TOXIC COMMENTS CLASSIFICATION CHALLANGE

Sambit Nath

02/13/2021

About the business use case:

The use case has been selected from Kaggle site. Here we need to build a multi-headed model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate.

Further details regarding this use case has been described over this link : https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview.

The project is mainly divided into 2 sections namely:

1. Data preparation and training.
2. Testing results.

Data Preparation and training: We are given a set of labelled data/ comments which has labels for 6 different types of toxicity as per the given use case. There are 159572 number of comments in our training data to be exact.

I have taken only a sample of 75000 comments from the total training set for my project because I just wanted my process to take lesser time to train, plus my local system does not have that high computational speed to process that much data in less amount of time.

So, after taking the sample of 75000 from the original train.csv file, I followed the 'bag of words' approach to solve this problem set. First, I created a list of all the comments and applied the following processing on each of those comments:

1. Applied regular expression to filter out only the spaces and the alphabets and the words (in small and capitals) and remove the unwanted punctuations.
2. Converted them to lower cases.
3. Applied stemming of the words (e.g removing tenses of the words such as 'running' converted to 'run'). There is another technique called lemmatizing which is used in chatbots.
4. Filtered out the stop words which would not contribute to the sentiment of a particular comment (such as the, this, on etc.)
5. Collected all the processed comments into a corpus.

Once this was done the corpus was converted into a sparse matrix to create the bag of words model. After that, the labels for the different toxicity types were checked for imbalances. I had set some class

weights accordingly as per the label imbalances for each class for the different toxicity types.

Lastly, I trained the model using Random Forest classifier algorithm. There is a parameter called class weight where I plugged in the class weights for each of the types of toxicities and saved each of those models in the system as their serialized versions.

Testing results: In this section I used the testing dataset and performed the same pre-processing to convert the set of words into a bag of words. After that, I used the pretrained models to predict the results and generate classification report, confusion matrix and other metrics to check the scores.

Code flow details:

I have divided the project into 2 scripts for each section as discussed earlier:

1. data preprocessing and training.py
2. testing results.py

1. **data preprocessing and training.py :** In this script the labelled comments for different toxicities were pre-processed as described previously. After that it was converted into a sparse matrix to make a bag of words model using the CountVectorizer module. I set the max_features parameter in CountVectorizer to be 5000 to take only the top 5000 most occurring words in the sparse matrix.
   The imbalance in each of the toxicity type labels were checked and accordingly I initialized a dictionary which stored the class weights.
   The models were trained for each toxicity type as per the class weights. These models were stored in the dictionary model_dict as key-value pairs which was further saved in local system using the joblib library to convert to a serialized version of those.

2. **testing results.py**: In this script the test dataframe was imported and a sample of 1000 were taken for generating the metrics for the trained models.
   The confusion matrix, accuracy score and the classification report were printed for each of the models and displayed. Below are some of the metrics for a few toxicities.:

```
These are the metrics for the comment: toxic
[[654 262]
 [ 63  21]]
0.675
              precision    recall  f1-score   support

           0       0.91      0.71      0.80       916
           1       0.07      0.25      0.11        84

    accuracy                           0.68      1000
   macro avg       0.49      0.48      0.46      1000
weighted avg       0.84      0.68      0.74      1000
```

```
These are the metrics for the comment: obscene
[[790 168]
 [ 39   3]]
0.793
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       958
           1       0.02      0.07      0.03        42

    accuracy                           0.79      1000
   macro avg       0.49      0.45      0.46      1000
weighted avg       0.91      0.79      0.85      1000
```

```
These are the metrics for the comment: insult
[[904  62]
 [ 31   3]]
0.907
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       966
           1       0.05      0.09      0.06        34

    accuracy                           0.91      1000
   macro avg       0.51      0.51      0.51      1000
weighted avg       0.94      0.91      0.92      1000
```

These results can further be improved by tweaking the hyperparameters or by adopting other models such as Naïve Bayes, Neural networks etc.

The class imbalance could also be handled using various other up-sampling (e.g SMOTE) and down-sampling techniques (e.g Near miss).