Ensemble Learning: Cyberbullying Classification

Git Repository - https://github.com/RISHIKESHAVAN/ensemble-learning

Final Project Report

M.Sc. in Data Sciences and Business Analytics - Ensemble Learning

CentraleSupelec

Rishikeshavan Rengarajan
rishikeshavan.rengarajan@student-cs.fr

Basma Wehbe
basma.wehbe@student-cs.fr

Raghuwansh Raj
raghuwansh.raj@student-cs.fr

Shivaditya Meduri
meduri.shivaditya@student-cs.fr

Syed Ahsan Abbas Naqvi
syedahsanabbas.naqvi@student-cs.fr

1 INTRODUCTION

In today's world which has been made smaller by technology, new age problems have been born. There is no doubt that technology has a lot of benefits. However, it also comes with a negative side. It has given birth to cyberbullying. Cyberbullying refers to the misuse of information technology with the intention to harass others.

Subsequently, cyberbullying comes in various forms. It does not necessarily mean hacking someone's profiles or posing to be someone else. It also includes posting negative comments about somebody or spreading rumors to defame someone. As social media usage becomes increasingly prevalent in every age group, a vast majority of citizens rely on this essential medium for day-to-day communication. Social media's ubiquity means that cyberbullying can effectively impact anyone at any time or anywhere, and the relative anonymity of the internet makes such personal attacks more difficult to stop than traditional bullying.

Problem Definition

Given a set of tweets taken from Twitter, the social networking service, we need to classify if the tweet pertains to cyberbullying. The problem is split into two:

- *Multi-classification* - If the tweet does pertain to cyberbullying, then classify the type of cyberbullying.
- *Binary classification* - Whether the tweet pertains to cyberbullying or not.

For example, the following is a tweet from the dataset.

```
Here at home. Neighbors pick on my family and I.
Mind you my son is autistic. It feels like high school.
They call us names attack us for no reason and
bully us all the time. Can't step on my front porch
without them doing something to us
```

Based on the terms in the text, the *multi-classifier* should classify this tweet as `Age` and the *binary classifier* should classify this tweet as `cyberbullying`.

2 DATASET DESCRIPTION

The dataset consists of two columns: *tweet_text* and *cyberbyllying_type*. There are 47692 tweets and each of them are labelled with their *cyberbyllying_type*. The *cyberbyllying_type* feature contains of the following classes:

- religion
- age
- gender
- ethnicity
- other_cyberbullying
- not_cyberbullying

The distribution of the dataset across the different classes is quite balanced with approximately 8000 records for each class as shown in Figure 1.

Figure 1: Distribution of records across different classes

In order to accommodate the dataset for binary classification, a new feature was added based on the `cyberbullying_type` feature where all the types of cyberbullying were labelled commonly as `cyberbullying`. Thus, in total the new feature had only two labels: `cyberbullying` and `not_cyberbullying`.

3 METHODOLOGY

The *tweet_text* column consists of actual tweets. In order to derive useful features from them, various text pre-processing steps were followed. After pre-processing, different ensemble methods were experimented on them with hyper-parameter tuning using `Grid Search` and `Random Search`.

Text Pre-processing

Text pre-processing is a method to clean the text data and make it ready to feed data to the model. Text data generally contains noise in various forms like emotions, punctuation, text in different cases etc. The data we have in the dataset contains of tweets, which contain lot of such noises. To handle these noises, different text pre-processing steps were carried out which are listed below:

- *Extract hashtags and emojis* - Hashtags and emojis are widely used to express emotions. Thus they might help in interpreting the emotions of the tweeter. We extract the emojis and

hashtags from the tweets and store them as separate features. Hashtags are extracted using the `regex` package and the emojis are extracted using the `emojis` package.

- *Remove mentions and URLs* - Mentions in tweets are when other users are tagged using the `'@'` symbol. We remove the name mentions and the URLs from the tweets since they won't contribute much to the model.
- *Handling punctuation* - Punctuation are noises while processing text. We are more interesting in the words itself.
- *Lower case* - We convert all upper case alphabets to lower case since a computer does not require upper case letters.
- *Normalize accented characters* - The tweets are mainly in *English*. Accents on characters in English have a fairly marginal status. In our context, the important question is *"How likely is it that users tweet with accents on the characters?"* The general behaviour would be that users tweet without adhering much to the rules of accents on the characters and that is due to various reasons like speed and laziness because tweets are meant to be short and instant. Thus, we proceed to remove such accents from characters and normalize all the letters to common English characters. This is done by listing all the common accents and using `string replace` to replace accented characters with their canonical form.
- *Stop word filtering* - These are actually the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are "the", "a", "an", "so", "what". Since they don't add much information to the text, we can remove them. We get the list of stopwords in English from the `nltk` package and ignore words that are in that list.
- *Lemmatization* - Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meanings to one word. We use the `WordNetLemmatizer` form the `nltk` package to get the lemmatized form of the words.
- *Extracting meaning of Emoji* - There were lots of emoji present in the text , so we separated the emoji from the text and created feature as number of emoji and demojify it in order to get the textual meaning of emoji.Later we vectorise it to get numerical features

Table 1 displays the WordClouds formed from the cleaned tweets of the different categories of tweets. *(The category that the WordCloud belongs to is given as a title to each image)*

TF-IDF Matrix

TF-IDF (Term Frequency - Inverse Document Frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics:

- *Term Frequency* - how many times a word appears in a document
- *Inverse Document Frequency* - inverse of the word frequency across a set of documents.

Table 1: WordClouds

We use `TfidfVectorizer` from the `scikit-learn` package to get the TF-IDF values for the cleaned tweet text, hash tags and the emoji texts. For the cleaned tweets, we add additional parameters to the `TfidfVectorizer` as follows:

- `min_df = .0005` - Ignores terms that have a document frequency strictly lower than 0.05% of the documents. Used to ignore terms that occur very rarely across the documents and thus have very less importance.
- `max_df = .90` - Ignores terms that have a document frequency strictly higher than 90% of the documents. Used to ignore very common terms similar to stop words.

The individual matrices are then concatenated together to form one single matrix and the duplicate columns are then dropped from them. While dropping the duplicate columns, we retain the TF-IDF values of the terms from the cleaned tweet and drop the repeated columns from the hash tag and emoji matrices. As a result of this we get a matrix of shape - (47692, 7007).

Due to the high dimensionality of the matrix, we want to use only the important features among them. For this purpose, we use decision trees to perform feature importance extraction. We extract the feature importance from the best decision tree model and use only the features with importance greater than 0. This reduced the matrix shape to (47692, 1757).

Count Vectorisation

A different approach called *count vectorisation* was also tried and used for some of the models. Count Vectorization involves counting the number of occurrences each words appears in a document. There are 2 types of parameters that we have played with. we have finally used maximum features of 1500 for text and 500 for each one of hashtag and emoji text. we have used ngrams in range of 1-3

- *max_features* - Setting this parameter can be incredibly useful when dealing with a large amount of documents as we may run into speed issues with computation.
- *n_gram_range* - This deals with contiguous sequences of words.we can get a lot more meaning than if we put all

of these words as independent features. we could lose the meaning if we don't set this parameter.

Ensemble Methods

The principle of ensemble learning is to combine multiple elements, that can be categorized as *weak learners*, in order to improve robustness and prediction performances. The parameters of this combination can also be tuned to improve the performance. This process is called *hyper-parameter tuning*.

Hyper-Parameter Tuning. For the different ensemble methods listed below, hyper-parameter tuning was done in order to optimize the parameters and achieve better results. Hyper-parameter tuning with `GridSearchCV` took a lot of time especially for the Boosting models. One such iteration for XGBoost for the multi-classification case took more than two days. Due to the time constraint, we carried out hyper-parameter tuning using `GridSearchCV` for Decision Tree and Random Forest and used `RandomSearchCV` with 30 iterations for the rest of the models.

Decision Trees: They are the basic structure of many ensemble learning models. With an aim to maximise the homogeneity of training data, decision trees work by splitting the input space into sub-spaces and maximise the homogeneity within each sub-space. *Hunt's Algorithm* is commonly used to build the tree structure. The method used to define the best split makes different decision tree algorithms. There are many measures that can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after splitting and the best splitting is the one that has more purity after the splitting. Instead of defining a split's purity, the impurity of its child node is used. There are a number of commonly used impurity measurements: *Entropy, Gini Index* and *Classification Error*. Decision trees are also vulnerable to over-fitting. In decision trees, over-fitting occurs when the tree is designed so as to perfectly fit all samples in the training data set. Thus, it ends up with branches with strict rules of sparse data. This affects the accuracy when predicting samples that are not part of the training set. One of the methods used to address over-fitting in decision tree is called *pruning*, where you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed.

The various hyper-parameters tested for decision tree are as follows:

- *max_depth*: Maximum depth of the tree
- *criterion*: Function to measure the quality of a split
- *splitter*: Strategy used to choose the split at each node

The performance of decision trees on both multi-classification and binary classification using count vectorization and TFIDF techniques are shown in the table below

Table 2: Decison Tree Performance

Vectorisation	Classification	f1 score
count_vectoriser	binary	0.82
count_vectoriser	multi	0.77
TFIDF_vectoriser	binary	0.82
TFIDF_vectoriser	multi	0.79

Decision trees are considered as *weak learners* They are not able to spot hidden non linearity in the data and are prone to have high variation. We move to the model that has low bias as well as low variance

Bagging: The idea of bagging (bootstrap + aggregating) is simple: fit several independent models and "average" their predictions in order to obtain a model with a lower variance. However, we can't, in practice, fit fully independent models because it would require too much data. So, we rely on the good "approximate properties" of *bootstrap samples* to fit models that are almost independent. Once multiple bootstrap samples are created, a weak learner is then fit for each of these samples and their outputs are averaged. This gives an ensemble model with less variance. Roughly speaking, as the bootstrap samples are approximately independent and identically distributed *(i.i.d.)*, so are the learned base models. Then, "averaging" weak learners outputs do not change the expected answer but reduce its variance (just like averaging *i.i.d.*random variables preserve expected value but reduce variance).

Random Forest: The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree and the class with the most votes becomes our model's prediction. A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The various hyper-parameters tested for random forest are as follows:

- *max_depth*: Maximum depth of the trees
- *n_estimators*: Number of trees in the forest
- *min_samples_split*: Minimum number of samples required to split an internal node
- *min_samples_leaf*: Minimum number of samples required to be at a leaf node

The performance of random forest on both multi-classification and binary classification are shown in the table below

Table 3: Random Forest Performance

Vectorisation	Classification	f1 score
TFIDF_vectoriser	binary	0.85
TFIDF_vectoriser	multi	0.82

While Random Forest model does perform better than individual decision trees,thanks to its averaging technique that reduces the

variance of the model It has its own limitations. The trees are made uncorrelated to maximize the decrease in variance, but the algorithm cannot reduce bias (which is slightly higher than the bias of an individual tree in the forest). Hence the need for large, unpruned trees, so that the bias is initially as low as possible.

Boosting: Boosting is an ensemble learning method that combines a set of weak learners into a strong learner to minimize training errors. In boosting, a random sample of data is selected, fitted with a model and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. With each iteration, the weak rules from each individual classifier are combined to form one, strong prediction rule.. In bagging, weak learners are trained in parallel, but in boosting, they learn sequentially. This means that a series of models are constructed and with each new model iteration, the weights of the misclassified data in the previous model are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance.

XGBoost: XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. One major problem that xgboost suffers is over fitting and hence we need to be mindfull of that. the over tuning can be reduced by right selection of parameters.

The various hyper-parameters tested for XGBoost are as follows:

- *max_depth*: Maximum depth of the trees
- *n_estimators*: Number of trees in the forest
- *learning_rate*: Step size shrinkage used in update to prevents overfitting
- *min_child_weight*: Minimum sum of instance weight (hessian) needed in a child.
- *gamma*: Minimum loss reduction required to make a further partition on a leaf node of the tree
- *colsample_bytree*: Sub-sample ratio of columns when constructing each tree.

The performance of XGBoost on both multi-classification and binary classification are shown in the table below

Table 4: XGBoost Performance

Vectorisation	Classification	f1 score
TFIDF_vectoriser	binary	0.8718
TFIDF_vectoriser	multi	0.8382

The XGboost model was trained on following features and parameters and gave an f1-score of 0.83

- `objective='multi:softmax`.
- `eval_metric='mlogloss'`
- `n_estimators='100'`
- `total_vectorisation=2214`

The XGboost has yeilded the best f1-score across all model we tried for multiclass problem and hence we plotted the classification report to see performance across each class.

Table 5: XGB Classification Report(MultiClass)

class	Precision	Recall	f1-score	support
age	0.98	0.97	0.98	1557
ethnicity	0.99	0.98	0.99	1627
gender	0.94	0.82	0.88	1626
not-cyberbully	0.64	0.51	0.57	1572
other-cyberbully	0.57	0.81	0.67	1519
religion	0.97	0.93	0.95	1638
weighted-avg	0.85	0.84	0.83	9539

XGBoost is based on weak learners (high bias, low variance). In terms of decision trees Boosting reduces error mainly by reducing bias (and also to some extent variance, by aggregating the output from many models).Although there are certain disadvantage they suffers like Time and computation expensive,Complexity of the classification increase and higher chance of overfitting.

LightGBM:. LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages: faster training speed and higher efficiency, lower memory usage, better accuracy, support of parallel, distributed, and GPU learning, capable of handling large-scale data. In XGBoost, trees grow depth-wise while in LightGBM, trees grow leaf-wise which is the fundamental difference between the two frameworks. The Light gradient boosting model was tried on tf-idf vectoriser with max-features=1500 and we obtained a pretty decent f1-score on our test data.

Table 6: LightGBM Performance

Classification	f1 score
binary	0.8761
multi	0.8382

The binary classification score came out to be highest for light-GBM and so we obtained a class wise classification report.

Table 7: LGBM Classification Report(Binary)

class	Precision	Recall	f1-score	support
not-cyberbully	0.72	0.36	0.48	1572
cyberbully	0.89	0.97	0.93	7967
weighted-avg	0.86	0.87	0.85	9539

Hyper Parameter Tuning

We picked the best performing model for each of the problem binary and multi-class.Apparently the boosting model was dominating in both problems. In binary classification Light GBM performance was highest while in multiclass, Xgboost model was best performer.

XGBOOST:. the Hyper parameters were tuned for Xgboost model in order to reduce overfitting and acheive better f1 score. The parameters of Xgboost after the training were following:`max_depth:6`, `reg_lambda='1'`, `n_estimators='100'`, `learning_rate=0.30`.

LIght-GBM:. We picked the best performing model for each of the problem binary and multi-class. the Hyper parameters were tuned for Xgboost model in order to reduce overfitting and acheive better f1 score. The parameters of Xgboost after the training were following: `max_depth:6`, `reg_lambda='1'`, `n_estimators='100'`, `learning_rate=0.30`

4 CONCLUSION

In conclusion, the project comprised of two main parts. The first was processing the textual data where techniques like lower-casing, handling punctuation, normalize accents stop-word filtering etc. were applied. Secondly, the processed data was fed to different ensemble methods and their performance were compared. The ensemble methods were implemented for both binary and multi-classification. Both LightGBM and XGBoost had similar performances in terms of multi-classification with an F1 score of 0.84 and LightGBM performed marginally better than XGBoost in terms of binary classification with an F1 score of 0.88. In terms of final conclusion , The tree based methods failed to extract the semantic meaning of the text which is the most important criteria for semantic segmentation or text classification.Models like RNN or LSTM has this capability to extract this information and hence they would perform better here.