tweet_location

September 14, 2021

1 Classifying Tweets: Location

In this project, we will be using a Naive Bayes Classifier to patterns in tweets. Using tweets separated by location from New York `new_york.json`, London `london.json`, and Paris `paris.json`. The goal is to be able to classify a tweet as belonging to one of these cities.

1.0.1 Imports

Our imports are pretty simple, we only need pandas and scikitlearn for this functionality.

[14]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

1.1 Investigating the Data

First we'll look at `new_york.json` and print out the: - The number of tweets - The columns (features) of the tweet - The text of the 12th tweet in the set

[15]:
```
new_york_tweets = pd.read_json("new_york.json", lines=True)
print(len(new_york_tweets))
print(new_york_tweets.columns)
print(new_york_tweets.loc[12]["text"])
```

```
4723
Index(['created_at', 'id', 'id_str', 'text', 'display_text_range', 'source',
       'truncated', 'in_reply_to_status_id', 'in_reply_to_status_id_str',
       'in_reply_to_user_id', 'in_reply_to_user_id_str',
       'in_reply_to_screen_name', 'user', 'geo', 'coordinates', 'place',
       'contributors', 'is_quote_status', 'quote_count', 'reply_count',
       'retweet_count', 'favorite_count', 'entities', 'favorited', 'retweeted',
       'filter_level', 'lang', 'timestamp_ms', 'extended_tweet',
       'possibly_sensitive', 'quoted_status_id', 'quoted_status_id_str',
       'quoted_status', 'quoted_status_permalink', 'extended_entities',
       'withheld_in_countries'],
```

```
      dtype='object')
Be best #ThursdayThoughts
```

Now lets load in the tweets from London and Paris and see how many tweets each has.

[16]:
```
london_tweets = pd.read_json('london.json', lines=True)
paris_tweets = pd.read_json('paris.json', lines=True)
print(f'London tweets: {len(london_tweets)}')
print(f'Paris tweets: {len(paris_tweets)}')
```

```
London tweets: 5341
Paris tweets: 2510
```

1.2 Naive Bayes Classifier: Using Language to Segment Tweets

Let start by creating lists of all the tweet's text, and assigning them labels by location. A New York tweet will have a label of `0`, London will be `1`, and Paris will be `2`.

[17]:
```
new_york_text = new_york_tweets['text'].tolist()
london_text = london_tweets['text'].tolist()
paris_text = paris_tweets['text'].tolist()
all_tweets = new_york_text + london_text + paris_text
labels = [0] * len(new_york_text) + [1] * len(london_text) + [2] *␣
 ↪len(paris_text)
```

Now lets break up our data into training and test sets. We'll change `the test_size` to be 20% (from the default 25%) and set a `random_state` to keep values consistent across runs.

[18]:
```
train_data, test_data, train_labels, test_labels = train_test_split(all_tweets,␣
 ↪labels, random_state = 1, test_size = 0.2)
```

Now lets transform our word lists into count vectors with `CountVectorizer`.

[19]:
```
counter = CountVectorizer()
# teaching our counter the vocab
counter.fit(train_data)
train_counts = counter.transform(train_data)
test_counts = counter.transform(test_data)
```

Lets look at index `[3]` to see what the data looks like.

[20]:
```
print(train_data[3])
print(train_counts[3])
```

```
saying bye is hard. Especially when youre saying bye to comfort.
  (0, 5022)     2
  (0, 6371)     1
  (0, 9552)     1
  (0, 12314)    1
  (0, 13903)    1
  (0, 23994)    2
```

```
  (0, 27146)    1
  (0, 29397)    1
  (0, 30274)    1
```

1.3 Training and Testing our Classifier

Now that we have the inputs, lets use the CountVectors to make our classifier. After we train it, lets check what our predictions look like.

[21]:
```
classifier = MultinomialNB()
classifier.fit(train_counts, train_labels)
predictions = classifier.predict(test_counts)
print(predictions)
```

```
[0 2 1 … 1 0 1]
```

1.4 Evaluating the Model

Now that we've made our predictions, lets check the accuracy. We'll do it by checking with skikitlearn's `accuracy_score` and a `confusion_matrix`.

[22]:
```
# testing the predictions
print(accuracy_score(test_labels, predictions))

#          NY : London : Paris
# True : 541 : 404 : 28
# True : 203 : 824 : 34
# True :  38 : 103 : 340
print(confusion_matrix(test_labels, predictions))
```

```
0.6779324055666004
[[541 404  28]
 [203 824  34]
 [ 38 103 340]]
```

Now we'll test with our own tweet!

[27]:
```
tweet = "Earl Grey in the afternoon is one of life's greatest pleasures."
tweet_counts = counter.transform([tweet])
print(classifier.predict(tweet_counts))
```

```
[1]
```

Since it classified the tweet as `1`, we correctly predicted it as a London tweet!

Data Sources Data was provided by twitter.