TOXICITY DETETCTION IN TWEETS

CSCI 6708: NATURAL LANGUAGE PROCESSING

B00812651 NAINA NIJHER
B00808427 NIRAV SOLANKI

DALHOUSIE UNIVERSITY | Faculty of Computer Science

ABSTRACT

The web is a vast place that has users across various backgrounds of education, culture, ethnicity, religious beliefs, gender. A major issue that comes along with the vast reach of the Internet is cyber bullying by posting toxic content that may include derogatory words or as much as a threat also. It is a challenge to separate such content from the clean content on the social networking websites like facebook, twitter, tumblr. Using the Wikipedia dataset, we have experimented with different models that include important steps like vectorising the comments and transforming them using TFIDF and passing these values into different machine learning models to obtain a model that gives us a high accuracy. After tuning one of the models, we have received an average accuracy of approximately 97.5%. We have also created a module that can be used for extraction of tweets from a particular twitter account to run the model on for prediction. [7]

INTRODUCTION

The dangers of the open Internet have been a major concern as the internet reaches greater parts of the population. Keeping the online conversations constructive is an important task for all the social networking platforms. The people using these platforms come from different cultural, education, religious backgrounds. Difference in opinions may lead to hostile feelings which may lead to verbal assault thereby, causing an unnatural environment. Use of toxic language might lead to a negative impact on the society. Coming across such derogatory language can lead to lower self-esteem and mental illness.

Fighting online harassment, that includes cyber abuse, is a major concern which has been causing an unrest in the online community. Hence, it is critical to come up with an automated model that will be able to successfully predict any such toxic content on the web.[6][8]

According to Wikipedia, hate speech is defined as, "Any speech that attacks a person or group on the basis of attributes such as race, religion, ethnic origin, national origin, gender, disability, sexual orientation, or gender identity." We define offensive language as the text which uses abusive slurs or derogatory terms.

We take two datasets into account to investigate these errors: comments on Wikipedia talk pages presented by Google Jigsaw during Kaggle's Toxic Comment Classification Challenge. The dataset consists of classified labelled data for approximately 150,000 records.

The aim is to keep a check on cyber abuse by evaluating the content extracted from a user's Twitter account. The models implemented makes use of Random Forest Classifier and Naïve Bayes Classifier. The reason behind implementing multiple models was to compare the accuracy amongst these classifiers. After passing the

tweets through these models, the tweets can be classified under these categories – toxic, obscene, threat and identity hate. The tweets can fall under multiple categories depending upon its content.

RELATED WORK

In 2017, Google and Jigsaw launched a project called Perspective, which used probabilistic machine learning methodology to detect toxic content in social media. They were immediately able to detect harassment, online abuse and insults. They even have a platform where user can input some text and the toxicity level of the sentence could be revealed. The main issue with their system was that with slight alternation to the text, toxicity level would be greatly reduced. [1]

Original Phrase (Toxicity Score)	Modified Phrase (Toxicity Score)
Climate change is happening and it's not changing in our favor. If you think differently you're an **idiot**. (84%)	Climate change is happening and it's not changing in our favor. If you think differently you're an **idiiot**. (20%)
They're **stupid**, it's getting warmer, we should enjoy it while it lasts (86%)	They're **st.upid**, it's getting warmer, we should enjoy it while it lasts (2%)
They are liberal **idiots** who are **uneducated** (90%)	They are liberal **i.diots** who are **un.educated** (15%)
idiots. backward thinking people. **nationalists**. not accepting facts. susceptible to **lies**. (80%)	**idiiots**. backward thinking people. **nationaalists**. not accepting facts. susceptible to **l.ies**. (17%)
They are **stupid** and **ignorant** with no class (91%)	They are **st.upid** and **ig.norant** with no class (11%)
It's **stupid** and wrong (89%)	It's **stuipd** and wrong (17%)
If they voted for Hilary they are **idiots** (90%)	If they voted for Hilary they are **id.iots** (12%)
Anyone who voted for Trump is a **moron** (80%)	Anyone who voted for Trump is a **mo.ron** (13%)
Screw you trump supporters (79%)	**S c r e w** you trump supporters (17%)

Here, in the above example just by varying the dot position or slightly altering spelling in few words, toxicity level of the sentence was greatly reduced, even though the meaning of the sentence does not change. [1]

In December 2017, a competition was held by Jigsaw and Conversation AI on Kaggle to find toxic content in comments using dataset prepared by Wikipedia. By the year 2019, more then 4500 teams have participated in the competition. Solutions ranging from HMM to deep learning techniques such as RNN have been presented [2].

Another paper called "Challenges for Toxic Comment Classification: An In-Depth Error Analysis" was also published to address the challenge. They applied Logistic Regression, Convolutional Neural Network and 4 different types of Recurrent Neural Network to detect and classify toxic content. To deal with misspelled words, they have also utilized sub-word embedding. FastText and Glove is used on two different corpuses (Tweeter dataset and Wikipedia dataset) for sub-word embedding to get different results. In the end they were able to achieve more then 98% accuracy for few of the techniques defined above in combination with sub-word embedding obtained from Wikipedia dataset [3].

PROBLEM DEFINITION

Even for humans, identification of text as hate or toxic speech is not an easy task. There are many considerations that need to be kept in mind to classify any such text as derogatory. Another concern is the huge amount of data that the internet generates. Keeping a track of all the data while working on such a large amount is a tedious job. Each second, 6000 tweets are posted on tweeter. 300 hours of content is uploaded to YouTube every minute. 4.75 billion pieces of content are shared on Facebook every day. There are many other social media sites such as Quora, reddit, tumblr, which have Petabytes worth of content. It is impossible to filter toxic content manually. Most of this site have mechanism to filter out swear words. But still many of the ill intended content are not filtered out.

The research question that we address in this work is:

How to effectively identify the class of a new twitter post? After identifying the class of the post, identifying a user to be "toxic"?

To address this question, we have identified the following as our main goal:

- To develop an optimized classification model with an acceptable accuracy.

METHODOLOGY

- ## SYSTEM OVERVIEW

  To build the classification model, we started with a research of the libraries that would be required. We have used the following libraries:

  - import pandas
  - import re
  - from nltk.tokenize import word_tokenize
  - from nltk.corpus import stopwords
  - from nltk.stem import PorterStemmer
  - from sklearn.feature_extraction.text import CountVectorizer
  - from sklearn.feature_extraction.text import TfidfTransformer
  - from sklearn.model_selection import train_test_split
  - from sklearn.ensemble import RandomForestClassifier
  - from sklearn.naive_bayes import MultinomialNB
  - from sklearn.metrics import roc_curve, auc, accuracy_score
  - from sklearn.model_selection import cross_val_score
  - from sklearn.model_selection import GridSearchCV
  - from sklearn.metrics import confusion_matrix

Our system model basically has the following structure:

- Data Pre-Processing that includes data cleaning like removal of any special characters, digits or extra spaces between words. The next step was tokenizing the strings into individual words followed by removing the stop-words belonging to the English dataset. To improve the accuracy of our model, we performed stemming on the entire dataset.
- Feature Extraction that includes vectorizing the comments using Count Vectorizer and balancing out the frequencies by TFIDF Transformation.
- Data Modelling: For the predictions, we used Naïve Bayes and Random Forest Tree Classification. The training data was split into 75% training set and 25% test set. The comments were taken as features and the labels were taken as the classified columns – Toxic, Obscene, etc.
- Model Evaluation: After optimising both the models to give us the maximum accuracy, we tested the models using the Confusion Matrix and Accuracy function from the metrics library and np.mean from numpy.
- Tweets extraction and prediction: The model then extracts tweets from the account of a particular user which can be specified by mentioning the username and then, all the above-mentioned steps are performed on the unlabelled dataset to obtain the prediction.

• ALGORITHMS

We have used two classic machine learning techniques in order to classify toxic content among tweeter tweets:

Naive Bayes

Naive Bayes classifiers are set of supervised machine learning algorithm based on the Bayes theorem. Bayes theorem describes the probability of occurrence of an event based on already known conditions that might affect the event. In Naive Bayes model we determine the class label (output variable) based on one or multiple features (input variables).

$$\mathrm{P}(V_1|V_2, V_3, \ldots, V_n) = \frac{\mathrm{P}(V_2, V_3, \ldots, V_n|V_1) \cdot \mathrm{P}(V_1)}{\mathrm{P}(V_2, V_3, \ldots, V_n)}$$

Bayes Theorem [4]

Naive Bayes assumes that all the features are independent of each other. Hence, the name “Naive”. So, the above equation becomes:

$$\begin{aligned} \mathrm{P}(V_1|V_2, V_3, \ldots, V_n) &= \frac{\mathrm{P}(V_2, V_3, \ldots, V_n|V_1) \cdot \mathrm{P}(V_1)}{\mathrm{P}(V_2, V_3, \ldots, V_n)} \\ &= \frac{\mathrm{P}(V_2|V_1) \cdot \mathrm{P}(V_3|V_1) \cdot \ldots \cdot \mathrm{P}(V_n|V_1) \cdot \mathrm{P}(V_1)}{\mathrm{P}(V_2, V_3, \ldots, V_n)} \end{aligned}$$

[4]

In case of our project, the output variables are Labels (Toxic, Hate Speech, Insult, Obscene, Threat) and for the input variables TFIDF vectors of comments are used. We calculate whether a specific comment will fall under the given label. For each label a separate model is constructed. All the labels are completely independent of each other and a comment can fall under more than one label.

Random Forest Tree

Random Forest tree is also a supervised classification algorithm. It is an ensemble method in which decision trees are generated by randomly splitting the dataset. The decision trees are collectively known as forest. Each tree in the forest is based on independent data sample. The trees are generated using attribute selection indicator such as information gain, gain ratio, and Gini index [5].

During classification, each tree in the forest vote is used for determining the class. The count for each class label is calculated using vote cast by each tree. In the end the class label with the highest label is chosen as the final classification. The accuracy of this model depends on the number of trees present in the forest. Increased number of trees prevents Biased decision due to overfitting [5].

Same case as above Toxic, Hate speech, Insult, Obscene and Threat are used as class labels. We build a separate model for each of this class and using the model calculate whether a new comment can fall under the given class. During the model construction, comments are first transformed into TFIDF vectors. These vectors are randomly split to make different decision trees. To check whether a new comment falls under a given class, first it is transformed into TFIDF vectors. The vectors are run through the decision trees created during training stage. Each tree votes whether the given comment falls under the given class or not. In the end the votes are counted and the class with the majority voting is taken as the final verdict.

EXPERIMENT DESIGN

```
df = pd.read_csv("D:/Semester 2/Natural Language Processing/trainSplit.csv", sep = ",")
df["comment_text"]= df["comment_text"].str.lower()
df["comment_text"] = df["comment_text"].str.replace(r"\d+","")
df["comment_text"] = df["comment_text"].str.replace(r"[^a-z]+"," ")
df["comment_text"] = df["comment_text"].str.strip()
```

Image 1: Data Pre-Processing (Data Cleaning) steps

```
stopset = set(stopwords.words('english'))
df["commentText_noStopwords"] = df["comment_text"].apply(lambda x: ' '.join([word for word
                                                    in x.split() if word not in (stopset)]))
#df["commentText_noStopwords"].head()

df["tokenized_comment_text"] = df.apply(lambda row: nltk.word_tokenize
                                        (row["commentText_noStopwords"]), axis=1)
#df["tokenized_comment_text"].head()

stemmer = PorterStemmer()
df["stemmed"] = df["tokenized_comment_text"].apply(lambda x: [stemmer.stem(y) for y in x])
df['stemmed'] = df['stemmed'].apply(' '.join)

corpus = (df['stemmed'])
```

Image 2: Data Pre-Processing steps

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X).toarray()

y_toxic = list(df['toxic'])
```

Image 3: Feature Extraction

```
X_train, X_test, ytoxic_train, ytoxic_test = train_test_split(X_tfidf, y_toxic, train_size=0.75, test_size=0.25,
                                                              random_state=1)

model_NB = MultinomialNB(alpha = 1.0, class_prior=None, fit_prior=False).fit(X_train,ytoxic_train)

model_RF = RandomForestClassifier(criterion= 'gini', max_depth= 75, min_samples_leaf= 1, min_samples_split= 2,
                                  n_estimators= 20).fit(X_train,ytoxic_train)
```

Image 4: Model Implementation

```
predict_NB = model_NB.predict(X_t)
predict_RF = model_RF.predict(X_t)

print(np.mean(predict_NB == ytoxic_test))
print(np.mean(predict_RF == ytoxic_test))

print(confusion_matrix(ytoxic_test, predict_NB))
print(confusion_matrix(ytoxic_test, predict_RF))
```

Image 5: Accuracy Evaluation using confusion matrix and np.mean

Naïve Bayes Classifier Accuracy and Confusion Matrix:

```
insult:
0.9478696741854636
[[1860   41]
 [  63   31]]

identity_hate:
0.9774436090225563
[[1948   35]
 [  10    2]]

obscene:
0.9493734335839599
[[1851   43]
 [  58   43]]

toxic:
0.9233082706766917
[[1769   38]
 [ 115   73]]

threat:
0.9794486215538847
[[1954   36]
 [   5    0]]
```

Random Forest Classifier Accuracy and Confusion Matrix:

```
insult:
0.9624060150375939
[[1897    4]
 [  71   23]]

identity_hate:
0.9939849624060151
[[1983    0]
 [  12    0]]

obscene:
0.9714285714285714
[[1889    5]
 [  52   49]]

toxic:
0.9268170426065163
[[1803    4]
 [ 142   46]]

threat:
0.9974937343358395
[[1990    0]
 [   5    0]]
```

CONCLUSION

Keeping a check on the toxicity content in on the web is the need of the hour as it might lead to hostility among users. Thus, having an automated system that would take care of predicting and identifying a post as toxic or containing any kind of derogatory text is vital.

Also, from our models implemented there are few conclusions that we could draw:

- On a comparatively smaller dataset, both the models averaged to approximately the same accuracy however, while working with a larger dataset, Random Forest Classifier provided better accuracy that Naïve Bayes across all the categories.
- The time taken by Random Forest Classifier for prediction is significantly more than the time taken by Naïve Bayes Classifier.
- In some cases, Random Forest Classifier closed in to an accuracy of about 99% also. Accuracy of both the models were above 92% for all the labels.

REFERENCES

[1] H. Hosseini, S. Kannan, B. Zhang and R. Poovendran, "Deceiving Google's Perspective API Built for Detecting Toxic Comments", *arXiv*, vol. 1702, no. 08138, p. 4, 2017 [Online]. Available: https://arxiv.org/abs/1702.08138. [Accessed: 13- Apr- 2019]

[2] "Toxic Comment Classification Challenge | Kaggle", *Kaggle.com*, 2019. [Online]. Available: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge. [Accessed: 16- Apr- 2019]

[3] B. van Aken, J. Risch, R. Krestel and A. Loser, "Challenges for Toxic Comment Classification: An In-Depth Error Analysis", 2019 [Online]. Available: https://aclweb.org/anthology/W18-5105. [Accessed: 16- Apr- 2019]

[4] V. Keselj, *Web.cs.dal.ca*, 2019. [Online]. Available: https://web.cs.dal.ca/~vlado/csci6509/notes/nlp14.pdf. [Accessed: 16- Apr- 2019]

[5]"Random Forests Classifiers in Python", *DataCamp Community*, 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/random-forests-classifier-python. [Accessed: 16- Apr- 2019]

[6]*Beta.vu.nl*, 2019. [Online]. Available: https://beta.vu.nl/nl/Images/werkstuk-biere_tcm235-893877.pdf. [Accessed: Apr- 2019].

[7]*Arxiv.org*, 2019. [Online]. Available: https://arxiv.org/pdf/1809.08651.pdf. [Accessed: Apr- 2019].

[8]2019. [Online]. Available: https://www.researchgate.net/publication/322577160_Detecting_Offensive_Language_in_Tweets_Using_Deep_Learning. [Accessed: 16- Apr- 2019].