Willy Esquivel-Lopez

(wesquive - G01127937)

(miner2 user-name): Gopher123

HW1 Report

I First imported the data into a Pandas Data Frame to work with it more easily. I Began my process by cleaning the data from any noise by the provided reviews data. The approach I took was:

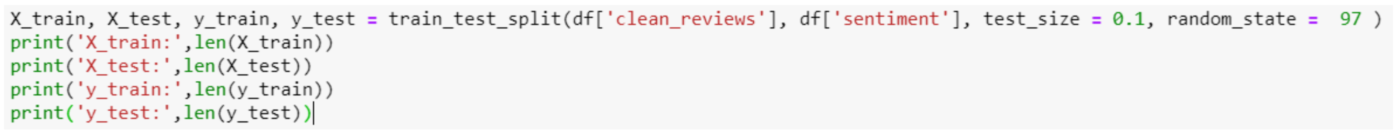
Cleaning data process

1. Remove all of the HTML tags – at a glance I could see several html and markup tags.

2. I also removed all punctuation also using Regex
3. I also began to eliminate stop words (such as “the”, “a”, “an”, “in”) as they are of no use
4. I also proceed to tokenize all the words so that I can use them as a bag of words and later be able to vectorize them
5. Finally, I lemmatized the words to shorten and simplify unnecessary and derived words (such as rocks: rock, corpora: corpus, better: good)

Splitting my training data into sets for training and testing to create my model

With my training data containing exactly 14999 entries of data I split the values as such



```
X_train, X_test, y_train, y_test = train_test_split(df['clean_reviews'], df['sentiment'], test_size = 0.1, random_state =  97 )
print('X_train:',len(X_train))
print('X_test:',len(X_test))
print('y_train:',len(y_train))
print('y_test:',len(y_test))
```

This resulted in a nice split with plenty of training data and enough testing data in a 90% to 10% split to train and test my model, like so:

```
X_train: 13499
X_test: 1500
y_train: 13499
y_test: 1500
```

Model Creation

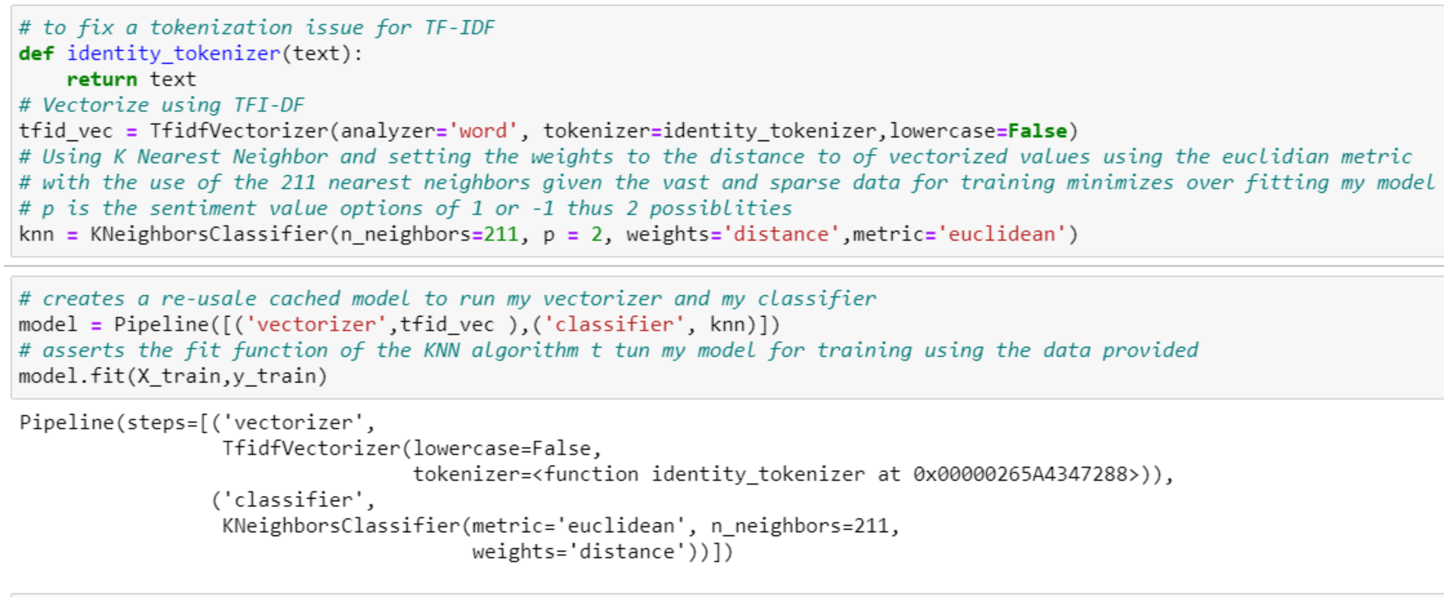
I ran into a problem with he pre tokenized values and utilizing a library which is used to make a matrix of TF-IDF features, thus I included a helper function. (this bug took some time to solve).

After using the "TfidfVectorizer" from SciKit learn, I began implementing my K-Nearest neighbor classifier algorithm using SciKit Learn :

I chose given the sparse data a high odd number of nearest neighbors = "n_neighbors=211" since it appeared that choosing something smaller would be Over Fitting my model selection.

The metric I used for measuring distance was Euclidean "metric='euclidean" paired with a power metric of "p=2" since the only options in sentiment were -1 and 1.

After that I was able to cache my model using a Pipeline which is able to run both the T-FID vectorizer and my classifier. Finally, I train with my model given the parameters "X_train,y_train" or the IV and DV train data.



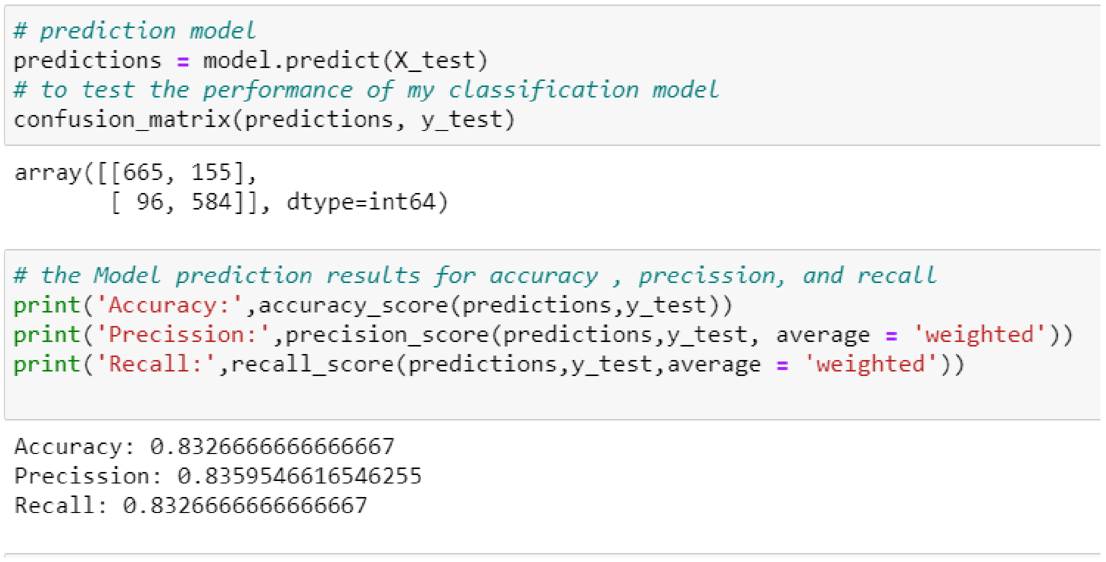
```
# to fix a tokenization issue for TF-IDF
def identity_tokenizer(text):
    return text
# Vectorize using TFI-DF
tfid_vec = TfidfVectorizer(analyzer='word', tokenizer=identity_tokenizer,lowercase=False)
# Using K Nearest Neighbor and setting the weights to the distance to of vectorized values using the euclidian metric
# with the use of the 211 nearest neighbors given the vast and sparse data for training minimizes over fitting my model
# p is the sentiment value options of 1 or -1 thus 2 possiblities
knn = KNeighborsClassifier(n_neighbors=211, p = 2, weights='distance',metric='euclidean')
```

```
# creates a re-usale cached model to run my vectorizer and my classifier
model = Pipeline([('vectorizer',tfid_vec ),('classifier', knn)])
# asserts the fit function of the KNN algorithm t tun my model for training using the data provided
model.fit(X_train,y_train)
```

```
Pipeline(steps=[('vectorizer',
                 TfidfVectorizer(lowercase=False,
                                 tokenizer=<function identity_tokenizer at 0x00000265A4347288>)),
                ('classifier',
                 KNeighborsClassifier(metric='euclidean', n_neighbors=211,
                                      weights='distance'))])
```

Next, I began to evaluate my classifications by doing some predictions on my split test data, I am able to gather the prediction outcome into a “confusion_matrix”

I am then able to gauge the outcomes of my prediction on my test data with accuracy, precision, and recall scores. All these scores are taken into account and result in a positive outcome given my metrics chosen for my model:



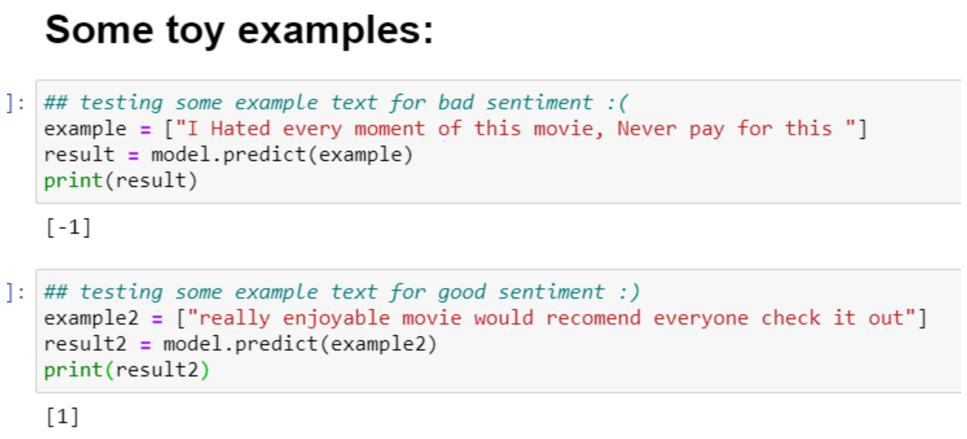
```
# prediction model
predictions = model.predict(X_test)
# to test the performance of my classification model
confusion_matrix(predictions, y_test)
```

```
array([[665, 155],
       [ 96, 584]], dtype=int64)
```

```
# the Model prediction results for accuracy , precission, and recall
print('Accuracy:',accuracy_score(predictions,y_test))
print('Precission:',precision_score(predictions,y_test, average = 'weighted'))
print('Recall:',recall_score(predictions,y_test,average = 'weighted'))
```

```
Accuracy: 0.8326666666666667
Precission: 0.8359546616546255
Recall: 0.8326666666666667
```

I preform a final but simple sanity check:



Some toy examples:

```
]: ## testing some example text for bad sentiment :(
   example = ["I Hated every moment of this movie, Never pay for this "]
   result = model.predict(example)
   print(result)
```

```
[-1]
```

```
]: ## testing some example text for good sentiment :)
   example2 = ["really enjoyable movie would recomend everyone check it out"]
   result2 = model.predict(example2)
   print(result2)
```

```
[1]
```

I preform the same analysis on the Test_data.txt and saved the results in a file “output.txt”

P.S. I could not make a good plot or graph to save my life, thus, to choose a good number of nearest neighbor and test train slip I made some iterative for and while loops and compared results and found this to produce the best outcome