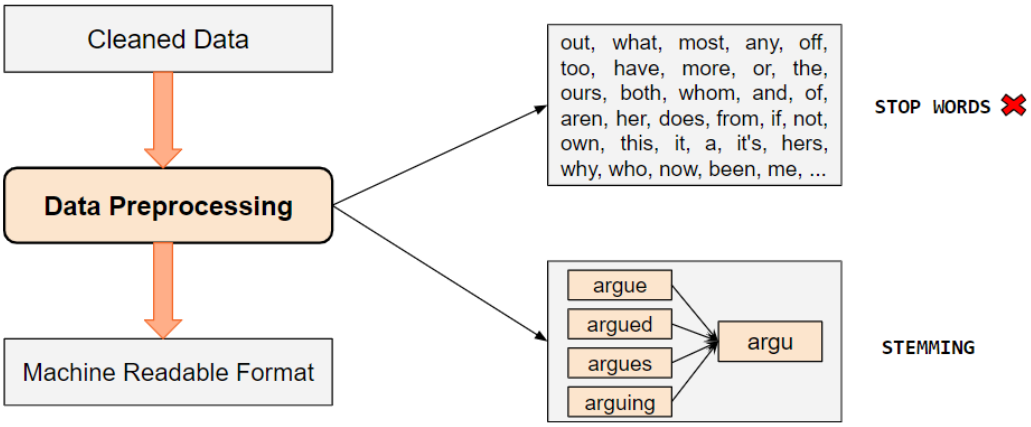


Figure 14: Data Preprocessing FlowChart

3.1 Removing the stop words

The stop words are those words in the comment text, which don't add any physical meaning to the comment, but are generally used as connectors in the natural language in order to make the text grammatically correct. They generally comprise of the auxiliary verbs like is, am, are, was, etc., interrogative pronouns like who, was, were, etc., or any other commonly used words used in the english language. They also have punctuations, that may not be useful. We removed the stop words from the comment text using the NLTK stopword corpus.

3.2 Stemming

In the natural language, we find that words have different derived forms, which either convey different tenses or different points of view. However, as these words are only grammatically significant, they must be converted into the same form, as they mean the same. To do this conversion, stemming or lemmatization is used.

Stemming involves converting a word from its normal form to its base stem. Here, the base stem comprises the set of characters which are required to form a word and its derivatives. In the case of stemming, the base stem need not be a correct english word. Thus, stemming helps in reducing the vocabulary of words in the comment text. E.g.: If the words arguing, argued, argues, argue are present, all of them are

reduced to ‘argu’ during the stemming process. Here, the base stem needs to be chosen correctly so that it does not refer to any other word with a different meaning. E.g. for happier, happiness and happy - ‘happ’ shall not be taken as the base stem, as it may mean ‘happen’. Instead, ‘happi’ shall be chosen as the base stem.

For stemming, we used the Porter Stemmer algorithm of the NLTK package, which consists of a set of rules to be applied on a given word to convert into its stem form.

3.3 Lemmatization

This process also converts the word into its root form, but it converts the word into its base lemma, where lemma shall be a correct word of the english language. Therefore, lemmatization is supposed to give a better performance over stemming, since it also preserves the parts of speech, by also considering the context of the sentence and the neighbouring words present. However, owing to this reason, it is also a comparatively slower process. E.g. for the case of arguing, argued, argues and argue, they will be reduced to ‘argue’ instead of ‘argu’. Also, this helps in reducing ‘good’, ‘better’ and ‘best’ to the same lemma, for which stemming might produce different root stems.

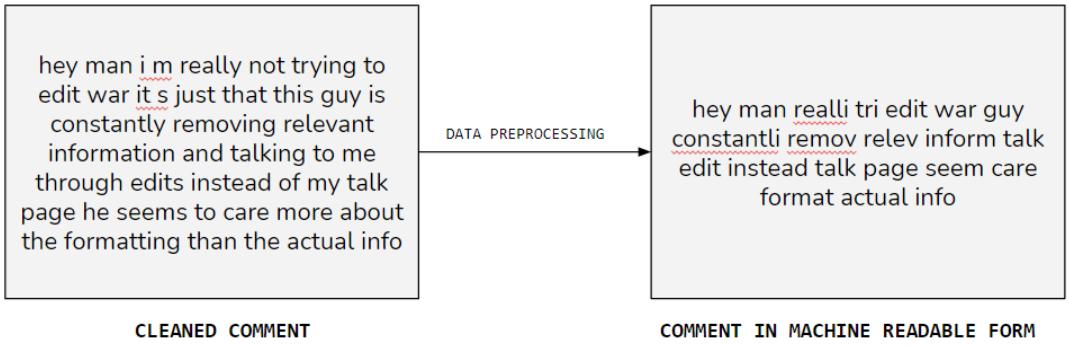


Figure 15: Data Preprocessing

3.4 Encoding Methodology

The preprocessed data is still in natural language form, which is not understandable by the machine. So, it needs to be broken down into tokens or words, and these tokens need to be encoded in the form of integer or floating point numbers, to be used by machine learning algorithms for prediction. This complete process of converting the preprocessed text in natural language form into encoded form is called vectorization or feature extraction. [2]

The following methodologies are typically used for text feature extraction:

1. **Tokenizing**: Splitting text in the form of tokens and assigning integer or floating-point value to each token.
2. **Counting**: counting the frequency or the occurrence of token in the document.
3. **Normalizing**: Penalizing the tokens or reducing the impact of those tokens that occur in most of the documents.

Two popular techniques used for vectorization are Count Vectorizer and TF-IDF Vectorizer.

3.4.1 Count Vectorizer

This vectorizer encodes the text by scanning through every comment text and building a vocabulary of words, containing the words from all the comments. Then, it finds the count of words present in the vocabulary, for each comment. Finally, it creates a vector of dimension $|V| \times |D|$ representing the encoded form of the comment texts, where $|V|$ denotes the number of words in the vocabulary, and $|D|$ denotes the number of comment texts i.e. the number of documents.

Though this is the simplest technique to encode text, it suffers from drawbacks. It finds the count of a term locally in a comment, and does not consider the count of the term in the whole list of comments. The words that occur in most of the documents are not much important, therefore they must be penalized, which is not done by Count Vectorizer. This problem is solved by the TF-IDF Vectorizer.

3.4.2 TF-IDF Vectorizer

This vectorizer also normalizes the text, i.e. reduces the impact of the tokens occurring in most of the documents, apart from tokenizing and counting. The value of the TF-IDF term in the vector increases if the frequency of the term is more in the comment text. However, the TF-IDF term decreases if the token appears in most of the comments. [3]

TF-IDF stands for Term Frequency Inverse Document Frequency. It consists of two parameters - term frequency (TF) and inverse document frequency (IDF), which decide the final value of the TF-IDF term

- **Term Frequency - TF (t, d)**: This is a local parameter which calculates the frequency of a term 't' in a document 'd'. This is similar to the Count Vectorizer method of encoding text.
- **Inverse Document Frequency - IDF (t)**: The IDF gives the inverse of the document frequency, i.e. it is a global parameter. Here, Document Frequency DF (t) is the count of documents (i.e. comments) that contain a tern t 't'. Therefore, it calculates the number of documents in which a term appears and it takes the inverse and log of that. In order to avoid a zero-division error, an extra 1 is added to the denominator. [4]

  $$IDF(t) = \log \frac{1 + |D|}{1 + df(t)} + 1$$

  where $df(t)$ denotes the number of documents containing the term 't' and $|D|$ denotes the total number of documents.

The final TF-IDF term is calculated as follows:

$$TFIDF = TF(t, d) \times IDF(t)$$

In this formula, the first term increases if the word is frequent in the document. However, if the word appears in most of the documents, then the second term decreases. Therefore, the TF-IDF term considers both local and global impact of a word in a comment, and it penalizes those words that have a high document frequency and are not meaningful in making predictions. Hence, TF-IDF vectorizer performs better than Count Vectorizer for feature extraction.

4 Prediction Algorithms

4.1 Problem Transformation

Since this problem is a multi-class classification problem as well as a multi-label classification problem, we need to transform it into either a separate single-class classification problem, or separate single-label classification problem, so that further machine learning algorithms can be applied to the problem. Following two methods are used for this [5]:

4.1.1 Binary Relevance

In this method, we transform the problem into separate single-class classification problems, each of the problems having a single label. We then apply the machine learning algorithm on each problem separately to get the result. As an example, if the problem is to classify a comment with six possible toxic labels, then the problem would be broken into separate six problems, such that each problem has a single label, hence each problem is a single-class classification problem. Thereafter, the results of the six problems can be combined to get all the labels for a comment.

Though this is a simple method to solve such multi-label classification problems, it has drawbacks. Since all the problems are treated as independent problems, this method neglects the correlation between the labels. Hence, it gives poor result if any correlation is observed between the labels.

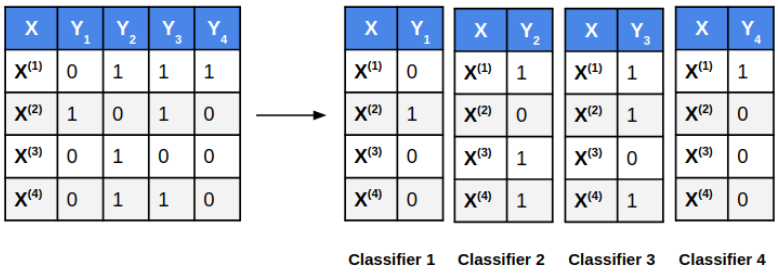


Figure 16: Binary Relevance

4.1.2 Classifier Chains

In this method, we transform the problem into separate single-label classification problems, such that if ith classifier is trained on input variable(s) X, then (i+1)th classifier is trained on input variable X as well as the output produced by ith classifier. Hence, in this method, the first classifier will be the same as the first classifier of Binary Relevance method, but the subsequent classifiers will also include the predictions of the previous classifiers as an input variable.

Thus, this technique also considers the correlation between the labels, since for every new classifier, the predictions of the previous classifiers are taken into account, i.e. for a given target variable, it also considers the correlation between previous target variables. Hence, this method is efficient than the Binary Relevance method when the labels are correlated with each other. However, it may perform poorly when all the labels are independent with each other.

For the task of hate speech detection, we applied the problem transformation methods using Binary Relevance and Classifier Chains, on Logistic Regression and Support Vector Machines. As there was a strong correlation between the tags, such as toxic-obscene, toxic-insult, etc, hence the Classifier Chains technique performed better than the Binary Relevance technique.

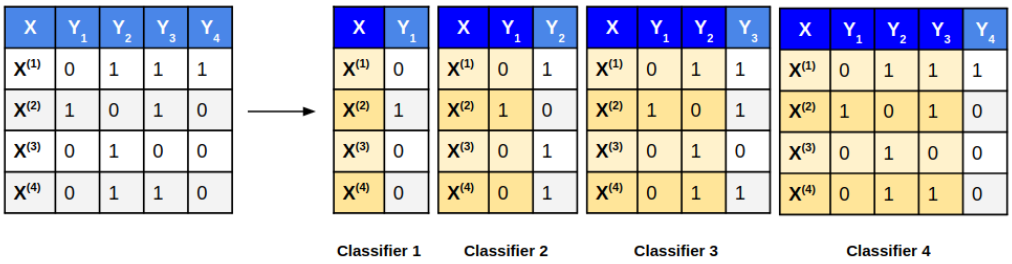


Figure 17: Classifier Chains

4.2 Support Vector Machines (SVM)

Support Vector Machines use the concept of support vectors and hyperplanes for classification tasks. They fall under the domain of supervised machine learning algorithms and can be used for both classification and regression tasks, particularly for the former one. They can be used for both linear and non-linear data. The main

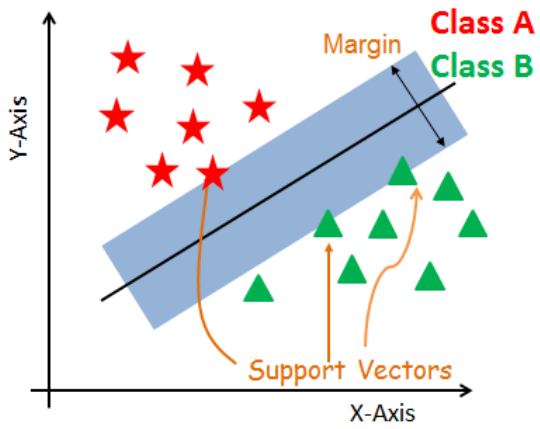


Figure 18: Support Vector Machine

objective of the support vector machine is to construct an optimal hyperplane in an N-dimensional space i.e. a boundary separating the data points, such that the margins between the support vectors is maximized. Here, the support vectors denote the data points which are closest to the hyperplane and are useful for training tasks, and the margin denotes the distance between the two parallel lines passing through the closest support vectors on either side of the hyperplane.

For non-linearly separable data, it transforms the original data into higher dimensions for the classification task. This is called the Kernel Trick. As an example, for the data points shown in the figure, the data points plotted in the left figure in x-y plane are not linearly separable, therefore the same data points are plotted in higher dimensional plane, say x-z plane in the right figure, such that $z = x^2 + y^2$. Thus, these data points can be linearly separable using support vector machines.

Support Vector Machines are very efficient for classifying higher dimensional data. They are very effective when the dimension of data is greater than the number of samples. This is because the complexity of the SVM classifier does not depend on the dimensionality of data, rather on the number of support vectors. Since only the support vectors are used for training, they are memory-efficient too. However, they perform poorly in the case of noisy data, e.g. when there is an overlap between the classes. Also, they are not much efficient for large datasets, as the time required to train the model is higher when the dataset is large.

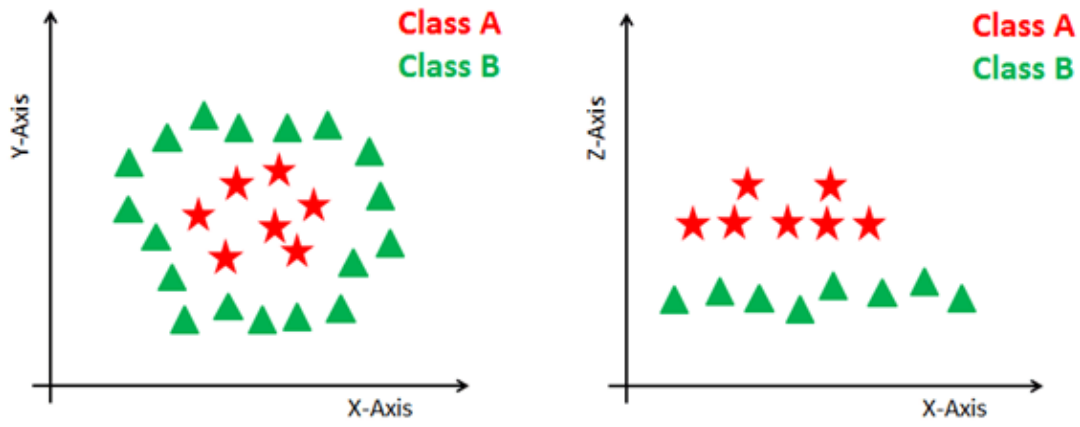


Figure 19: SVM on non-linearly separable data

4.3 Tree Based Methods

4.3.1 Decision Trees

Decision Trees are tree-like structures formed from simple decision rules by analysing the data. Decision trees are supervised learning algorithms. These can be used on both classification as well as regression tasks and hence the name CART (Classification And Regression Trees). Unlike other supervised algorithms such as Logistic or Linear Regression, the decision trees are non-parametric, so training/fitting steps do not involve 'learning' the parameters. Instead some simple decision based rules are learnt and arranged in a hierarchical fashion to help in classification/regression tasks. Decision trees have an advantage over other algorithms that they can be visually quite intuitive to understand on what factors the target variable is being decided - which features contribute to what extent in this decision process.

Our approach involves the use of a classification decision tree, in which the decision rules to test are located in the nodes of the tree. Based on the result of the decision node, a particular branch at that node is chosen which leads to some other node other tree with some other decision to make. This process is continued in a recursive fashion until the leaves of the tree are reached at last which contains a class label. This class label is attributed to the data item.

Decision trees are well suited for problems in which the data point can belong to several different target classes which is so in our case. For large datasets, the

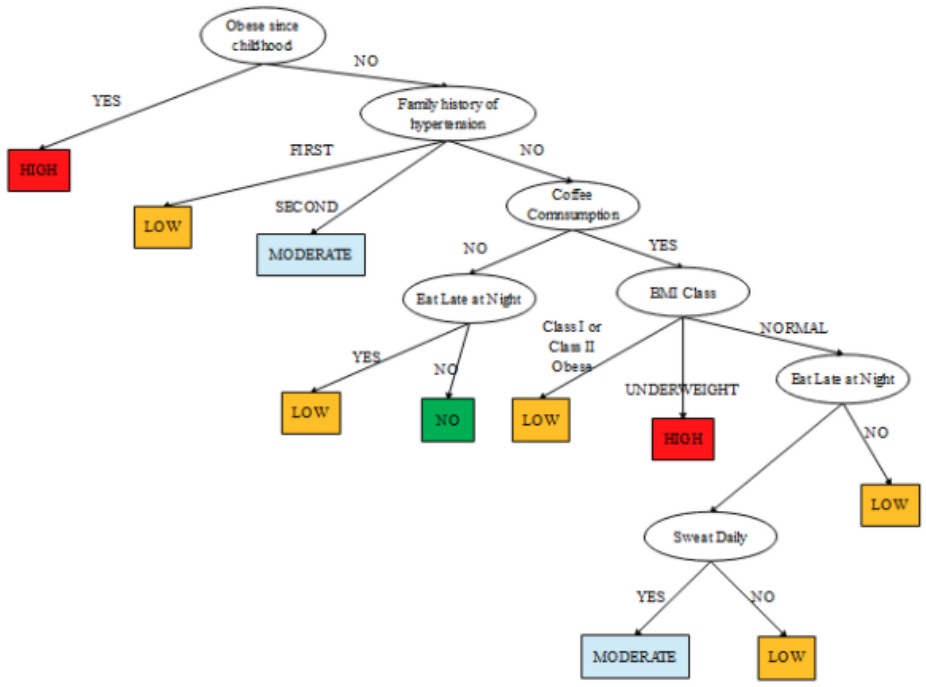


Figure 20: Decision Trees

efficiency of trees is proportional to log(N) where N is the number of data items in the training set. Besides, as opposed to a neural network based model, the decision trees are white-box models ie. to say one can easily visualise and understand why a particular example is being classified as toxic or obscene.

On the flip side, with a large number of data points available, decision trees become quite complicated, leaving the model exposed to the problem of overfitting the training set. Secondly there is no known efficient algorithm to learn a decision tree in polynomial time efficiently. Apart from this, our application involves classification of an imbalanced dataset and decision trees are known to perform poorly in such cases if proper class balancing methods have not been applied apriori.

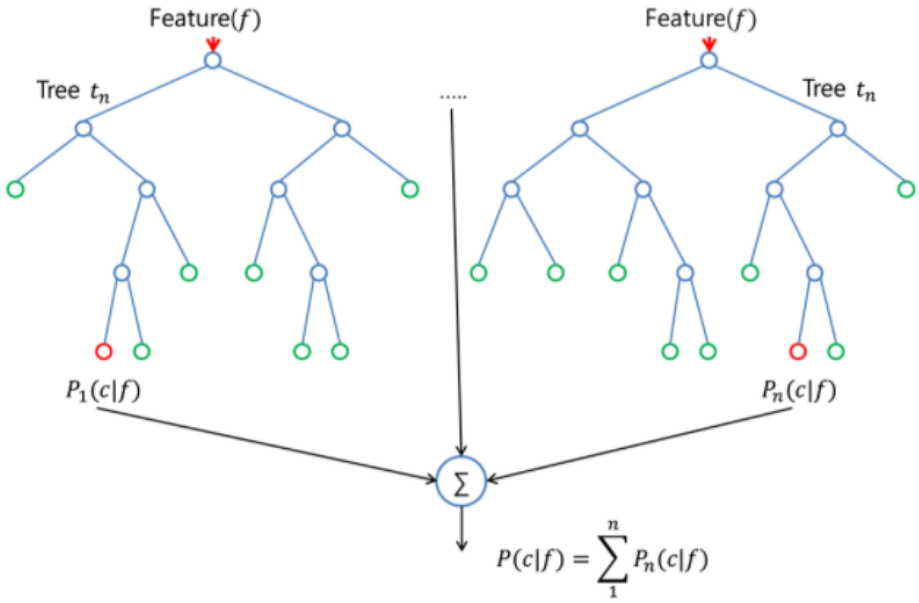


Figure 21: Decision Trees Ensembling

4.3.2 Decision tree Ensembling

To mitigate the shortcomings of the decision trees, ensemble methodologies are employed in which we combine the results of several decision trees which collectively decide the target variable for each data item. This improved the performance of the model at the cost of training time and efficiency as we now need to learn several trees instead of just one tree but this cost is justified in terms of 'return ratio' or performance gain in most cases. In an ensemble model, each decision tree is first learned independently and predicts the outcome for each data item. This prediction from different trees can now be combined to yield a single predicted estimate for target in a variety of ways. Generally mode is chosen for classification problems and arithmetic mean is chosen for regression problems.

4.3.3 Bagging

Bagging or bootstrap aggregation is used to form model ensemblings. The idea is that now each decision tree doesn't make the prediction based on all the training samples. Instead only a part of the total training data is used by each tree to make

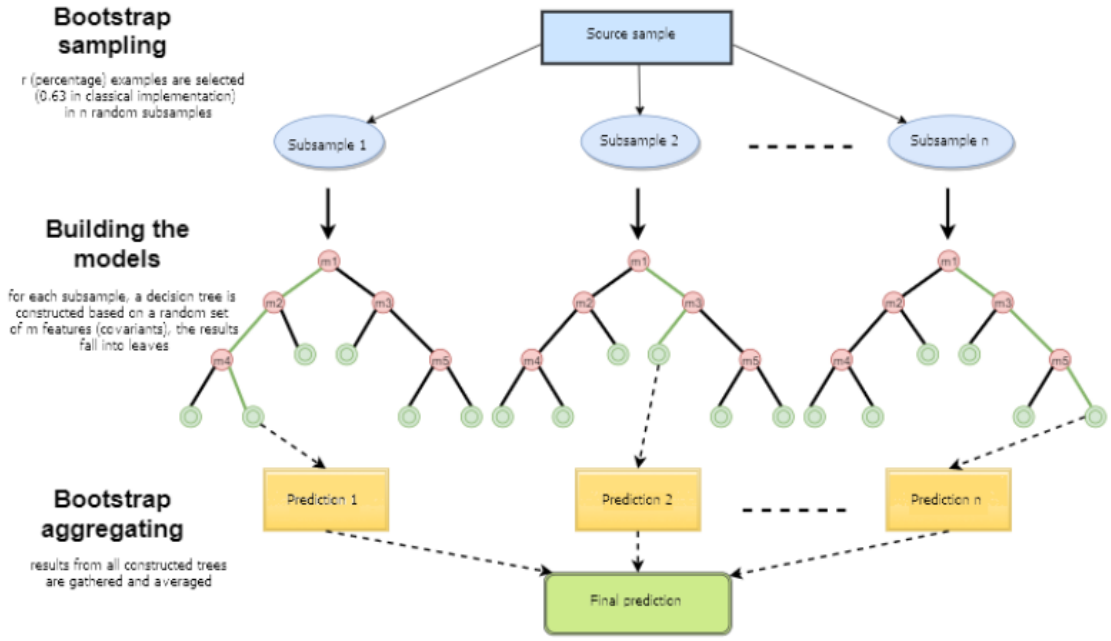


Figure 22: Bagging

a decision. The exact percentage used varies with implementation with 0.63 being a popular choice. This, in practice, has been shown to reduce the variance of the model.

For each tree a subset of the total data is chosen and is used during the training phase. It is possible for the same data point to appear in multiple subsets as the selection happens with replacement. The aim is to reduce the error of the ensembled model as all the trees won't make the same type of error. If one or two trees make a slight error in prediction, the prediction from other trees will serve to correct the error. While bagging on one hand increases the accuracy or performance of the model but at the same time it leads to a less white box model as it becomes more difficult to visualise the prediction of a set of decision trees.

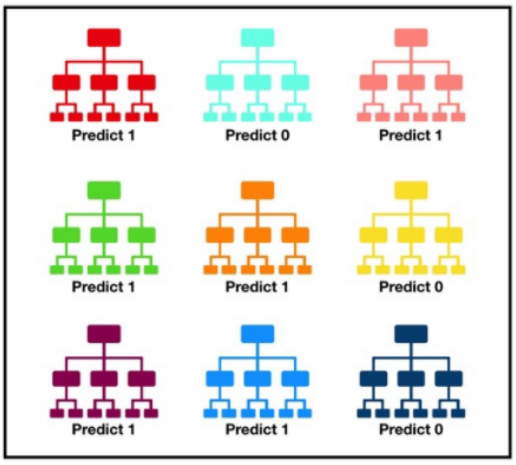


Figure 23: Random Forest

4.3.4 Random Forests

Bagging aims to correct the error of a few trees by combining the erroneous prediction with correct predictions of other trees. This approach will work only if all the trees do not make the same type of errors. But with bagging no steps are employed to take this into account. Though all the trees fit on a subset of data, yet all of them make the decision at each node in a greedy manner based on all the features. So the possibility that all of them will output the same result, correct or erroneous, is quite high. This defeats the very purpose of ensembling different decision trees.

Random Forests approach aims to decorrelate the predictions of the decision trees from one another. If the correlation between the predictions is less, then there is less chance that the trees will make the same type of error and this helps to reduce the chances of error. The decision trees in a random forest choose a random subset of features to make the predictions followed by selection of best features to make split. Thus all the trees end up pretty much uncorrelated with each other. This also ensures that a strongly relevant feature doesn't completely overshadow the contributions from all other weakly relevant features in all the trees. This helps the model to generalise well to the test set data and prevent the chances of predicting erroneous output.

4.3.5 Extremely Random Trees

Extremely Random Trees differ from the Random forest in two main points. First instead of choosing the optimal feature to make a split at each node, the extremely random tree chooses a random feature. This makes it more random as compared to the random forest and hence the name. Now based on this feature optimal split point is decided. Since a random feature is chosen instead of the optimal feature, the extremely random trees are much more computationally expensive as compared to the random forest at little or no cost of performance reduction. This serves to reduce the variance of the model.

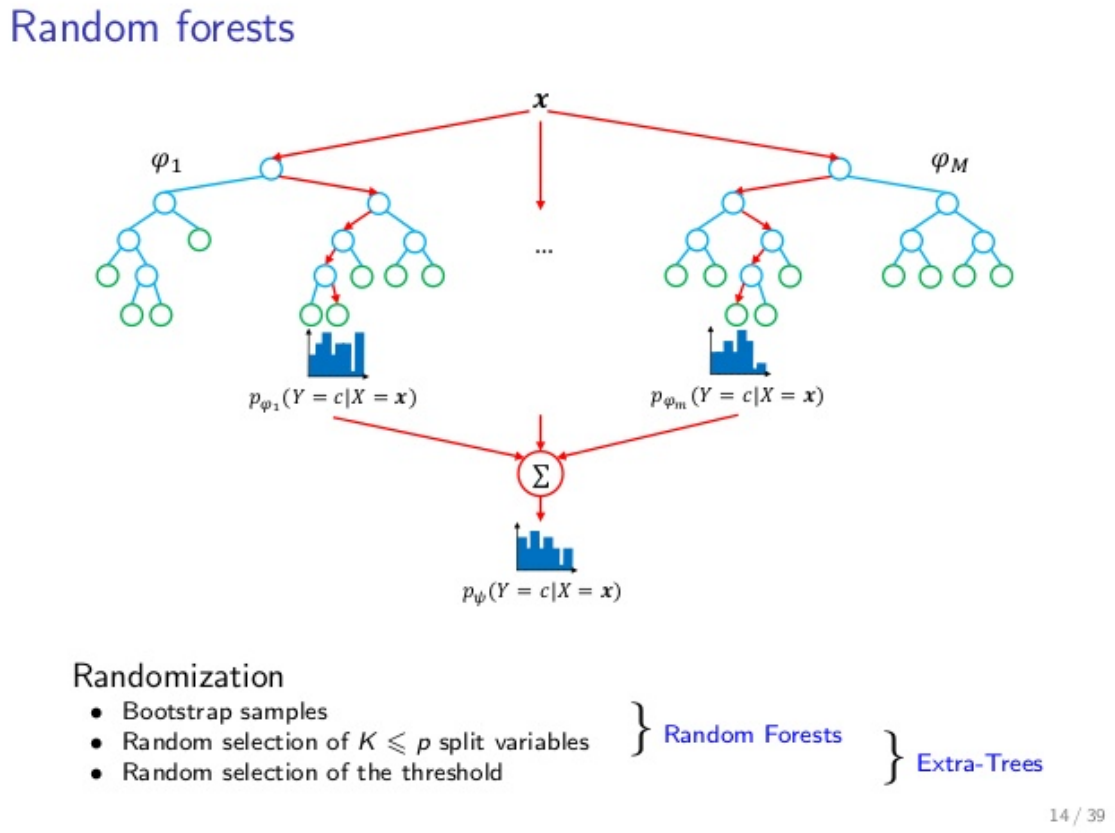


Figure 24: Extra Trees

Second difference arises in the way training subsample is choses. While random forest choses a subset of training data, the extra trees train on the complete training data thereby reducing the model bias. Thus these two differences help in reduction

of both bias as well as the variance of the extra trees as compared to the random forest while at the same time making it much more easy to train.

4.3.6 Gradient Boosting

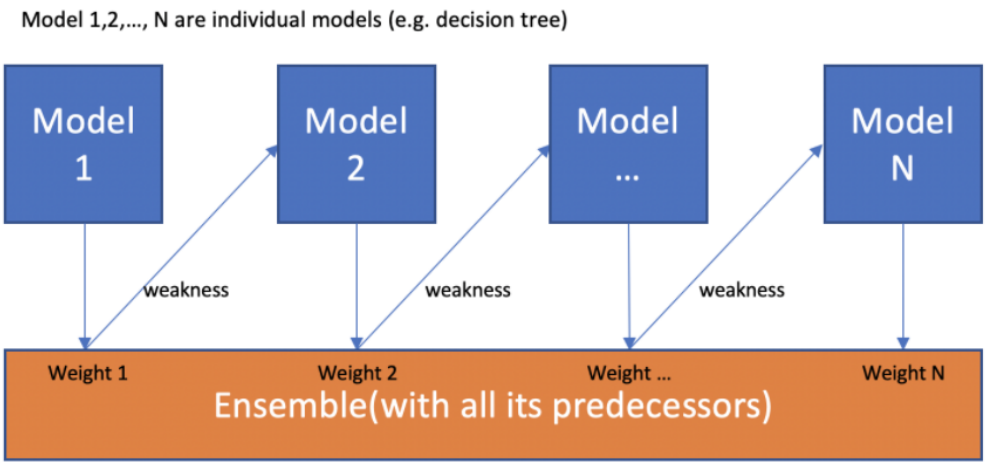


Figure 25: Boosting

Gradient Boosting can be explained as a blend of two concepts: gradient descent and boosting technique to leverage the advantages of the two.

Boosting can be understood as an ensemble technique employed to improve the model performance just like bagging. But here sequential training of models is done. This aims to do away the errors of the earlier trees while preserving the useful knowledge gained from them. The later trees are trained in such a way that the training samples which were classified incorrectly by the early trees are given more weightage which in turn incentivises them to classify them correctly. Owing to the sequential nature of training boosting is relatively slow however, it gives a significant 'boost' to the model performance thereby justifying its cost factor. Another important point to note is that earlier trees are simpler trying to capture only the coarse data distribution hence they are prone to underfitting. As we progress towards later trees, models become more and more complicated fitting the training set better. It is thus, important to carefully fine tune the correct number of models being trained in step to avoid overfitting the training set.

Gradient descent, as the name suggests, involves moving in the direction of steepest gradient. Though Gradient descent by itself does nothing to ensure that we always land in Global Minima and not in some local optima, but in practice, with multi dimensional datasets, local minima becomes relatively a non significant issue to worry about. However saddle points still remain a potential problem but we have a whole bunch of optimisers to alleviate this ranging from very simple ones like gradient descent with momentum to more complicated ones like Adam and Nesterov which are default choices almost everywhere.

Now Gradient Boosting as the name suggests involves building a sequence of trees one by one and then optimizing the loss of later trees by the use of gradient descent. The upside of Gradient Boosting is that it opens the door for the use of a wide range of loss functions as almost any differential loss function can be used for training the model with Gradient Descent Algorithm.

4.3.7 XG Boost

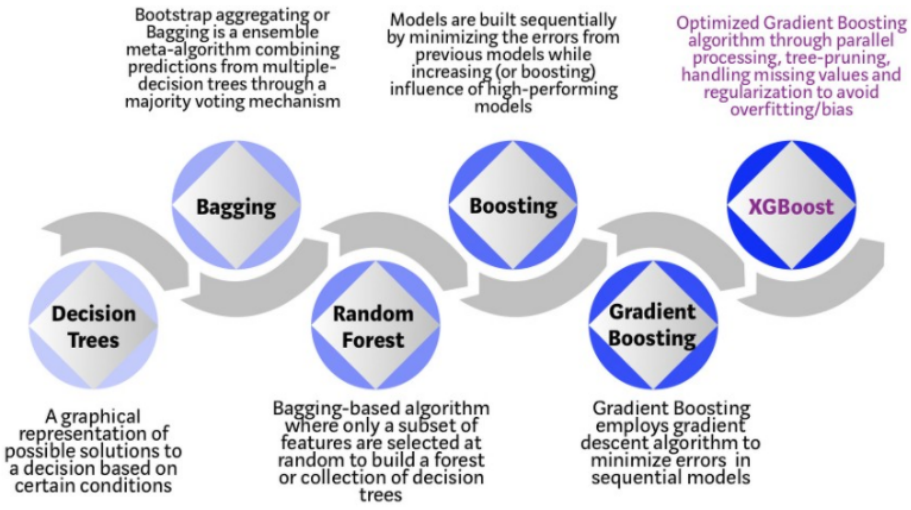


Figure 26: XGBoost

eXtreme Gradient Boost is a software library developed to curtail the training cost and improve the performance. It focuses on dual objectives: Algorithmic Improvements, and System Enhancement.

Algorithmic Improvements

1. Use of L1 and L2 regularisation to incentivise the model towards learning a simpler model. This in turn helps it genralise well to unseen examples.

2. The training data may have different levels of sparsity depending on the domain of the problem. XG Boost handles each of them by using a suitable representation. Apart from this, it also handles the missing data effectively.

3. To find the optimal split point, Weighted Quantile Sketch is employed.

4. With XGBoost, one can eliminate the validation phase altogether as the library itself does the cross validation. This makes it easier and more efficient at coding level to use XGBoost.

System Enhancements

1. Boosting approach requires building the trees in sequential fashion which is slow. XGBoost tries to parallelize this process as much as possible to eliminate the bottleneck. A Multi-Core CPU is used for this.

2. In the modern day applications, datasets sizes are insane and despite the increase in memory of machines, it's not uncommon to find the RAM size falling short of accommodating the complete training set. Fortunately XGBoost is quite scalable which stems from the fact that it uses 'out-of-core' computing.

3. XGBoost uses maximum depth parameter instead of negative loss criterion to prune the trees in reverse fashion. This increases the efficiency owing to the use of depth first method.

4. XGBoost also employs cache and buffer optimization to squeeze out the maximum possible gain in system performance.

Recognising the improvements and the potential of XGBoost we applied it in our case and the results were quite encouraging taking into account the hardness of the problem at hand.

4.4 Recurrent Neural Networks

Recurrent Neural Networks(RNNs) were developed for the processing of sequential data. RNN are recurrent in nature in the sense that they apply the same transformation function to every input of the data.The transformation function takes the previous hidden state along with current input. RNNs use their hidden states as memory to remember previously processed data. This makes RNN suitable for sequential data. RNN can capture dependencies but the problem is that the gradient decays exponentially with respect to length gap. Hence, simple RNNs face vanishing gradients. LSTM models can be used to overcome vanishing gradient problem of RNNs. They are described in the following section.

4.4.1 LSTM

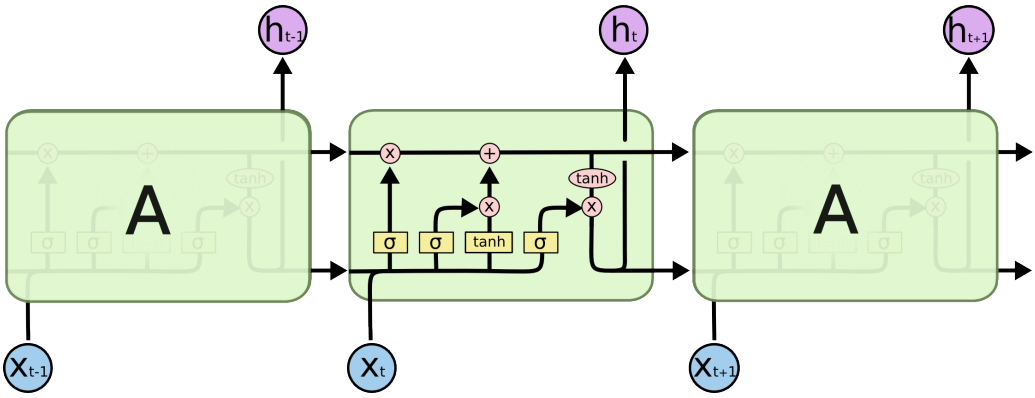
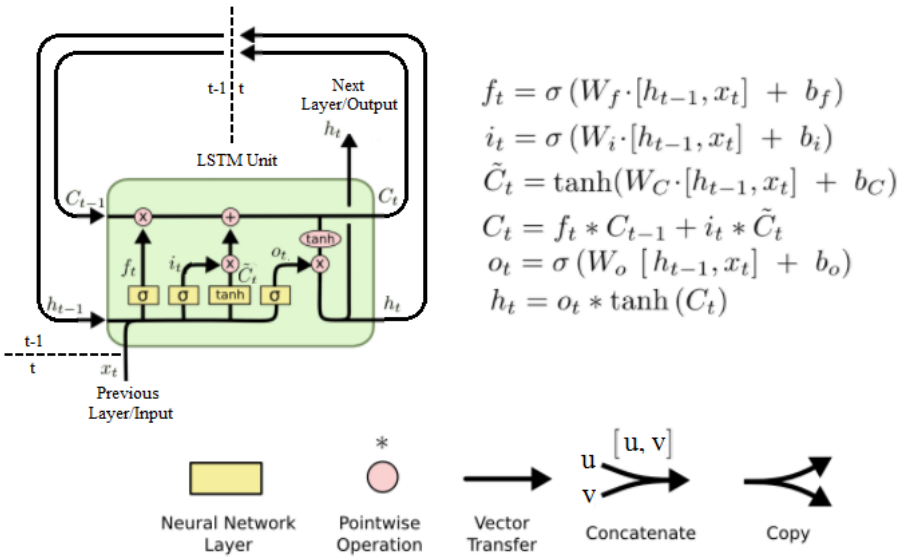


Figure 27: LSTM Model [1]

LSTMs are designed to capture long term dependencies. The recurring units of the LSTM store cell states(Ct) along with the hidden states. The information can flow through the cell state without any change. The cell state is regulated using gates which determine the amount of information that will flow through them. They also determine the amount of previous information that will be retained. LSTM unit takes current input, previous hidden state, previous cell state as the input and results in the new cell state and hidden state.

A single LSTM unit contains the following layers :

- Input Gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 28: Detailed LSTM unit with equations
(Source : Minimal LSTM)

- Forget Gate layer
- Output Gate layer
- Candidate layer
- Output layer

All the gate layers use sigmoid as the activation function. The candidate and output layers use tanh activation function. Candidate layer takes H_{t-1} and X_t as input and output candidate cell state($\tilde{C}_t$) based on the given current input and previous hidden state.

$$\tilde{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c)$$

Forget gate (f_t) determines how much the previous cell state will be forgotten. The forget gate layer takes current input and previous hidden state as input and outputs a number between 0 to 1. The resulting number is later element-wise multiplied by previous cell state.

$$f_t = \sigma(W_f.[h_t - 1, x_t] + b_f)$$

Input layer also takes current input and previous hidden state as input and outputs a number(It) between 0 to 1.The input gate regulates the amount by which candidate cell state will affect the actual cell state.

$$i_t = \sigma(W_i.[h_t - 1, x_t] + b_i)$$

Output layer generates the candidate output by applying tanh activation function on current cell state. Output gate layer takes current input and previous hidden state as input and outputs a number between 0 to 1. The final output is obtained by multiplying the output generated by the output layer and the output gate.

$$o_t = \sigma(W_o.[h_t - 1, x_t] + b_o)$$

The cell state and hidden state are modified as following :

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * tanh(C_t)$$

4.4.2 Bi-directional LSTM

Vanilla RNN and LSTM give output on the basis of current data and past data that have already passed through it. Unidirectional LSTM does not take into account data further in sequence while predicting on current data. Bi-directional LSTM overcomes this shortage. Bi-directional LSTM trains two independent LSTMs in opposite directions and connect the both the hidden layers to the same output. One LSTM is vanilla LSTM which trains in positive time direction (hidden states in this LSTM are called forward states), the other LSTM trains in reverse direction,that is for negative time direction (hidden states in this LSTM are called backward states).Using forward states and backward states, they can see the past and future context of the word and model is able to capture the whole picture and give better outputs.

4.4.3 Word Embeddings

Word embeddings are used to represent words in structured format so that machine learning algorithms can be applied on it. Since, the real world data is not structured, word embeddings techniques are a useful tool to transform data into more structured format so that useful information can be extracted.

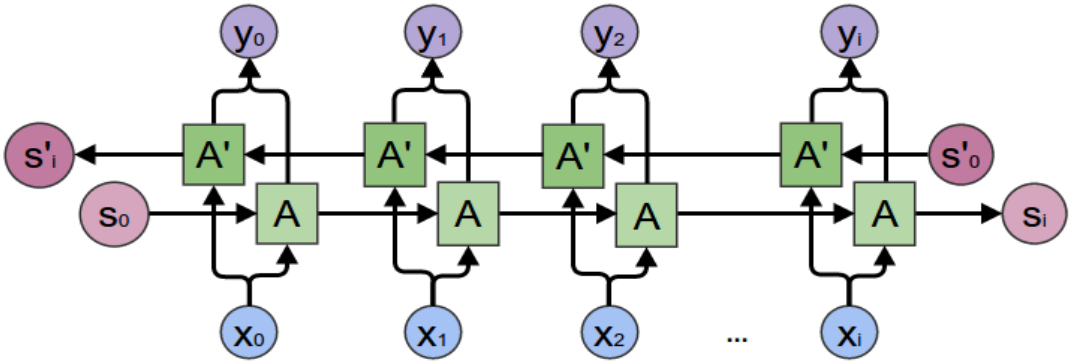


Figure 29: BiLSTM Model
(Source: Neural Networks, Types, and Functional Programming by Christopher Olah)

The Bag of words models are effective methods for extracting features from text, but it fails to capture semantic and context information from unstructured texts. One-hot encoded vectors may lead to a highly sparse structure which causes the model to overfit. To overcome these shortcomings of the above approaches, word embeddings are used.

Word embeddings represent words in the form of vectors in pre-defined dense vector space. These vectors contain meaningful semantic information about the words. The core idea behind the approach is that similar words have similar semantic information. Hence, the similar words will be in proximity within the high dimensional vector space. Thus, we can significantly reduce the vector size in contrast to the one-encoding technique.

4.4.4 Pretrained Word Embeddings

Pretrained word embeddings are obtained by unsupervised training of a model on a large corpora. As they are trained on a large corpus, they capture the semantic information of the words. Many pretrained-embeddings are made available by different organizations. Some of them are:

Word2Vec

Word2Vec is one of the earliest pre trained embedding models. It is classified in two approaches:

1. **Continuous Bag of Words (CBOW) model**: In this approach, the algorithm tries to predict the word if a context is given. In this way, the word embeddings vectors are generated.
2. **Skip-Gram Model**: In this approach, the algorithm tries to predict the context or surrounding words in which the word would have been used.It learns by predicting the surrounding words given a current word.

Fasttext

Fasttext learns embeddings by breaking each word in n-grams. This approach helps in obtaining information about the prefixes and suffixes of the word It uses the skip-gram model for learning embeddings. Consequently, it can work well on the less frequent words as well as the words which were not present in the training sample.

GloVe

GloVe stands for Global Vectors. In this approach, a word co-occurrence matrix is constructed. This helps in capturing the semantic information. The co-occurence matrix stores information about the frequency that appear in some context. Thus, it takes both local statistics and global statistics into account to obtain the embeddings.

5 Results and Analysis

5.1 Performance Measures

The problem involves highly unbalanced dataset. So accuracy is not a well suited performace measure. With only 10% of the training data belonging to the positive class (hate tags), it is trivial to achieve 90% accuracy by a naive model which simply labels every input as clean. Indeed we verified this by fitting a very simple naive bayes model which achieved a validation accuracy of whopping 97% which looks good on paper only as long as one doesn't analyse the model predictions manually or by some other performance measures. As quite expected, the recall was merely 65% which means the model is simply unable to recognise 35% of the hate comments. The precision scores were still poorer just 35% indicating even among the comments predicted as hate tags, 65% are misclassified. Needless to say, such a model is not of any use which highlights the essence of a good evaluation metric. Precision-Recall or F1 score seem like the next obvious choice however they have their own share of limitations including selection of threshold value and relative importance to be given to precision vs recall. Hence we finally settled on the ROC curve and AUC score which give a very accurate picture of the performance of a discriminative model.

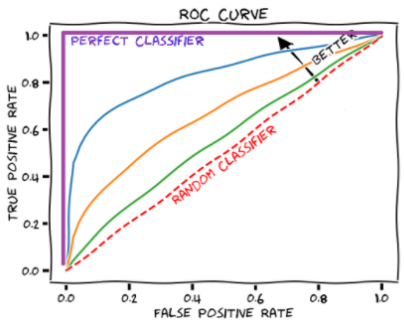


Figure 30: ROC

5.1.1 Receiver Operating Characteristic

A Receiver Operating Characteristic is a curve which plots True Positive Rate vs True Negative Rate. These two factors are an indicator of the model performance.

1. **True Positive Rate (TPR)**: $TPR = \frac{TP}{TP+FN}$

2. **False Positive Rate (FPR)** : $FPR = \frac{FP}{FP+TN}$

The aim of any discriminative model is to maximize the TPR while at the same time keeping the FPR as low as possible. The adjoining figure shows a perfect classifier and a random classifier.

5.1.2 AUC

AUC denotes the complete Area Under the ROC curve for the given domain. AUC values can range from 0 to 1. The idea of AUC stems from the observation that a better model will have more area under the ROC curve with a perfect model having AUC=1 and a model which always predicts incorrectly having AUC score=0. In this sense AUC can be understood as the average of performance measures of the classifier across all thresholds. Another interesting interpretation is that it expresses the probability that the model gives a higher positive score to a hate comment in our case as compared to a clean comment for any threshold value chosen.

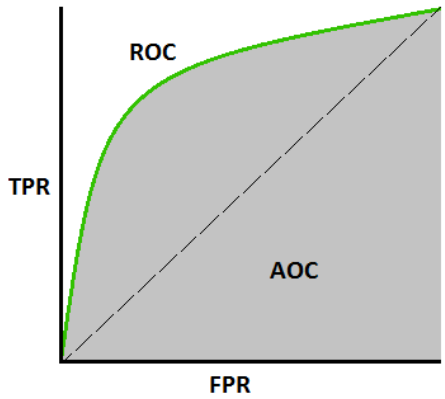


Figure 31: AOC ROC Curve

Advantages

1. **Scale Invariant**: AUC is independent of the scale and number of the predictions. It is based on fractions which always lie between 0 and 1. So it is a versatile model which can be used to compare the performance of several models alike.

2. **Threshold Invariant**: Unlike F-measure, it doesn't depend on the threshold set for classifying an example as positive.

Despite these, AUC does have some shortcomings which can be alleviated by combining it with some other metrics. We however limit ourselves to AUC for the purpose of this project.

5.2 Comparison Table

The mean AUC_ROC scores obtained from different models are mentioned in the table:

Model	Mean AOC_ROC Score
Support Vector Machines (Binary Relevance)	0.66
Support Vector Machines (Classifier Chains)	0.67
Logistic Regression (Binary Relevance)	0.73
Logistic Regression (Classifier Chains)	0.76
Extra Trees	0.93
XGBoost	0.96
LSTM without pretrained embeddings	**0.97**
LSTM with FastText embedding	0.96
LSTM with Glove embedding	0.88
LSTM with Word2Vec embedding	0.85

Table 1: Mean AUC_ROC scores obtained from different models

Conclusion

We compared the performance of the model based on the mean AOC_ROC scores. We observed that the classical models like Support Vector Machines and Logistic Regression failed to achieve high AUC_ROC scores. We also found out that the classifier chain method performed slightly better than binary relevance in this task. The tree based ensembling models performed significantly better than the classical models. The RNN based models outperformed other models in this task. We observed that the use of pre trained word embeddings did not improve the performance of the LSTM model. One reason may be that the word embeddings were trained in a different context and the number of common words may be very less. We can further test the performance of state of the art models like Transformers on this task. We can also experiment with more sophisticated models like GRUs. We can ensemble the results obtained from the various models in a majority vote fashion.

References

[1] Understanding lstm networks – colah's blog.

[2] Feature extraction, scikit learn.

[3] Tf-idf model for page ranking, Aug 2019.

[4] William Scott. Tf-idf for document ranking from scratch in python on real world dataset., May 2019.

[5] Shubham Jain. Multi label classification: Solving multi label classification problems, Jun 2020.

[6] José Hernández-Orallo, Peter Flach, and César Ferri. Roc curves in cost space. *Machine Learning*, 93(1):71–91, 2013.

[7] P. Sedgwick. How to read a receiver operating characteristic curve. *Bmj*, 350(may08 2), 2015.

[8] Bagging and boosting: Most used techniques of ensemble learning, Nov 2020.

[9] Classification: Roc curve and auc — machine learning crash course.

[10] Jason Brownlee. A gentle introduction to xgboost for applied machine learning, Apr 2020.

[11] Vishal Morde. Xgboost algorithm: Long may she reign!, Apr 2019.

[12] AlindGuptaCheck. Ml: Extra tree classifier for feature selection, Jul 2020.

[13] Frank Ceballos. An intuitive explanation of random forest and extra trees classifiers, Apr 2020.

[14] Naman Bhandari. Extratreesclassifier, Oct 2018.

[15] Tony Yiu. Understanding random forest, Aug 2019.

[16] Prashant Gupta. Decision trees in machine learning, Nov 2017.

[17] Decision trees - science direct.

Appendix (Code)

```
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.svm import SVC

model_br = BinaryRelevance(classifier = SVC(), require_dense = [False, True])
model_br.fit(X_train_feat, y_train)
```

```
BinaryRelevance(classifier=SVC(C=1.0, break_ties=False, cache_size=200,
                               class_weight=None, coef0=0.0,
                               decision_function_shape='ovr', degree=3,
                               gamma='scale', kernel='rbf', max_iter=-1,
                               probability=False, random_state=None,
                               shrinking=True, tol=0.001, verbose=False),
                require_dense=[False, True])
```

```
preds_train = model_br.predict(X_train_feat)
```

```
from sklearn.metrics import roc_auc_score, accuracy_score

print(roc_auc_score(y_train, preds_train.toarray()))
```

```
0.7689564894630726
```

```
preds_val = model_br.predict(X_val_feat)
```

```
print(roc_auc_score(y_val, preds_val.toarray()))
```

```
0.6665239237377513
```

Figure 32: Support Vector Machine (Binary Relevance)

```
from skmultilearn.problem_transform import ClassifierChain
from sklearn.svm import SVC

model_cc = ClassifierChain(classifier = SVC(), require_dense = [False, True])
model_cc.fit(X_train_feat, y_train)
```

```
ClassifierChain(classifier=SVC(C=1.0, break_ties=False, cache_size=200,
                               class_weight=None, coef0=0.0,
                               decision_function_shape='ovr', degree=3,
                               gamma='scale', kernel='rbf', max_iter=-1,
                               probability=False, random_state=None,
                               shrinking=True, tol=0.001, verbose=False),
                order=None, require_dense=[False, True])
```

```
preds_train = model_cc.predict(X_train_feat)
```

```
from sklearn.metrics import roc_auc_score, accuracy_score

print(roc_auc_score(y_train, preds_train.toarray()))
```

```
0.7645674561502386
```

```
preds_val = model_cc.predict(X_val_feat)
```

```
print(roc_auc_score(y_val, preds_val.toarray()))
```

```
0.6751504109269661
```

```
preds_test = model_cc.predict(X_test_feat)
```

```
print(roc_auc_score(labels_consider, preds_test.toarray()))
```

```
0.6718590534519624
```

Figure 33: Support Vector Machine (Classifier Chains)

```
model=LogisticRegression(C=20.0, max_iter=1000000)
```

```
## Binary Relevance
```

```
scores_roc_aoc = []

for label_name in classes:
    print('Class:', label_name)

    model.fit(X_train_feat, y_train[label_name])
    preds_train = model.predict(X_train_feat)
    train_roc_auc = roc_auc_score(y_train[label_name], preds_train)
    print('Train ROC AUC Score:', train_roc_auc)

    preds_val = model.predict(X_val_feat)
    val_roc_auc = roc_auc_score(y_val[label_name], preds_val)
    print('Val ROC AUC Score:', val_roc_auc)

    preds_test = model.predict(X_test_feat)
    test_roc_auc = roc_auc_score(labels_consider[label_name], preds_test)
    print('Test ROC AUC Score:', test_roc_auc)
    scores_roc_aoc.append(test_roc_auc)

    print()


print(np.mean(scores_roc_aoc))
```

Figure 34: Logistic Regression (Binary Relevance)

```
## Classifier chains
```

```
from scipy.sparse import hstack, csr_matrix

def add_feat(x, feat):
  return hstack([x, csr_matrix(feat).T], 'csr')
```

```
scores_roc_aoc = []

for label_name in classes:
    print('Class:', label_name)

    model.fit(X_train_feat, y_train[label_name])
    preds_train = model.predict(X_train_feat)
    train_roc_auc = roc_auc_score(y_train[label_name], preds_train)
    print('Train ROC AUC Score:', train_roc_auc)

    preds_val = model.predict(X_val_feat)
    val_roc_auc = roc_auc_score(y_val[label_name], preds_val)
    print('Val ROC AUC Score:', val_roc_auc)

    preds_test = model.predict(X_test_feat)
    test_roc_auc = roc_auc_score(labels_consider[label_name], preds_test)
    print('Test ROC AUC Score:', test_roc_auc)
    scores_roc_aoc.append(test_roc_auc)

    X_train_feat = add_feat(X_train_feat, y_train[label_name])
    X_val_feat = add_feat(X_val_feat, y_val[label_name])
    X_test_feat = add_feat(X_test_feat, labels_consider[label_name])
    print(X_train_feat.shape)
    print(X_val_feat.shape)
    print(X_test_feat.shape)
    print()

print(np.mean(scores_roc_aoc))
```

Figure 35: Logistic Regression (Classifier Chains)

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.ensemble import ExtraTreesClassifier

losses = []
predictions = []
for class_name in class_names:
    train_target = train[class_name]
    classifier = ExtraTreesClassifier(n_estimators=30)
    score = np.mean(cross_val_score(classifier, comments_train, train_target, cv=2, scoring='roc_auc'))
    print(class_name, score)
    classifier.fit(comments_train, train_target)
    predictions.append(classifier.predict_proba(comments_test)[:, 1])
```

```
toxic 0.9483975859448478
severe_toxic 0.9356866512204218
obscene 0.972690700510528
threat 0.8646456662150368
insult 0.9551477410459125
identity_hate 0.8819367764977541
```

```
labels=pd.read_csv('/content/drive/My Drive/Colab_data/Data_Mining/Data-Mining-Project/dataset/test_labels.csv')
labels=np.array(labels.iloc[:,1:])
sum_labels=np.sum(labels,axis=1)
idx=sum_labels>=0
```

```
preds_consider=np.array(predictions)[:,idx]
labels_consider= labels[idx]
preds_consider.shape,labels_consider.shape
```

```
((6, 63978), (63978, 6))
```

Figure 36: Extra Trees

```
import xgboost as xgb
```

```
def run(train_X, train_y, test_X, test_y=None, feature_names=None):
    dic = {}
    dic['objective'] = 'binary:logistic'
    dic['eta'] = 0.1
    dic['max_depth'] = 6
    dic['silent'] = 1
    dic['eval_metric'] = 'auc'
    dic['min_child_weight'] = 1
    dic['subsample'] = 0.7
    dic['colsample_bytree'] = 0.7
    num = 100
    list_dic = list(dic.items())

    xgtrain = xgb.DMatrix(train_X, label=train_y)
    xgtest = xgb.DMatrix(test_X, label=test_y)

    model = xgb.train(list_dic, xgtrain, num, [ (xgtrain,'train'), (xgtest, 'test') ], early_stopping_rounds=10)

    return model
```

```
col = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']
preds = np.zeros((test.shape[0], len(col)))

for i, j in enumerate(col):
    print('fit '+j)
    model = run(comments_train, train_y[j], comments_val,val_y[j])
    preds[:,i] = model.predict(xgb.DMatrix(comments_test), ntree_limit = model.best_ntree_limit)
    gc.collect()
```

Figure 37: XGBoost

```
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
```

```
inp = Input(shape=(maxlen, ))
```

```
embed_size = 128
x = Embedding(max_features, embed_size)(inp)
```

```
x = LSTM(60, return_sequences=True,name='lstm_layer')(x)
```

```
from keras.layers import Bidirectional, GlobalMaxPool1D
```

```
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
```

```
x = Dense(50, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(6, activation="sigmoid")(x)
```

```
from keras.models import Model
import keras.metrics as metrics
```

```
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=[
                          metrics.MeanSquaredError(),
                          metrics.AUC(),
                  ])
```

```
model.summary()
```

Figure 38: LSTM without pretrained embeddings

```
embedding_matrix = loadEmbeddingMatrix('fasttext')
```

```
Loaded 110995 word vectors.
total embedded: 59312 common words
```

```
embedding_matrix.shape
```

```
(221341, 300)
```

```
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
from keras.layers import Bidirectional, GlobalMaxPool1D,Bidirectional
from keras.models import Model
```

```
inp = Input(shape=(maxlen, ))
```

```
x = Embedding(len(tokenizer.word_index), embedding_matrix.shape[1],weights=[embedding_matrix],trainable=False)(inp)
x = Bidirectional(LSTM(60, return_sequences=True,name='lstm_layer',dropout=0.1,recurrent_dropout=0.1))(x)
```

```
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
```

```
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
x = Dense(50, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(6, activation="sigmoid")(x)
```

```
import keras.metrics as metrics
```

```
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=[
                          metrics.MeanSquaredError(),
                          metrics.AUC(),
                  ])
```

Figure 39: LSTM with FastText embedding

```
: embedding_matrix = loadEmbeddingMatrix('glove')
```

```
Loaded 1193514 word vectors.
total embedded: 81129 common words
```

```
: embedding_matrix.shape
```

```
: (221341, 25)
```

```
: from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
  from keras.layers import Bidirectional, GlobalMaxPool1D,Bidirectional
  from keras.models import Model
```

```
: inp = Input(shape=(maxlen, ))
```

```
: x = Embedding(len(tokenizer.word_index), embedding_matrix.shape[1],weights=[embedding_matrix],trainable=False)(inp)
  x = Bidirectional(LSTM(60, return_sequences=True,name='lstm_layer',dropout=0.1,recurrent_dropout=0.1))(x)
```

```
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will
rnel as fallback when running on GPU
```

```
: x = GlobalMaxPool1D()(x)
  x = Dropout(0.1)(x)
  x = Dense(50, activation="relu")(x)
  x = Dropout(0.1)(x)
  x = Dense(6, activation="sigmoid")(x)
```

```
: import keras.metrics as metrics
```

```
: model = Model(inputs=inp, outputs=x)
  model.compile(loss='binary_crossentropy',
                    optimizer='adam',
                    metrics=[
                            metrics.MeanSquaredError(),
                            metrics.AUC(),
                    ])
```

Figure 40: LSTM with Glove embedding

```
embedding_matrix = loadEmbeddingMatrix('word2vec')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:36: DeprecationWarning: Call to deprecated `wv` (Attribute will be removed i
n 4.0.0, use self instead).
Loaded 306943 word vectors.
total embedded: 0 common words
```

```
embedding_matrix.shape
```

```
(221341, 300)
```

```
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
from keras.layers import Bidirectional, GlobalMaxPool1D,Bidirectional
from keras.models import Model
```

```
inp = Input(shape=(maxlen, ))
```

```
x = Embedding(len(tokenizer.word_index), embedding_matrix.shape[1],weights=[embedding_matrix],trainable=False)(inp)
x = Bidirectional(LSTM(60, return_sequences=True,name='lstm_layer',dropout=0.1,recurrent_dropout=0.1))(x)
```

```
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
WARNING:tensorflow:Layer lstm_layer will not use cuDNN kernel since it doesn't meet the cuDNN kernel criteria. It will use generic GPU ke
rnel as fallback when running on GPU
```

```
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
x = Dense(50, activation="relu")(x)
x = Dropout(0.1)(x)
x = Dense(6, activation="sigmoid")(x)
```

```
import keras.metrics as metrics
```

```
model = Model(inputs=inp, outputs=x)
model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=[
                         metrics.MeanSquaredError(),
                         metrics.AUC(),
                  ])
```

Figure 41: LSTM with Word2Vec embedding