Qi Sun

About

Using Natural Language Processing to find deceptive spams on the review websites

Qi Sun 1 day ago · 6 min read

Introduction

Do you have this kind of experience: You want to have a big meal after several days of frustrating work, so you open Yelp and find a restaurant which has a extremely high rating and lots of positive reviews. However, when you go to that restaurant, you find that it is not satisfactory and you are very disappointed.

This may be because you have fallen into the trap of spams!

Image credit: http://www.pexels.com/

Nowadays, with the development of different kinds of evaluation websites, nearly all consumer products, ranging from commodities to restaurants, can be commented online. Without actually seeing or experiencing them, customers are more likely to choose the consumer products depending on their comments.

This brings convenience, at the same time, however, brings a new problem🙈. To attract more customers, many merchants create fake positive reviews, at the same time, they hire people to write negative reviews under their rival merchants. We call these kinds of reviews 'spam'. This not only deceive the customers, but also destroy the fair competing environment between merchants.

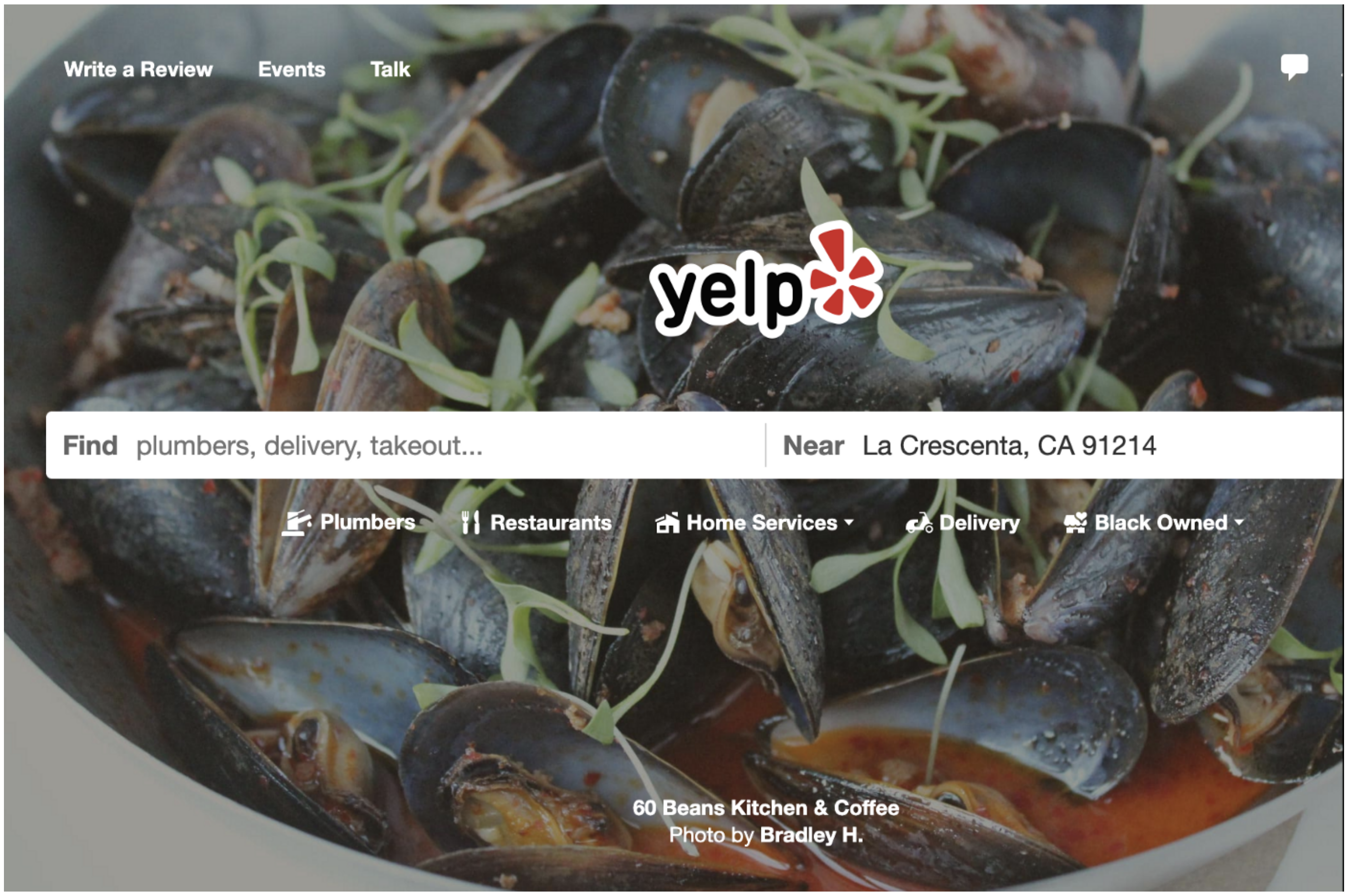


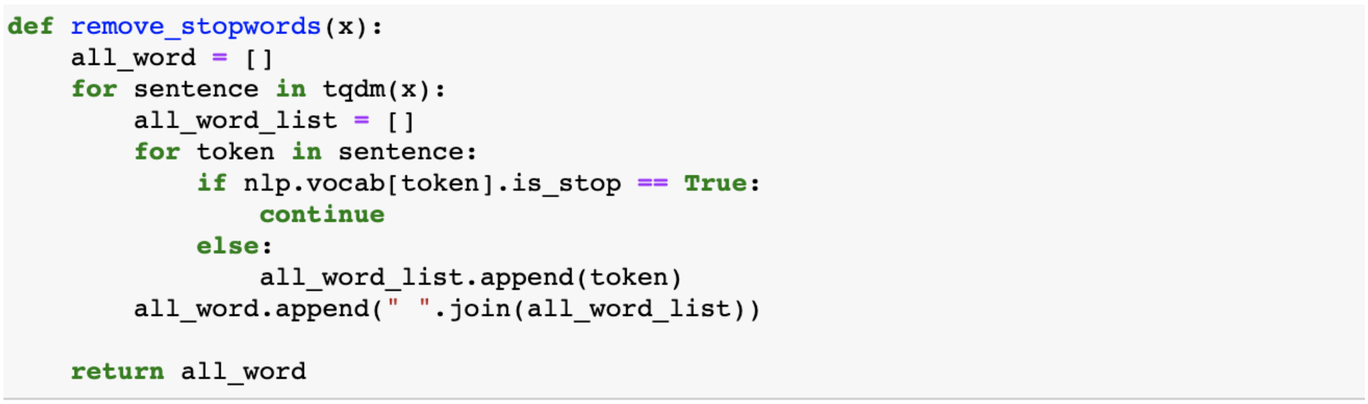
image credit: https://www.yelp.com/

Therefore, I want to use Natural Language Processing to try to detect those deceptive spam reviews. To achieve the target, I will train various machine learning models, which will be introduced in the section 'method'.

Data

I use the dataset which is collected from Yelp. It contains about 600,000 reviews accompanied by the information of reviewers' id, restaurants' id, the date of review and the rating given by the reviewer. The url of the data source is: http://odds.cs.stonybrook.edu/yelpzip-dataset/. In the dataset, the ratio between spams and non-spams is about 1:6.5.

The dataset is overall clean. Therefore, to preprocess it, I only the stop words. The punctuation will be ignored when you use tokenizer.



```python
def remove_stopwords(x):
    all_word = []
    for sentence in tqdm(x):
        all_word_list = []
        for token in sentence:
            if nlp.vocab[token].is_stop == True:
                continue
            else:
                all_word_list.append(token)
        all_word.append(" ".join(all_word_list))

    return all_word
```

remove stopwords

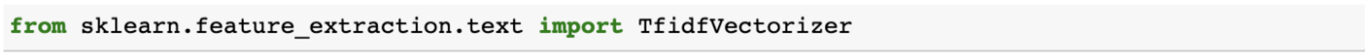
Method

For my methods to classify the spams, I want to divide them into two parts, which are supervised model establishing and neural network models establishing.

supervised model establishing

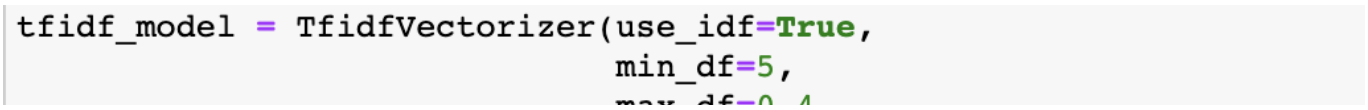
Before establishing supervised model, I firstly extract features.

The features I choose to extract firstly are the word frequency(count) vectors and the TF-IDF vectors. Sklearn has a good vectorizer 'TfidfVectorizer' to combine the previous two vectors.

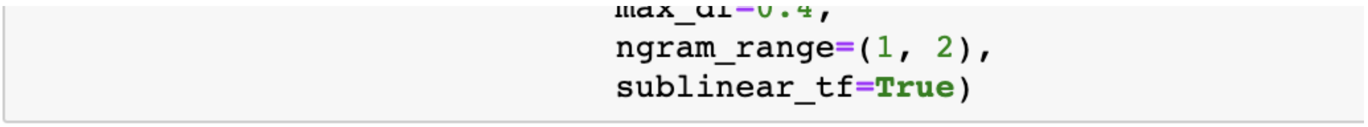


```python
from sklearn.feature_extraction.text import TfidfVectorizer
```

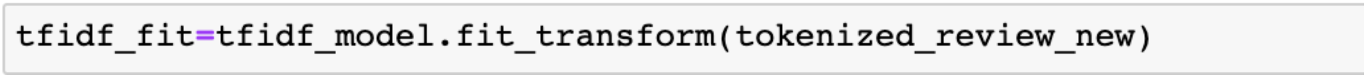
how to import this vectorizer



```python
tfidf_model = TfidfVectorizer(use_idf=True,
                              min_df=5,
```



```
                                max_df=0.4,
                                ngram_range=(1, 2),
                                sublinear_tf=True)
```



```
tfidf_fit=tfidf_model.fit_transform(tokenized_review_new)
```

how to use it

In addition to the above features, I also extract some metadata like the length of reviews, reviewers' id, restaurants' id, the date of the review, the rating given by the reviewer.

a. Naive Bayes Model

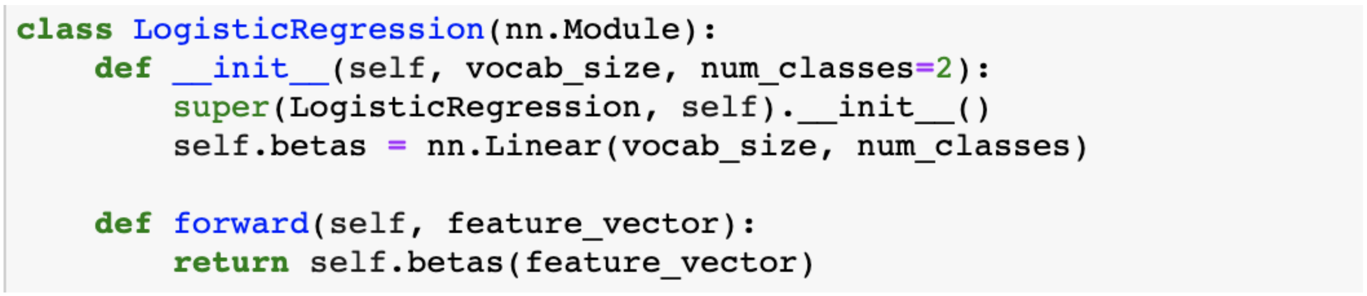
Firstly, I trained the Naive Bayes model. I calculate the probability of each class(spam and non-spam) given each review and choose the class whose probability is bigger. To train this model, I follow the equation:

$$P(class|word) = \frac{P(class)P(word|class)}{\sum_{class} P(class)P(word|class)}$$

the equation of the Naive Bayes

b. Logistic regression

Logistic regression is also frequently used in text classification. Since logistic regression model is just a neural network with one layer. Therefore, the pytorch library can be used to train it. The tutorial of pytorch is on https://pytorch.org/tutorials/. The following code show about how to establish it on a very fundamental basis.



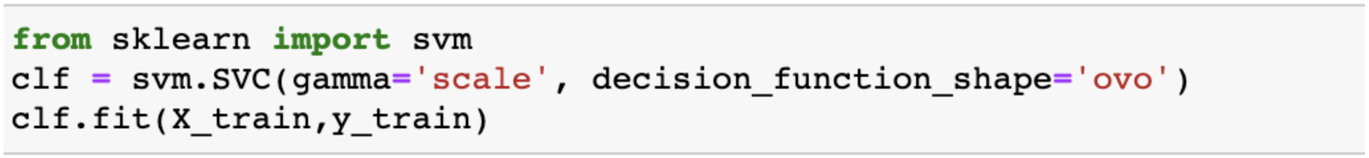
```
class LogisticRegression(nn.Module):
    def __init__(self, vocab_size, num_classes=2):
        super(LogisticRegression, self).__init__()
        self.betas = nn.Linear(vocab_size, num_classes)

    def forward(self, feature_vector):
        return self.betas(feature_vector)
```

establishing logistic regression model using pytorch

c. SVM model

For SVM model, the sklearn library has done a very good job in training it. What needs to be done is just importing the model and fitting it. However, the training speed of this model is not fast,



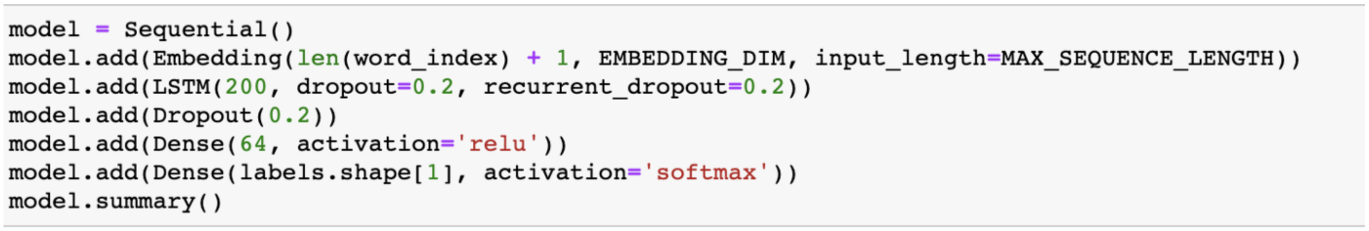
```
from sklearn import svm
clf = svm.SVC(gamma='scale', decision_function_shape='ovo')
clf.fit(X_train,y_train)
```

establishing svm model using sklearn

neural networks establishing

a. LSTM model

Neural networks model is very good at learning document-level representation. And LSTM is the model which can capture the previous information in the sequences. Therefore, I think establishing a LSTM model for classification is feasible. To establish this model, Keras library is used. The tutorial of it is on https://keras.io/guides/ and you can refer to it. The following images can give you some ideas of how to establishing LSTM using Keras. The following codes show fundamentally how to establish the LSTM model using Keras. You need to set your own embedding dimensions and maximum sequence length if you follow them.



```
model = Sequential()
model.add(Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH))
model.add(LSTM(200, dropout=0.2, recurrent_dropout=0.2))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(labels.shape[1], activation='softmax'))
model.summary()
```

Basic codes for establishing LSTM

b. BERT model

A MLM(Masked Language model) can used to do the classification task. To establish and use the BERT classification, I would advise using simpletransformers library. The tutorial is on https://simpletransformers.ai/docs/classification-models/. It is very easy to use.

```
model = ClassificationModel('distilbert',
                            'distilbert-base-uncased'
```

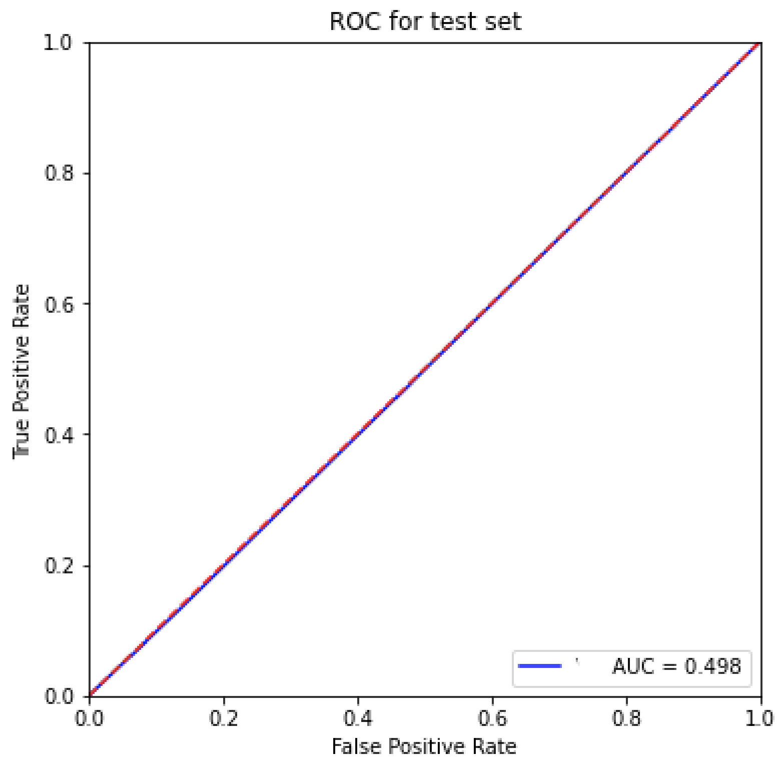
```
                                  'distilbert-base-uncased',
                                  args=model_args,
                                  use_cuda=cuda_available)
```

establishing BERT model using simpletransformers.

Results and Discussions

Because of the imbalance of the dataset, I use ROC curve and AUC to evaluate the classifiers. The reason using ROC is because the shape of it is comparatively stable when the distribution of different classes changes. So it can reduce the interference caused by different distribution of classes, and more objectively measure the performance of the model itself. AUC is the area under the ROC curve.If the ROC bends more to the upper left side and the AUC is bigger, the classifier is better.

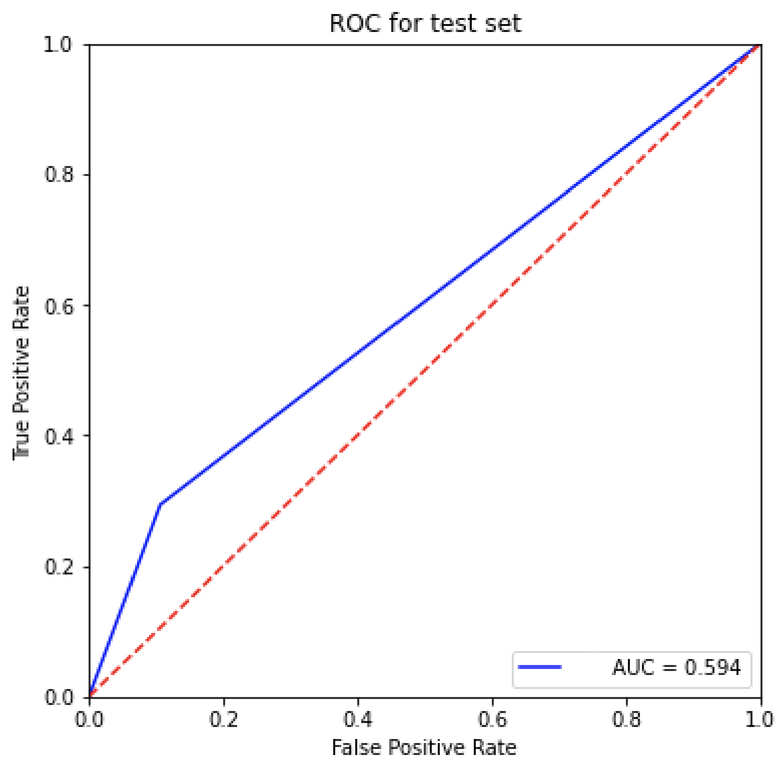
I use a random classifier for my baseline, the ROC curve and AUC of it is:



ROC curve and AUC for the Random classifier

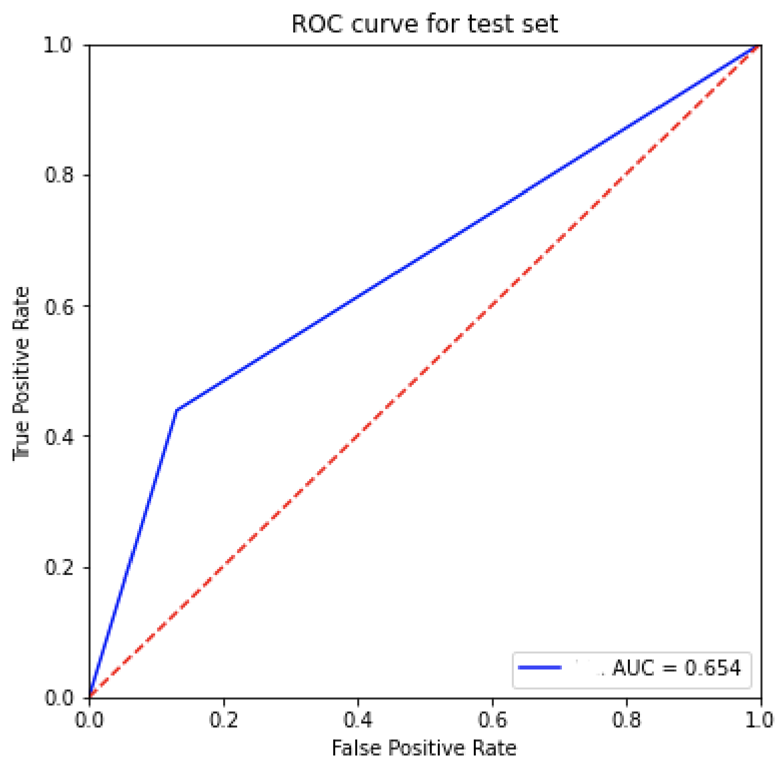
As long as the following models has a better ROC curve and bigger AUC, we can say that the trained classifier is meaningful.

For the Naive Bayes classifier, the ROC curve and AUC of it is:



ROC curve and AUC for the Naive Bayes classifier

For the logistic regression classifier, the ROC curve and AUC of it is:



ROC curve and AUC for the Logistic regression classifier

For the SVM classifier, the ROC curve and AUC of it is:

ROC curve and AUC for the SVM classifier

For the LSTM classifier, since the dimension of the prediction is 2, I do not draw the ROC curve. And its AUC is 0.710.

For the BERT classifier, the ROC curve and its AUC of it is:

ROC curve and AUC for the BERT classifier

In conclusion, for all the models I trained, their AUC are:

Setting the AUC of random classifier as the baseline, we can see from the ROC curves and AUC that all the trained models make sense as the classifier to classify spams. The Naive Bayes classifier is the most fundamental models so it is within my expectation for it to achieve the worst performance. For the best classifiers, SVM and BERT model have the best performance. Since the size(20% of the original size) I used to train SVM is much smaller(it is too slow to train😂!), we can not say that the performance of SVM is better than BERT. Therefore, I put these two models in parallel.

However, the performance is still not perfect since the highest AUC is only 0.851. The main reason may be the fact that the spams written nowadays are more similar to a real one than before. Actually, even for a human being, it is more difficult for him now to detect the spams easily. Therefore, I will discuss next about some of my ideas on improving the performance.

What's next

To improve the performance of the models, I have some ideas but I have not implemented them. If you have time, you can try them😊:

1. For the supervised learning models, you can try to extract more features. For example, I have not taken the reviewers' behavioral features into account. However, they may be useful in prediction because spammers have some distinct properties from regular customers. The features can be the content similarity, the maximum number of reviews post on one day for a single reviewer.

2. For the neural networks models, you can try to increase the number of training epochs if time permits. I believe more training time can bring a better performance!

Medium

About Help Legal

Get the Medium app

Download on the App Store

GET IT ON Google Play