Malignant Comment Classification Project

Submitted by:

Laxmi Narayan

Acknowledgement

I would like to express my gratitude to my primary SME, **Keshav Bansal**, who guided me throughout this project. I would also like to thank my friends and family who supported me and offered deep insight into the study. I wish to acknowledge the help provided by the technical and support staff in the **Data Science of Flib Robo Technologies**. I would also like to show my deep appreciation to my supervisors who helped me finalize my project.

Introduction

Online platforms when used by normal people can only be comfortably used by them only when they feel that they can express themselves freely and without any reluctance. If they come across any kind of a malignant or toxic type of a reply which can also be a threat or an insult or any kind of harassment which makes them uncomfortable, they might defer to use the social media platform in future. Thus, it becomes extremely essential for any organization or community to have an automated system which can efficiently identify and keep a track of all such comments and thus take any respective action for it, such as reporting or blocking the same to prevent any such kind of issues in the future.

This is a huge concern as in this world, there are 7.7 billion people, and, out of these 7.7 billion, more than 3.5 billion people use some or the other form of online social media. Which means that every one-in-three people uses social media platform. This problem thus can be eliminated as it falls under the category of Natural Language Processing. In this, we try to recognize the intention of the speaker by building a model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate. Moreover, it is crucial to handle any such kind of nuisance, to make a more user-friendly experience, only after which people can actually enjoy in participating in discussions with regard to online conversation.

Problem Statement

The proliferation of social media enables people to express their opinions widely online. However, at the same time, this has resulted in the emergence of conflict and hate, making online environments uninviting for users. Although researchers have found that hate is aproblem across multiple platforms, there is a lack of models for online hate detection.

Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour.

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental illness, self-hatred and suicidal thoughts.

Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate speech, machine learning can be used to fight it. The problem we sought to solve was the tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying.

Evaluation Metrics

→ Label bases metrics include one-error, average precision, etc. These can be calculated for each labels, and then can be averaged for all without taking into account any relation between the labels if exists. Average Precision (AP): Average precision is a measure that combines recall and precision for ranked retrieval results. For one information need, he average precision is the mean of the precision scores after each relevant document is retrieved, where, and are the precision and recall at the threshold.

→ Example based metricsinclude accuracy, hamming loss, etc. These are calculated for each of the examples and then averaged across the test set. Let –

Accuracy is defined as the proportion of correctly predicted labels to the total no. of labels for each instance. Hamming-loss is defined as the symmetric difference between predicted and true labels, divided by the total no. of labels.
When we have a look at the data, we observe that every 1 in 10 samples is toxic, every 1 in 50 samples is obscene and insulting, but the occurrences of sample being severe_toxic, threat and identity hate is extremely rare. Thus, we have skewed data, and accuracy as metric will not give us the required results. Thus, we will be using hamming-loss as the evaluation metric

1.Preparing the data

Before developing any method to distinguish toxic comments from non-toxic ones, there are a few steps to apply on the data itself before doing anything else onto it.

1.1. Standardizing text

We used a few regular expressions to clean up our data, such as removing http links that do not add anything to the toxicity of a comment, and standard- izing numbers, special characters and letters.

To do so, we proceeded as follows:

Source Code 1: Standardizing data function

```
def standardize_text(df, text_field):
    """
    Use a few regexp to clean up data
    """
    df[text_field] = df[text_field].str
    .replace(r"http\S+", "")
    df[text_field] = df[text_field].str
    .replace(r"http", "")
    df[text_field] = df[text_field].str
    .replace(r"@\S+", "")
    df[text_field] = df[text_field].str
    .replace(r"[^A-Za-z0-9(),!?@\'\`\"\_\n]", " ")
    df[text_field] = df[text_field].str
    .replace(r"@", "at")
    df[text_field] = df[text_field].str.lower()
    return df
```

Standardizing our data may impede us to grab some specific intent in some comments but enables us to not create double standards for different comments.

1.2. Deleting stop-words

There is a corpus of stop-words, that are high- frequency words such as "the", "to" and "also", and that we sometimes want to _lter out of a document before further processing. Stop-words usually have little lexical content, do not alter the general meaning of a sentence and their presence in a text fails to distinguish it from other texts.

We used the one from Natural Language Toolkit (1), a leading platform for building Python programs to work with human language.

Methodology

We utilized several classic and state-of-the-art methods, including ensemble learning techniques, with a 80% - 20% split for the training and test data. To reduce the time required for training, we used 500 thousand examples from our dataset. Linear Regression, Random Forest and Gradient Boost were our baseline methods. For most of the model implementations, the open-source Scikit-Learn package [7] was used.

1. Linear Regression

Linear Regression was chosen as the first model due to its simplicity and comparatively small training time. The features, without any feature mapping, were used directly as the feature vectors. No regularization was used since the results clearly showed low variance.

2. Random Forest

Random Forest is an ensemble learning based regression model. It uses a model called decision tree, specifically as the name suggests, multiple decision trees to generate the ensemble model which collectively produces a prediction. The benefit of this model is that the trees are produced in parallel and are relatively

uncorrelated, thus producing good results as each tree is not prone to individual errors of other trees. This uncorrelated behavior is partly ensured by the use of Bootstrap Aggregation or bagging providing the randomness required to produce robust and uncorrelated trees. This model was hence chosen to account for the large number of features in the dataset and compare a bagging technique with the following gradient boosting methods

3. Decision Tree

Decision Tree (DT) is a model that generates a tree-like structure that represents set of decisions. DT returns the probability scores of class membership. DT is composed of: a) **internal Nodes:** each node refers to a single variable/feature and represents a test point at feature level; b) **branches,** which represent the outcome of the test and are represented by lines that finally lead to c) **leaf Nodes** which represent the class labels. That is how decision rules are established and used to classify new instances. DT is a flexible model that supports both categorical and continuous data. Due to their flexibility they gained popularity and became one of the most commonly.

4.KNeighborsClassifier

K Nearest Neighbor(KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms. KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won’t vote. KNN algorithm used for both classification and regression problems. KNN algorithm based on feature similarity approach.

SYSTEM DEVELOPMENT

The system requirements for the algorithms to run efficiently and for the implementation of the
whole idea are:

- Windows 10 (64-bit)
- 8 GBRAM
- Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
- ANACONDA
- Python

Performance Analysis

The main reason for this huge market is that when you buy a New Car and sale it just another day without any default on it, the price of car reduces by 30%.

There are also many frauds in the market who not only sale wrong but also they could mislead to wrong price.

So, here I used this following dataset to Predict the price of any used car.

```
##Importing all necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import warnings
warnings.filterwarnings('ignore')
```

The next step is to import the training data set in the object named train and the testing data set in the object named test.
This import is done by pandas method of reading .csv files into the system.

```
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')
```

Next we check the the features present in the loaded data sets.

```
#checking the null value
print(train.isnull().sum())
print(sns.heatmap(train.isnull()))
```

```
id                  0
comment_text        0
malignant           0
highly_malignant    0
rude                0
threat              0
abuse               0
loathe              0
dtype: int64
AxesSubplot(0.125,0.125;0.62x0.755)
```

In Data Analysis We will try to Find out the below stuff

1. Missing Values in the dataset.

2. All the Numerical variables and Distribution of the numerical variables.

3. Categorical Variables

4. Outliers

5. Relationship between an independent and dependent feature(*selling_price*)

Feature Engineering

You can see what the data looks like, but before using it we need to customize it. We will be performing all the below steps in Feature Engineering

- Handling of Missing values

we'll drop the column if it's not serving in our model or we can use **central tendency measures such as mean, median, or mode** of the **numeric feature.**

- **Substitution of Categorical variables**

To be able to use categorical data, we used **pd.get_dummies()** also called **one hot encoding.** This essentially turns every unique value of a variable into its own binary variable.

Model Building

Now,the development of a model for predicting if the user will apply for a loan or not will start.

Dummies will be used for converting categorical variables into numerical variables because sklearn models allows only numerical inputs.

The train data set will be divided into two parts,80% of the data will act as training data and the remaining 20% data will be the validation data.

First we are spliting the data to train and test for the model

```
: from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 25)
```

Logistic Regression:

We will first build a Logistic Regression model since logistic regression is used for classification problems.

```
# LogisticRegression
LG = LogisticRegression(C=1, max_iter = 3000)

LG.fit(x_train, y_train)

y_pred_train = LG.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = LG.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9595520103134316
Test accuracy is 0.9552139037433155
[[42729   221]
 [ 1923  2999]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.93      0.61      0.74      4922

    accuracy                           0.96     47872
   macro avg       0.94      0.80      0.86     47872
weighted avg       0.95      0.96      0.95     47872
```

Random Forest Algorithm:

Now let us calculate the accuracy using another algorithm called Random Forest.

First we have to set up the model for this algorithm.

```
#RandomForestClassifier
RF = RandomForestClassifier()

RF.fit(x_train, y_train)
y_pred_train = RF.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = RF.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9988003473621071
Test accuracy is 0.9550885695187166
[[42407   543]
 [ 1607  3315]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42950
           1       0.86      0.67      0.76      4922

    accuracy                           0.96     47872
   macro avg       0.91      0.83      0.87     47872
weighted avg       0.95      0.96      0.95     47872
```

Decision Tree:

Now let us calculate the accuracy using another algorithm called Decision Tree.

```
# DecisionTreeClassifier
DT = DecisionTreeClassifier()

DT.fit(x_train, y_train)
y_pred_train = DT.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = DT.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9988092999937331
Test accuracy is 0.9388577874331551
[[41593  1357]
 [ 1570  3352]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97     42950
           1       0.71      0.68      0.70      4922

    accuracy                           0.94     47872
   macro avg       0.84      0.82      0.83     47872
weighted avg       0.94      0.94      0.94     47872
```

KNeighborsClassifier:

Now let us calculate the accuracy using another algorithm called KNeighborsClassifier.

```
#KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=9)
knn.fit(x_train, y_train)
y_pred_train = knn.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = knn.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.9223627785387515
Test accuracy is 0.917697192513369
[[42812   138]
 [ 3802  1120]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     42950
           1       0.89      0.23      0.36      4922

    accuracy                           0.92     47872
   macro avg       0.90      0.61      0.66     47872
weighted avg       0.92      0.92      0.89     47872
```

Conclusion and Future Work

Originally, we first started to think about developing right away a software to solve the given problem, but we quickly realized that we would have to investigate at least a little to know what has already been done in the topic: therefore, we realized a state-of-the-art of existing methods. Then, after providing a comparison of the-then methods, we agreed on one, which was the Long Short Term.
Memory neural networks, first forking an existing solution and afterwards tried to apply our ideas onto it. Eventually, after uploading our solution here (8), we get a score of 0.9772, as we write those lines, and thus get a better score than almost 40% of the entrants of the competition.

As a general conclusion for this project, it was very rewarding for us to take part in this competition because it allowed us to use what we had learned in class directly onto real-life problems, but also to make our own research, to investigate and to try to come up with new ideas when we did not find anything that would suit what we wanted to achieve. It was for all of us an interesting first step into the world of research, and it provided us a hands-on experience of what can be done with machine learning techniques in general.

References

1. S. Bird, E. Klein, and E. Loper, \Natural language processing with python," 2014. http://www.nltk. org/book/ch02.html.

2. A. Robb, “How capital letters be- came internet code for yelling," 2014. https://newrepublic.com/article/117390/netiquette-capitalization\ protect\discretionary{\char\hyphenchar\ font}{}{}how-caps-became-code-yelling

3. J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation," 2018. https://nlp.stanford.edu/projects/ glove/.

4. Ivan, “Lstm: Glove + lr decrease+bn+cv,” 2018. https://www.kaggle.com/demesgal/ lstm-glove-lr-decrease-bn-cv-lb-0-047.

5. A. Srinet and D. Snyder, “Bagging and boosting." https://www.cs.rit.edu/~rlaz/prec20092/ slides/Bagging_and_Boosting.pdf.

6. T. Cooijmans, N. Ballas, C. Laurent, and A. C. Courville, “Recurrent batch normaliza-tion," CoRR, 2017. https://arxiv.org/pdf/ 1603.09025.pdf.

7. A. Pentina and C. H. L. 1, “Multi-task learning with labeled and unlabeled tasks," 2017. http: //pub.ist.ac.at/~apentina/docs/icml17.pdf.

8. Kaggle, “Toxic comment classi_cation chal- lenge," 2018. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/ leaderboard.