Name: Shu-Ya Chiang
DSC 478 Final project
Instructor: Aleksandar Velkoski

Executive Summary

The goal is to find the best performance on text classification using three methods, SGDClassifier, Navies Bayes, logistic regression. Also, I compared different data cleaning approaches like NLTK and TfidfVectorizer. Using TfidfVectorizer is easier and has the best accuracy score and AUC, but it doesn't make sense to use TfidfVectorizer. It's an optimistic result because it transforms the text to vector before train test split, which it's different from the data in the real world. Doing train test split before transform text to vector makes more logical sense, so NLTK is a better approach to clean text data. In training models part, Navies Bayes uses the least time to get the result, but the accuracy is the worst. Comparing SGDClassifier, Navies Bayes, and logistic regression, the best procedure in my opinion is to use NLTK cleaning text data, then to split train and test. After splitting data, I would do countvectorizer to transform text to vector, then apply SGDClassifier to build models and get prediction. The further work can use NLP approaches to conduct identity columns, and it can reduce some bias and has a better performance on detecting toxic comments.

Main report

Dataset:

Source: https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/data
1. The size of train.csv is (1804874, 45). 1804874 observations and 45 variables.
- Variables: Id, target, comment_text, severe_toxicity, obscene, identity_attack, Insult, threat, Asian (77.55% missing), atheist (77.55% missing), bisexual (77.55% missing), black (77.55% missing), buddhist (77.55% missing), christian (77.55% missing), female (77.55% missing), hindu (77.55% missing), heterosexual (77.55% missing), homosexual_gay_or_lesbian (77.55% missing), intellectual_or_learning_disability (77.55% missing), jewish (77.55% missing), latino (77.55% missing), male (77.55% missing), muslim (77.55% missing), other_disability (77.55% missing), other_gender (77.55% missing), other_race_or_ethnicity (77.55% missing), other_religion (77.55% missing), other_sexual_orientation (77.55% missing), physical_disability (77.55% missing), psychiatryic_or_mental_illness (77.55% missing), transgender (77.55% missing), white (77.55% missing), created_date, publication_id, parent_id (43.14% missing), article_id, rating, funny, wow, sad, likes, disagree, sexual_explicit, identity_annotator_count, toxicity_annotator_count
2. The size of test.csv is (97320, 2). It has 97320 observations and 2 variables.
- One variable is ID, and the other one is comment_text.
3. The size of sample submission.csv is (97320, 2).
- One variable is ID, and the other one is prediction.
4. Target is the predict variable.

When the target >= 0.5 means it's a toxic comment, the target <0.5 means it's not a toxic comment. There're 1,660,540 observations are not toxic, and 144,334 observations are toxic in the train.csv file.
5. The main variable to use for prediction of Target is Comment_ text, which is text classification.

Introduction

In this competition, the company wants participators to use Identity to decrease bias classification on toxicity. Identity columns: [Male, female, homosexual_gay_or_lesbian, Christian, Jewish, Muslim, black, white, psychiatric_or_mental_illness] But, building models including identity columns needs to use Neuro-linguistic programming (NLP) algorithms. There're kennels shared they used pre-trained word embeddings, and text classification like CNN, LSTM methods, and the popular frameworks are Keras and TensorFlow. Their models include identity columns, and they're very complicated, but they did have high accuracy and reduce bias using different weights. However, I choose to use basic classification algorithms for text like Naive Bayes, Stochastic gradient descent Classifier, Logistic Regression to predict Target based on comment text.

Project goals

Comment text is the only variable be used to build models which is X, and Target is the predicted variable which is Y. The goals are to use different basic algorithms for text classification to predict comment text is toxic or not and to evaluate models by AUC, the average accuracy of 5 cross-validations, and the accuracy score.

Process:

Preprocessing

There are two different approaches to clean the comment text.

1) TfidfVectorizer

- The first one is a simple approach, I use TfidfVectorizer to clean comment text from scikit-learn.

2) NLTK

- The second way to clean the data is more complicated. I import NLTK to clean the comment text. First, I make comment text to lower case, then I set stopwords = English. If comment texts are not English words, they will be removed. I also remove punctuation in the comment texts, and I drop the original column. I added a new column called cleaned_comment, and I set cleaned_comment to X.

```
x=df_train.cleaned_comment
x
```

```
0          cool    s like    would want mother read      ...
1          thank     would make life lot less anxiety ind...
2          urgent design problem   kudos taking   impress...
3              something  ll able install site   releasing
4                                haha guys bunch losers
5                                     ur sh tty comment
6                                hahahahahahahahhha suck
7                                     ffffuuuuuuuuuuuuuu
8          ranchers seem motivated mostly greed   one rig...
9             great show   combo  d expected good together
```

I transform Target (Y) to 0 and 1 because Target was float type, its value is between 0 and 1.

- When the target >= 0.5 means it's a toxic comment and I convert the target to 1, the target <0.5 means it's not a toxic comment and set it to 0. Counter= {0: 1660540, 1: 144334}

Split data (train / test split)

In train.csv file, I split 75 % of data into a training set, and 25 % into a test set. I set random_state=21, which it's useful to compare performance of different models. After splitting data, it returns x_train, x_test, y_train, y_test.

Apply CountVectorizer on cleaned_comment

I apply CountVectorizer on cleaned_comment(X). X can become a vector, then I can use methods to compute Y prediction. In the beginning, I transform X into a vector before I split data into training and test, and the result of accuracy is 88.52%. I saw an example of spam filter on ritchieng.com, and it said that the accurate way is to split train test first, then conduct CountVectorizer and then do fit_transform on x_train.

Methods

Methods used on NLTK processing data

- **Use CountVectorizer on cleaned_comment before train and test split**

1. Naive Bayes -MultinomialNB
   AUC = 0.8559
   accuracy score:0.8852818697794197

- **Use CountVectorizer on x_train after train and test split**

1. SGDClassifier (*loss='hinge'*)- default
   Accuracy score: 0.9405366352037481
   AUC: None

Using SGDClassifier (*loss='hinge'*) doesn't support predict_proba, so I can't compute AUC (Area under the ROC Curve), and then I tried loss="log" which supports predict_proba to calculate AUC.

2. SGDClassifier (loss="log")
   AUC: 0.9236210609607348
   Accuracy score: 0.9376400373211234

3. Naive Bayes -MultinomialNB
   AUC: 0.858458994589174
   Accuracy score: 0.8946210155157474

Summarizing performance on train test split on cleaned comment (countvectorizer) :
AUC: SGDClassifier (loss="log") > Naive Bayes
Accuracy score: SGDClassifier (*loss='hinge'*) > SGDClassifier (loss="log") > Naive Bayes

SGDClassifier (*loss='hinge'*) has highest accuracy score, and SGDClassifier (loss="log") has the best AUC, SGDClassifier (*loss='hinge'*) is linear Support Vector Machine which is highly recommend for text classification because it has high accuracy.

- **Methods used on TfidfVectorizer processing comment_text**
  1. Logistic Regression
     Accuracy score: 0.9472916698986523

     AUC: 0.9484

Using TfidfVectorizer to do preprocessing data cleaning has the best AUC and Accuracy score.

Cross-validation

Cross validation is different from train and test split. First, I transform comment_text by TfidfVectorizer to a vector and cleaned_comment by CountVectorizer to a vector. Next, I use cross_val_score to do cross validation, and I only set cv=5 because there're 1804874 observations, it takes a lot of time to get the output. Finally, I evaluate the result by the average of accuracy.

- Conduct cv on cleaned_comment by CountVectorizer

1. SGDClassifier (*loss='hinge'*)- default
   Cross validation: [0.90725104 0.90763111 0.90935424 0.90687082 0.9081759]
   The average accuracy of 5 cross-validation: 0.9078566229277186
2. SGDClassifier (loss="log")
   Cross validation: [0.92103556 0.92195166 0.92370365 0.92242237 0.92290456]
   Average accuracy of 5 cross validation: 0.9224035606433437
3. Naive Bayes -MultinomialNB
   Cross validation : [0.85397306 0.8502629 0.84690677 0.855935 0.85479617]
   Average accuracy of 5 cross validation: 0.8523747779007984

- Conduct cv on comment_text by processing TfidfVectorizer

4. Logistic Regression
   Cross validation: [0.94852595 0.94707324 0.94872224 0.94681234 0.94668067]
   Average accuracy of 5 cross validation: 0.9475628881034062

The average accuracy of 5 cross-validation: Logistic Regression (TfidfVectorizer) > SGDClassifier (loss="log") > SGDClassifier (*loss='hinge'*) > Naive Bayes -MultinomialNB

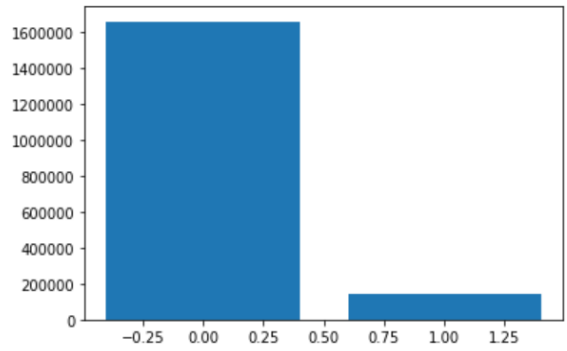
Conclusions

This is a huge dataset, so I realized I spent a lot of time on reading the file through pandas, running the code, and waiting for the output of each method. I didn't code timer to record how much time each method takes. Yet, when I do cross validation, Naive Bayes can get the result very fast, on the other hand, using logistic regression is the slowest. Getting output from Naive Bayes is the fastest, but it has the lowest accuracy. I prefer to use SGDClassifier (*loss='hinge'*) to build the model, because it has the best performance overall, and I didn't wait too much time to get the output. On the data cleaning side, using TfidfVectorizer has better performance than using NLTK, but it doesn't make sense to do vectorizing before splitting train and test. Using NLTK to clean the comment text is very slow, but it's a better approach like analyzing data in real world. In the future, trying NLP on this dataset may get a better model.

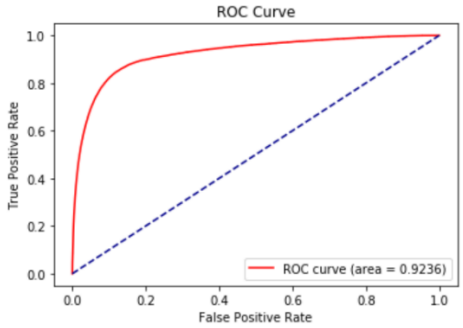
Visualization:

1 is toxic comment, 0 is not toxic comment.

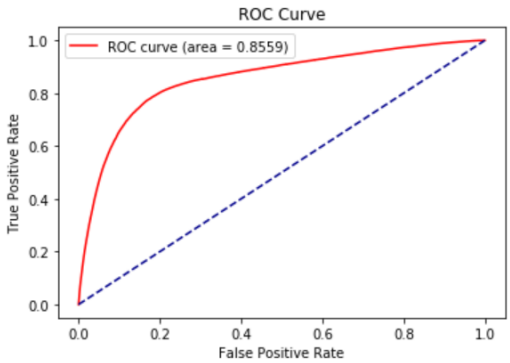
```
Counter({0: 1660540, 1: 144334})
```



SGDClassifier



Naive Bayes



TfidfVectorizer + logistic regression

```
print(x_train)
```

```
  (0, 228600)   0.6505919545142376
  (0, 53302)    0.45629230790627895
  (0, 296941)   0.35760368851034546
  (0, 263448)   0.3565702557349731
  (0, 60212)    0.2931852205084581
  (0, 204891)   0.16597324974586813
  (1, 261514)   0.28023311484303653
  (1, 23508)    0.22284829148050364
  (1, 133716)   0.34587354559324207
  (1, 110238)   0.211186557156831
  (1, 90450)    0.14253765028226184
  (1, 169161)   0.18645019044940705
```

Jupiter notebook:
1.ShuyaC478final-tfidf contains using TfidfVectorizer to clean comment text, train test split, Logistic Regression.

2. ShuyaC478final-nltk contains using NLTK to clean Text data -> count vectorizing cleaned_comment -> train test split -> Naive Bayes.

3. ShuyaC478FinalP2.html contains using NLTK to clean Text data -> train test split -> count vectorizing cleaned_comment -> SGDClassifier -> Naive Bayes -MultinomialNB.