Case Study - Sentiment Analysis

May 10, 2021

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

Note: Some of the cells in this notebook are computationally expensive. To reduce runtime, this notebook is using a subset of the data.

1 Case Study: Sentiment Analysis

1.0.1 Data Prep

In [ ]:
```
import pandas as pd
import numpy as np

# Read in the data
df = pd.read_csv('Amazon_Unlocked_Mobile.csv')

# Sample the data to speed up computation
# Comment out this line to match with lecture
df = df.sample(frac=0.1, random_state=10)

df.head()
```

In [ ]:
```
# Drop missing values
df.dropna(inplace=True)

# Remove any 'neutral' ratings equal to 3
df = df[df['Rating'] != 3]

# Encode 4s and 5s as 1 (rated positively)
# Encode 1s and 2s as 0 (rated poorly)
df['Positively Rated'] = np.where(df['Rating'] > 3, 1, 0)
df.head(10)
```

In [ ]:
```
# Most ratings are positive
df['Positively Rated'].mean()
```

In [ ]:
```
from sklearn.model_selection import train_test_split

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df['Reviews'],
                                                    df['Positively Rated'],
                                                    random_state=0)
```

In [ ]:
```
print('X_train first entry:\n\n', X_train.iloc[0])
print('\n\nX_train shape: ', X_train.shape)
```

2 CountVectorizer

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

# Fit the CountVectorizer to the training data
vect = CountVectorizer().fit(X_train)
```

In [ ]:
```
vect.get_feature_names()[::2000]
```

In [ ]:
```
len(vect.get_feature_names())
```

In [ ]:
```
# transform the documents in the training data to a document-term matrix
X_train_vectorized = vect.transform(X_train)

X_train_vectorized
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

# Train the model
model = LogisticRegression()
model.fit(X_train_vectorized, y_train)
```

In [ ]:
```
from sklearn.metrics import roc_auc_score

# Predict the transformed test documents
predictions = model.predict(vect.transform(X_test))

print('AUC: ', roc_auc_score(y_test, predictions))
```

In [ ]:
```
# get the feature names as numpy array
feature_names = np.array(vect.get_feature_names())

# Sort the coefficients from the model
sorted_coef_index = model.coef_[0].argsort()

# Find the 10 smallest and 10 largest coefficients
# The 10 largest coefficients are being indexed using [:-11:-1]
# so the list returned is in order of largest to smallest
print('Smallest Coefs:\n{}\n'.format(feature_names[sorted_coef_index[:10]]))
print('Largest Coefs: \n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))
```

3 Tfidf

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

    # Fit the TfidfVectorizer to the training data specifiying a minimum document frequency
    vect = TfidfVectorizer(min_df=5).fit(X_train)
    len(vect.get_feature_names())
```

In [ ]:
```
X_train_vectorized = vect.transform(X_train)

    model = LogisticRegression()
    model.fit(X_train_vectorized, y_train)

    predictions = model.predict(vect.transform(X_test))

    print('AUC: ', roc_auc_score(y_test, predictions))
```

In [ ]:
```
feature_names = np.array(vect.get_feature_names())

    sorted_tfidf_index = X_train_vectorized.max(0).toarray()[0].argsort()

    print('Smallest tfidf:\n{}\n'.format(feature_names[sorted_tfidf_index[:10]]))
    print('Largest tfidf: \n{}'.format(feature_names[sorted_tfidf_index[:-11:-1]]))
```

In [ ]:
```
sorted_coef_index = model.coef_[0].argsort()

    print('Smallest Coefs:\n{}\n'.format(feature_names[sorted_coef_index[:10]]))
    print('Largest Coefs: \n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))
```

In [ ]:
```
# These reviews are treated the same by our current model
    print(model.predict(vect.transform(['not an issue, phone is working',
                                        'an issue, phone is not working'])))
```

4 n-grams

In [ ]:
```
# Fit the CountVectorizer to the training data specifiying a minimum
    # document frequency of 5 and extracting 1-grams and 2-grams
    vect = CountVectorizer(min_df=5, ngram_range=(1,2)).fit(X_train)

    X_train_vectorized = vect.transform(X_train)

    len(vect.get_feature_names())
```

In [ ]:
```
model = LogisticRegression()
    model.fit(X_train_vectorized, y_train)

    predictions = model.predict(vect.transform(X_test))

    print('AUC: ', roc_auc_score(y_test, predictions))
```

```
In [ ]: feature_names = np.array(vect.get_feature_names())

        sorted_coef_index = model.coef_[0].argsort()

        print('Smallest Coefs:\n{}\n'.format(feature_names[sorted_coef_index[:10]]))
        print('Largest Coefs: \n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))
```

```
In [ ]: # These reviews are now correctly identified
        print(model.predict(vect.transform(['not an issue, phone is working',
                                            'an issue, phone is not working'])))
```