Brandeis University
LING 131A
Final Project
Bin Liu / Shuoge Shen

Toxic Comment Classification

Prerequisites and Usage

Make sure you have the following packages installed before running the Main.py:

Numpy, Sklearn, wxpython

1. Introduction

Nowadays many people took a huge amount of time surfing on the Internet, mostly using social media. People may make comments, post their opinions, write articles, and even play games on the internet. Since the Internet has become a major part of people's life, it inevitably needs some regulations on how people behave in the virtual world, because people may become "bolder" without facing a real person. In this situation, some people may not respect other people, and even say some toxic words and prohibit others from enjoying the social media. Therefore, some people are reluctant to share their opinion online in order to avoid verbal attacks, while some even make reprisals against whom posting the toxic comments. This can lead to a vicious circle of attacking and retaliating.

Online information is exploding and it's impossible to detect those comments by humans or waiting for people to report, so people may just leave the discussion before we have time to process them manually. For example, some direct humiliating words are easy to detect by using certain keywords. However, other unobvious toxic comments are usually hiding under table due to the complexity of language and can only be understand after the whole comments are read.

That's the reason why we wish to create a model that can classify different kind of toxity of comments, so that when people post toxic comments, the model can detect it and make quick responses. In this case, the machine can filter out most of the common or uncommon toxic words before people reach them. And therefore, a more friendly online environment can be achieved. Above all, our goal is to build a model that can process a short paragraph of comments and detect levels of different kinds of toxity.

2. Datasets

Our data comes from the Kaggle Competition of “Toxic Comment Classification Challenge”: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview.

2.1 Training and Testing data

The datasets are separated into the training and testing portions, with 16000 records used for training while another 4000 used for testing. Each record contains text and the corresponding label. The texts are original comments that are posted online. On the other hand, labels are binary with 0 (absent) or 1 (present). They are further divided into six levels in an increasing toxity order: toxic, severe toxic, obscene, threat, insult and identity hate. An example of the data is as follows:

id	comment_text	toxic	severe_toxic	obscene	threat	insult	Identity_hate
0	Why the edits made under my…	0	0	0	0	0	0
1	Hey man, I'm really…	1	0	0	0	1	0

3. Model Explanation

3.1 Preprocessing

The training and testing data were preprocessed in order to produce quality output of modeling. The data were cleaned by removing unnecessary texts and structuring words. The panda and nltk package were used for extracting data, tokenizing texts, removing stopwords and punctuation, and lemmatizing words. The class Preprocess was created and can be initialized by passing the original training and testing datasets:

```
myPreprocess = Preprocess(train_comments, test_comments)
```

And by using the clean() method in the object myPreprocess, we can get the cleaned training and testing data:

```
clean_train_comments, clean_test_comments = myPreprocess.clean()
```

Then, we can proceed to the step of building vectors.

2.3 Vectorizing

The features (or comment text) in datasets need to be encoded for the purpose of modeling. The TFidfVectorizer from sklearn package was used to convert the comment text into the vector space as a matrix of TFIDF features for measuring the importance of a word in text.

By calling the vectorizer() method in myPreprocess and passing the cleaned datasets, we obtained the training and testing data vector:

```
X, x_test = myPreprocess.vectorizer()
```

3. Logistic Regression Models

A Logistic Regression model in sklearn package was selected for modeling the datasets.

The logistic regression model was used due to the following reasons:

- Our labeled values are dichotomous in nature (present or absent)

- We vectorized our features and wished to explain the problem linearly

3.2 Modeling

We enumerated the six-levels toxity labels: toxic, severe toxic, obscene, threat, insult and identity hate. And for each of the level, we created an individual logistic regression model using the vectorized data, for example

```
lr = LogisticRegression(C=2,random_state = i,class_weight = 'balanced')
lr.fit(X,y[“severe toxic”])
```

3.3 Results

The testing accuracy of this model are around 0.4 in average, which is not an ideal output result. Therefore, we decided to improve our model by using more accurate datasets.

4. Model Enhancement

4.1 Reasons of bad performance

In the last model, the accuracy is a bit low so that we can't give a good estimate of the toxity scores. The problem may be the lemmatization, there're two reasons. The first is that lemmatization normalizes the word, which will cause loss of information, because two different words may become the same after the operation. The other reason is that lemmatization requires the comments to have correct grammar, but in many cases, people don't spell that correct on the Internet.

4.2 TFIDF vectorizer explanations

To make some improvements, we removed the part of lemmatization and utilized more features of TfidfVectorizer to do the work The logistic regression model was improved by redefining the parameters in TFidfVectorizer according to several experiments and estimations. The ngrams features were involved as the additional features in our datasets. For instance, for the sentence "I think the movie is bad", it can be separated into the 1-gram and 2-grams features {'I', 'think', 'the', 'movie', 'is', 'bad', 'I think', 'think the', 'the movie', 'movie is', 'is bad'}. By including, for example, the unigram and bigrams to the model, sometimes we can capture the more important information that expressed by multiple tokens. Therefore, we modified the parameters in TFidfVectorizer:

1. stop_words: instead of creating a stop words class by ourselves, we used the embedded stop_words parameter to remove the stop words.

2. lowercase: it will convert all characters into lowercase before tokenizing

3. analyzer = 'word': the features are transformed into word n-grams

4. ngram_range(1, 3): to ultilize more information, we used unigram, bigram, and trigram for the data set. An example of the ngram features are:

'browse'	'browser'	'burn hell'	'bush hell god'
'chiefs'	'child'	'calling vandal'	'cheers talk contribs'

5. max_features: we chose the top 8000 frequent features to train the data.

4.3 Output

Sample output after the TFIDF tranformation:

```
  (0, 2805)    0.17757114808135513
  (0, 2566)    0.1268546263299676
  (0, 7451)    0.1828395469079643
  (0, 3411)    0.2514248762684242
  (0, 2896)    0.19863253960582403
  (0, 6092)    0.15365700300727214
  (0, 7720)    0.21608469676435169
  (0, 4010)    0.09371392638121552
  (0, 3203)    0.24938951945737797
  (0, 7610)    0.2393464171487032
  (0, 4792)    0.12665285286654537
  (0, 7985)    0.2028437304524112
  (0, 2837)    0.2306341909962617
```

Assuming the format of the output is "(X, Y) Z", X means the document index. We only have one document of data, thus the index of X is always 0. Y is the specific word-vector index, which is the index of the n-grams we created. C is the TFIDF score for that specific word-vector.

4.4 Logistic Regression

There are some 3 parameters that we used for the logistic regression model:

C: inverse of regularization, which is used to apply a penalty to increasing the magnitude of parameter values in order to reduce overfitting. Because we have a lot of parameters but the input data may be just a sentence, so that may cause overfitting when training the data. To minimize the error, we chose C to be 2 to give some penalties to the large values of the parameters.

random_state: specify a random state for each category to ensure that every time we run the model it will have the same results

class_weight: we chose 'balanced' because it will adjust weights inversely proportional to class frequencies in the input data according to the documentation.

4.5 Prediction Result

The accuracy score for each of the category of toxity is as follows:

Toxic Accuracy Score	0.915
Severe Toxic Accuracy Score	0.978
Obscene Accuracy Score	0.952
Threat Accuracy Score	0.978
Insult Accuracy Score	0.954
Identity Hate Accuracy Score	0.979

5. Interfaces

A user-friendly interface was designed in Python in order to facilitate the use of the model. The wxpython package was employed to develop the interface. The interface is presented as follows

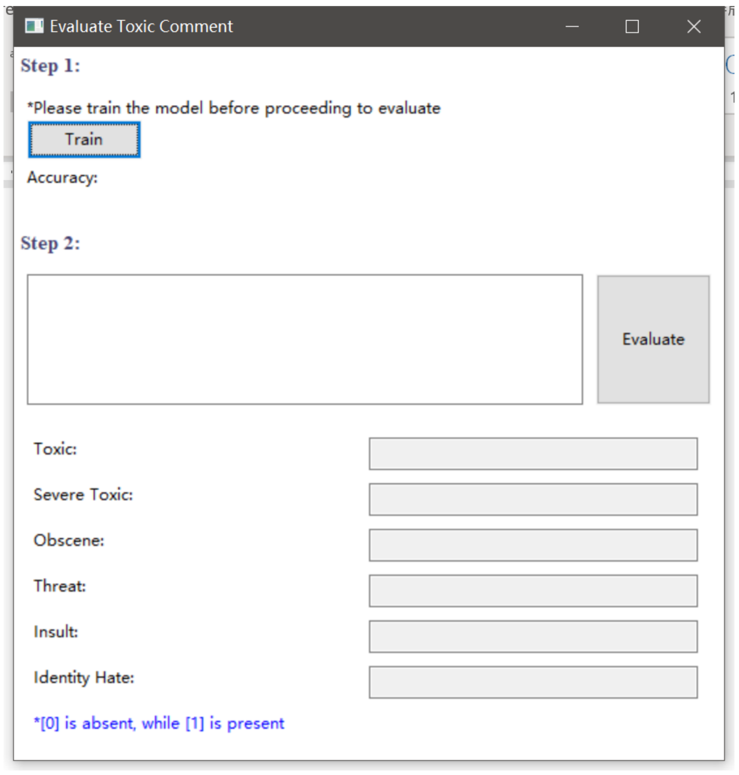
To open this interface, please download wxPython first using the following command:

```
pip install -U wxPython
```

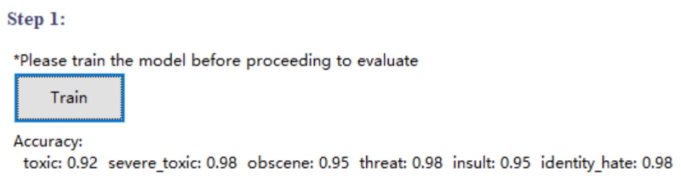
To run the program, enter the following command while in the folder 'Scripts':

```
pythonw main.py
```

The following interface should show up:



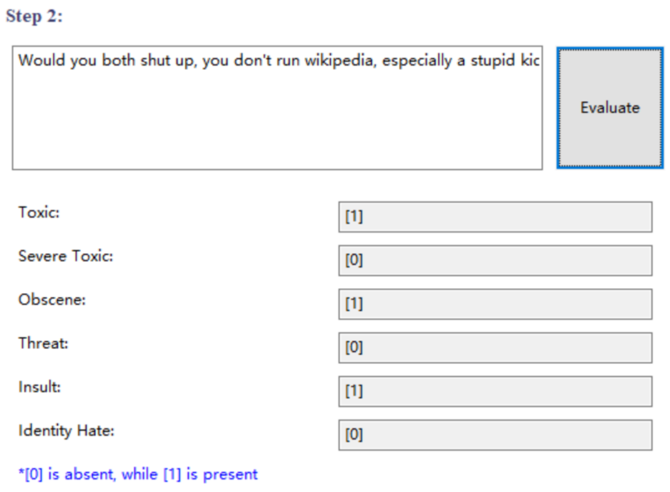
Step1: user have to train the model before evaluating their own comments. By pressing the 'Train' button, a progress bar will show up and after that, it will show the accuracy of each level of toxity.



```
toxic: 0.92  severe_toxic: 0.98  obscene: 0.95  threat: 0.98  insult: 0.95  identity_hate: 0.98
```

Step 2: the comment used to evaluate can be enter in the textbox, and by pressing the

‘Evaluate’ button, the results will be presented in the lower portion of interface.


Step 2:

Would you both shut up, you don't run wikipedia, especially a stupid kic

Toxic:	[1]
Severe Toxic:	[0]
Obscene:	[1]
Threat:	[0]
Insult:	[1]
Identity Hate:	[0]

*[0] is absent, while [1] is present

Where [0] stands for certain kind of toxic comments do not exist, while [1] stands for it exists. In above example, the comment contains toxic language, obscene and identity hate.

6. Conclusion

In this project, we presented a way of identifying toxic language in text, especially in the comments that posted on internet. We showed that the use of logistic regression model and ngrams features together can produce a satisfied model in detecting toxic comments. Furthermore, by designing the an easy-to-use interface, user can easily train and evaluate the online comments.