PROJECT REPORT

ON

TOXIC COMMENTS CLASSIFICATION IN SOCIAL NETWORKING

By

SUDHA SREE VODNALA

ON

DEEP LEARNING

Under the guidance

Smart Bridge

Mentor/project guide: Mahankali Surya Tej

Toxic Comments Classification In Social Networking

Project Overview

Nowadays, the flow of data over the internet has grown dramatically, especially with the appearance of social networking sites. Social networks sometimes become a place for threats, insults, and other components of cyberbullying. A huge number of people are involved in online social networks.
Toxic comments are textual comments with threats, insults, obscene, racism, etc. In recent years there have been many cases in which authorities have arrested some users of social sites because of the negative (abusive) content of their personal pages. Hence, the protection of network users from anti-social behaviour is an important activity. One of the major tasks of such activity is automated detecting the toxic comments.

Bag of words statistics and bag of symbols statistics are the typical source information for the toxic comments detection. Usually, the following statistics-based features are used: length of the comment, number of capital letters, number of exclamation marks, number of question marks, number of spelling errors, number of tokens with non-alphabet symbols, number of abusive, aggressive, and threatening words in the comment, etc. A neural network model is used to classify the comments.

Problem Statement

Given a group of sentences or paragraphs, which was used as a comment by a user in an online platform, our task is to classify it to belong to one or more of the following categories – toxic, severe-toxic, obscene, threat, insult or identity-hate with approximate probabilities or discrete values .This is a Multi-label Classification problem.

A multi-label classification problem differs from a multi-class classification problem (in which each sample can only be assigned to one of the many-labels). Hence in our problem each comment can belong to more than one label for eg, a comment can be obscene, toxic and insult, all at the same time.

Solving a multi-label classification problem

1.One way to approach a multi-label classification problem is to transform the problem into separate single-class classifier problems. This is known as 'problem transformation'.

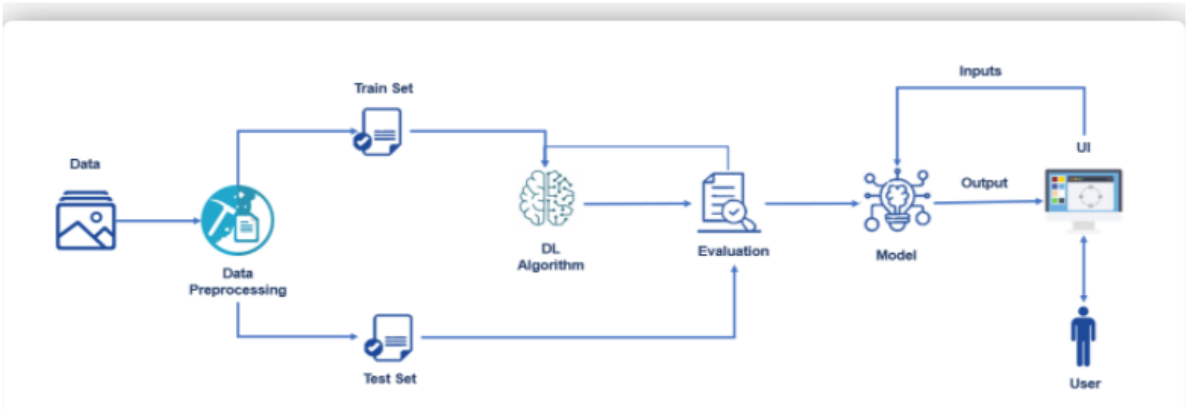
There are three methods:

i. Binary Relevance: This is probably the simplest which treats each label as a separate single classification problems. The key assumption here though, is that there are no correlation among the various labels.

ii. Classifier Chains: In this method, the first classifier is trained on the input X. Then the subsequent classifiers are trained on the input X and all previous classifiers' predictions in the chain. This method attempts to draw the signals from the correlation among preceding target variables.

iii. Label Powerset: This method transforms the problem into a multi-class problem where the multi-class labels are essentially all the unique label combinations. In our case here, where there are six labels, Label Powerset would in effect turn this into a 2^6 or 64-class problem.

Architecture:



Data Exploration

The dataset consists of the following fields-

Id: A hexadecimal value, to identify the person who had written this comment

Comment _text: A multi-line text field containing the exact comment

Toxic: binary label containing 0/1

Severe _toxic: binary label containing 0/1

Obscene: binary label containing 0/1

Threat: binary label containing 0/1

Insult: binary label containing 0/1

Identity _hate: binary label containing 0/1

Out of these fields, the comment_text field will be preprocessed and fitted into different classifiers to predict whether it belongs to one or more of the labels/outcome variables.

```
train_df = pd.read_csv('Downloads/Data/Data/train.csv')
test_df = pd.read_csv('Downloads/Data/Data/test.csv')
train_df.head()
```

	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

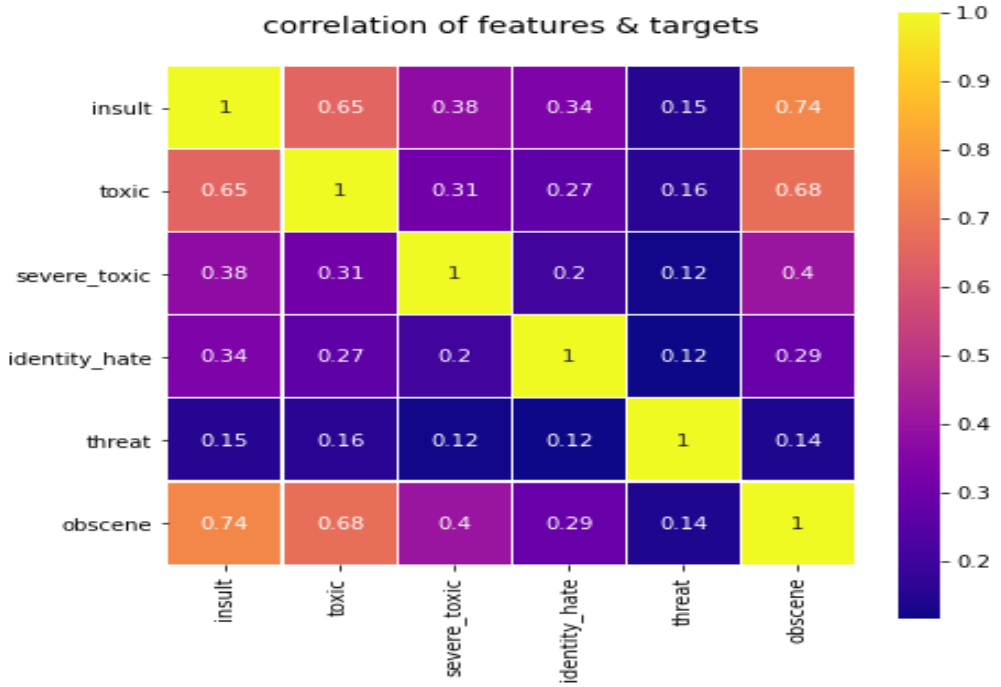
Checking for null comments

```
# check for null comments in test_df
print(test_df.isnull().any())
```

```
id              False
comment_text    False
dtype: bool
```

Correlations

Correlation analysis is a statistical method used to evaluate the strength of relationship between two quantitative variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related



Indeed, it looks like some of the labels are higher correlated, e.g.insult-obscene has the highest at 0.74, followed by toxic-obscene and toxic-insult. This co-relation can be taken in use for other kind of algorithms, such as multi-chain labelling

Data Preprocessing

Preprocessing involved the following steps:

✣ **Preparing a string containing all punctuations to be removed :**

1. The string library contains punctuation characters. This is imported and all numbers are appended to this string.

2. Also, we can notice that our comment_text field contains strings such as won't, didn't, etc which contain apostrophe character('). To prevent these words from being converted to wont/didnt, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.

✣ Applying Count Vectoriser :

1. Count Vectoriser is used for converting a string of words into a matrix of words with column headers represented by words and their values signifying the frequency of occurrence of the word.
2. It accepts stop words, convert to lowercase(set as true), and regular expression as its parameters. Here, we will be supplying our custom list of stop words created earlier and using lowercase option. Regular expression will have its default value.

```
# Define all_text from entire train & test data for use in tokenization by Vectorizer
train_text = train_df['comment_text']
test_text = test_df['comment_text']
all_text = pd.concat([train_text,test_text])

# Vectorize the data
# import and instantiate CountVectorizer

from sklearn.feature_extraction.text import CountVectorizer
word_vect = CountVectorizer(
                            strip_accents='unicode',
                            analyzer='word',
                            token_pattern=r'\w{1,}',
                            stop_words='english',
                            ngram_range=(1,1)
                            )
```

Binary Relevance - build a multi-label classifier using Logistic Regression

Model Building and Fitting

Import and instantiate the Logistic Regression model. The model is trained using the train data and then the training accuracy is computed.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
logreg = LogisticRegression(C=12.0)


mapper = {}
for label in cols_target:
    mapper[label] = logreg
    filename = str(label+'_model.sav')
    print(filename)
    print('...processing{}'.format(label))
    y = train_df[label]
    # train the model using train_features & y
    mapper[label].fit(train_features,y)
    # train the model using train_features & y
    pickle.dump(mapper[label], open(filename, 'wb'))
    # compute the training accuracy
    y_pred_X = mapper[label].predict(train_features)
    print('Training Accuracy is {}'.format(accuracy_score(y,y_pred_X)))
    # compute the predicted probabilities for X_test_dtm
    test_y_prob = mapper[label].predict_proba(test_features)[:,1]

```

Application Building

Building an Application to integrate the model

After the model is built, we will be integrating it into a web application so that normal users can also use it to know if any comment is toxic or not. In the application, the user provides any text(comment) to check and analyzed by calling the different label model files generated during cleaning. Finally, the toxicity of the comment is displayed on the UI.

Flask App

Build the python flask app

In the flask application, the comment is taken from the HTML page and it is cleaned and processed. Using this processed text predictions are performed and are sent back to the HTML page to notify the user.

Step - 1: Import the required libraries

```
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer
import pickle
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from flask import Flask, request, jsonify, render_template, url_for
```

Step - 2: Load the count vectorizer and initialize flask app

```
loaded=CountVectorizer(decode_error='replace',vocabulary=pickle.load(open('word_features.pkl','rb')))

app = Flask(__name__)
```

Step - 3: Configure app.py to fetch the URL from the UI, process the text, and return the prediction

```python
def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "cannot ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text

@app.route('/')
def landingpage():
    img_url = url_for('static',filename = 'images/hello.png')
    print(img_url)
    flag=0
    return render_template('toxic.html',flag=flag)
```

```
@app.route('/predict')
@app.route('/', methods = ['GET','POST'])
def predict():
    if request.method == 'GET':
        img_url = url_for('static',filename = 'images/hello.png')
        return render_template('toxic.html',url=img_url)
    if request.method == 'POST':
        comment = request.form['comment']
        new_row = {'comment_text':comment}
        user_df = pd.DataFrame(columns = ['comment_text'])
        user_df = user_df.append(new_row,ignore_index = True)
        user_df['comment_text'] = user_df['comment_text'].map(lambda com : clean_text(com))
        user_text = user_df['comment_text']
        user_features = loaded.transform(user_text)
        cols_target = ['obscene','insult','toxic','severe_toxic','identity_hate','threat']
        lst= []
       # mapper = {}
        for label in cols_target:
            filename = str(label+'_model.sav')
            filename
            model = pickle.load(open(filename, 'rb'))
            print('... Processing {}'.format(label))
            user_y_prob = model.predict_proba(user_features)[:,1]
            print(label,":",user_y_prob[0])
            lst.append([label,user_y_prob])
        print(lst)

        final=[]
        flag=0
        for i in lst:
            if i[1]>0.5:
                final.append(i[0])
                flag=2
        if not len(final):
            text = "Yaayy!! The comment is clean"
            img_url = url_for('static',filename = 'images/happy.png')
            flag=1
            print(img_url)
        else:
            text="The comment is "
```

```
    else:
        text="The comment is "
        for i in final:
            text = text+i+" "
        img_url = url_for('static',filename = 'images/toxic.png')
    print(text)
    return render_template('toxic.html',ypred = text,url= img_url,flag=flag)
```

Step 4: Run the app

```
if __name__ == '__main__':
    app.debug=True
    app.run(host = 'localhost', debug = True , threaded = False)
```

Build An HTML Page

We Build an HTML page to take the text from the user and upon clicking on the button for submission it has to redirect to the URL for “**y_predict**” which returns if the text given is toxic or safe. The output is to be then displayed on the page.

The HTML pages are put under the templates folder and any style sheets if the present is put in the static folder.

Execute And Test The Model

Now we execute the model using Anaconda Prompt

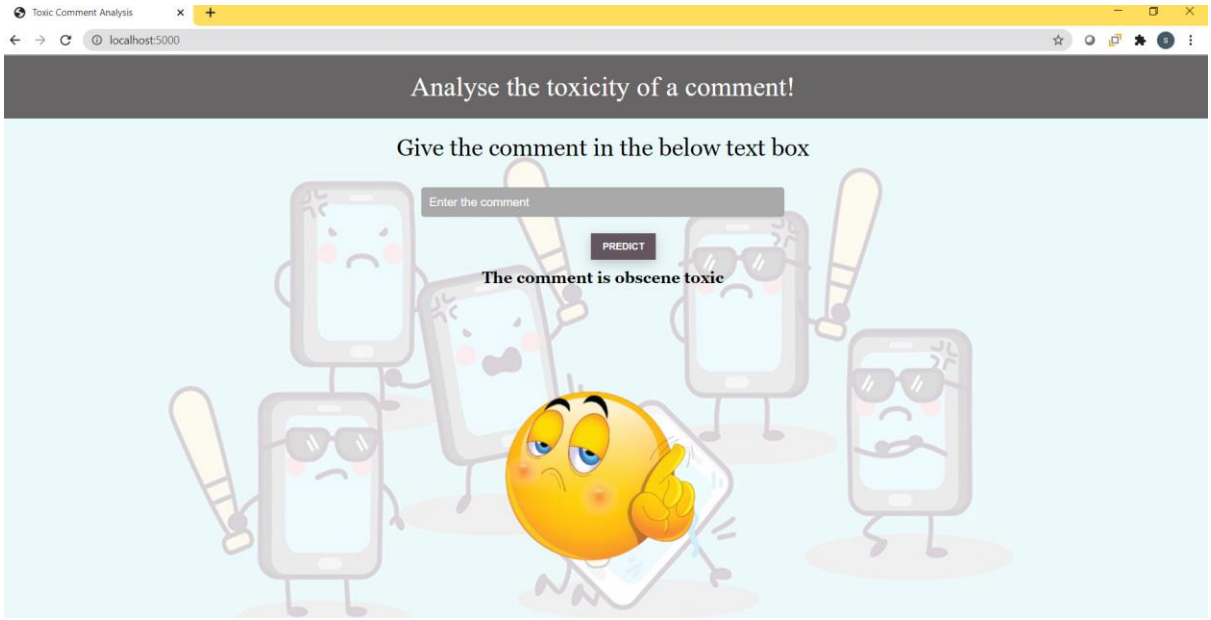
Execute the python code and after the module is running, open the localhost page and give the text to be predicted or tested.

```
(base) C:\Users\HP>cd C:\Users\HP\Downloads\Commet-Toxicity-Multi-Class-Classification-main\Commet-Toxicity-Multi-Class-Classification-main

(base) C:\Users\HP\Downloads\Commet-Toxicity-Multi-Class-Classification-main\Commet-Toxicity-Multi-Class-Classification-main>cd Flask

(base) C:\Users\HP\Downloads\Commet-Toxicity-Multi-Class-Classification-main\Commet-Toxicity-Multi-Class-Classification-main\Flask>python commentApp.py
* Serving Flask app "commentApp" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 118-409-284
* Running on http://localhost:5000/ (Press CTRL+C to quit)
```

For a toxic comment, the prediction is:



For the comment "it is amazing", the prediction is: