Challenge 3

Bekzod Normatov

November 2021

1 Describing my solution

In order to come up with a solution for this challenge I had to refer back to the Week 10 Notebook on text mining, where we started processing and preparing text data for machine learning. We have also already written a function called load_reviews that given a directory name, can retrieve information about all movie reviews, which includes filename, text, which is a content of the file, and kind, which is either positive or negative for a given movie review. This function returns a data frame. After loading movie reviews, I want to prepare my X, training and testing vectors, and y, target values, for model training and testing. y is my target values, which is basically the kind of a movie review, which is what we want to predict with our model. In order to make this work, I want to turn the kind variable into a numeric. In this case, I assigned 0 to all negative reviews and 1 to all positive reviews. In order to define training and testing vectors, I instantiate CountVectorizer with stop_words = 'english', which will remove all the stopwords from the resulting tokens. A particular method of this vectorizer, .fit_transform prepares my training and testing vectors by turning raw documents into a document-term matrix. Finally, I fit my chosen model, which is a multinomial Naive Bayes classifier by training it on a subset of the vectorized documents.

My predict_sentiment function, which takes in a review of type string, first transforms my review into a document term matrix and then uses my MultinomialNB model to predict a sentiment of a given review. If the prediction is equal to 0, then the model predicts the review to be negative, and if the prediction is equal to 1, then the model predicts the review to be positive.

2 Another explored solution

I have also tried a different approach, which turned out to be computationally much more expensive and not as accurate as my final solution. My initial idea was to iterate through all the reviews, tokenizing, stemming, removing stopwords and vectorizing each review one by one. I also used TfidfTransformer in order to create a tf-idf matrix. A sample of the initial solution can be seen below.

```
for index, row in dataset.iterrows():
    text = row['text']
    file = row['filename']
    stemmed_s = [snowball_stemmer.stem(x.lower()) for x in
    nltk.word_tokenize(text) if not x.lower() in stopWords]
    word_count_vector=cv.fit_transform(stemmed_s)
    tfidf_transformer.fit(word_count_vector)
    df = pd.DataFrame(tfidf_transformer.idf_,
    index=cv.get_feature_names(),columns=[file]).to_dict()
    dfs[file] = df[file]
    if row['kind'] == 'neg':
        kind.append(0)
    else:
        kind.append(1)
X = pd.DataFrame(dfs).T.fillna(0)
```

However, what made this particularly slow is that vectorized each document separately as well as transformed document-term matrix into a tf-idf matrix representation. However, the performance accuracy of this model was much lower, compared to my final solution. Therefore, I decided that it is better to apply vectorization as well as tf-idf transformation on a whole corpus of documents, which does make sense as the weights assigned to each term are not limited by its frequency in a single document. Also, in ym final solution I decided not to initialize tfidf transformer because based on accuracy tests, it did not improve the accuracy of my predictions, but it did take much longer compared to a solution without tfidf transformer.

3 Performance analysis of my solution

3.1 Test 1 - import and pred. duration

Both Import and prediction times are quite good compared to my initial solution described above, where importing took around 40 seconds. As expected, import time is longer than prediction time:

```
Import: 3.814271926879883
Pred.:  0.0019638538360595703
```

3.2 Test 2 - first check

This test basically tests whether the function predict_sentiment performs as expected. That is given a string input, it returns 'neg' or 'pos'. We test the function on one positive and one negative review:

```
Result for a negative: neg
Result for a positive: pos
```

It seems like the function does output what we would expect. Although this test is quite simple, it is still a good indicator of the basic functionality of my solution.

3.3 Test 3 - Prediction performance: execution time

This test measures the execution time of performing predictions on the first 1000 entries of the movie reviews dataset. Given that the assignment requires that my solution takes no more than an hour to train and predict 1000 movie reviews I think the test shows a very good result:

```
1.07 s ± 27.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

3.4 Test 4 - Prediction performance: accuracy

This test is all about the accuracy of my predictions. It calculates the proportion of correctly predicted predictions by checking whether the prediction is equal to the actual kind for each movie review. I am quite impressed with the accuracy of my model, which is above 90% accuracy rate:

```
0.938
```

4 Conclusion

Overall, this was a fruitful and challenging task. The biggest challenge for me personally was to make my solution more efficient and faster. Surprisingly it was quite easy to achieve high accuracy with bad code and by 'bad' here I mean very expensive and time-consuming solution. The main advantage of my final solution is precisely the fact that I do not have to iterate through every movie review to achieve just as accurate model. I have also tried other models such as k-Nearest Neighbours. However, they turned out to be not as accurate as my Multinomial Naive Bayes model.