Assessed Coursework 3

CID 01252821

Problem 1

Part 1

Applying count vectorization to the sentence "see eye to eye" first involves splitting the sentence into separate words (tokens), specifically "see", "eye", "to", "eye", then the frequency for every unique word is counted, and finally the counts are represented in a vector: (count(eye), count(see), count(to)) or $(2, 1, 1)$.

Part 2

The count vectorization of the query "eye" assuming the vocabulary from `Part 1` (eye, see, to) is $(1, 0, 0)$. The Euclidean distance between the query and the above sentence is the Euclidean distance between the two vectors we obtained via the count vectorization: $(2, 1, 1)$ and $(1, 0, 0)$. For the purpose, I have used `Python` and the function `euclidean` from `scipy.spatial.distance`, resulting in an Euclidean distance between the count vectorization of the sentence "see eye to eye" and the query "eye" of approximately 1.73.

Part 3

To find the cosine similarity between the two documents first we are going to apply count vectorization using `CountVectorizer` from `sklearn.feature_extraction.text`, after that we are going to compute the cosine similarity using `cosine_similarity` from `sklearn.metrics.pairwise`. For this problem I will include all of the words in the two documents, hence a slight modification in `CountVectorizer` is needed. By default this function selects tokens of 2 or more alphanumeric characters, so we need to set `token_pattern=r"(?u)\b\w+\b"` in order to consider one letter words. The count vectorization obtained is

	a	at	common	denmark	have	in	is	lunch	midday	sandwich	the	uk	workers
D1	1	0	1	0	0	1	1	1	0	1	1	1	0
D2	0	1	0	1	1	1	0	1	1	0	0	0	1

and the cosine similarity between the two documents results in approximately 0.267 indicating a low degree of similarity between them.

Part 4

To compute the TF-IDF matrix for the two documents I am going to use `TfidfVectorizer` from `sklearn.feature_extraction.text`. As in the previous part we need to set `token_pattern=r"(?u)\b\w+\b"` in order to consider one letter words. The TF-IDF matrix for the two documents is:

	a	at	common	denmark	have	in	is	lunch	midday	sandwich	the	uk	workers
D1	0.378	0.000	0.378	0.000	0.000	0.269	0.378	0.269	0.000	0.378	0.378	0.378	0.000
D2	0.000	0.408	0.000	0.408	0.408	0.290	0.000	0.290	0.408	0.000	0.000	0.000	0.408

Each row represents a document (D1 and D2), and each column represents a unique word from both documents. The values in the matrix are the TF-IDF scores for each word in each document, indicating the importance of each word in the context of the specific document and the entire corpus. Higher values indicate greater importance.

Problem 2

We begin with loading the data and getting the first 1000 not NA descriptions. After that we are pre-processing the descriptions by converting the texts to lowercase, removing numbers and symbols, removing stop words, and stemming. This would help us clean the "noise" from the text, and keep mainly meaningful stemmed words. After that TF-IDF is performed by using `TfidfVectorizer` from `sklearn.feature_extraction.text` with `sublinear_tf=True` which applies sublinear tf scaling, i.e. replaces tf with $1 + \log(tf)$, and after that `TruncatedSVD` from `sklearn.decomposition` is applied with top 100 singular values to be kept, which is the recommended value by `sklearn` when applying LSA. Fitting LSA to our `descriptions` data consisting of 1000 descriptions produces a 1000×100 matrix.

We can observe that the explained variance ratio for the set of components is approximately 0.33 indicating that these components together capture about one-third of the total variance in the dataset. We can choose a bigger number of components leading to a higher explained variance ratio, but for the purpose of this problem I will demonstrate the dimensionality reduction with significantly lower number than the original dimension.

Now we can check what would happen if we take two descriptions not part of the 1000 initial descriptions: `my neighbor presented me with a 2 foot zucchini and this recipe.  i am glad that she did. it is so easy to do and i loved the end result. you can serve this cold as well but i prefer it hot.` and `good healthy low fat soup.`. First, we are transforming the test descriptions to the reduced dimensionality space obtained after applying LSA, this is done with the code `lsa.transform(test_descr)`. After that, the cosine similarity between the test description and the train descriptions can be calculated in the reduced dimensionality space, obtaining results of approximately 0.61 and 0.80. Now, we choose the biggest cosine similarity value among the results, and then we can see which description from the first 1000 descriptions corresponds to it and subjectively compare it with the test description and the cosine similarity value.

For the first test description with cosine similarity 0.61 the corresponding description is `this is a great meal eaten the same day ,but even better the next day , if you can wait! add your favourite spices, but try it first as it is and i think that you will enjoy the 'vegetable' taste. good for freezing.`: both of the sentences indeed have a positive semantic but nothing more specific which confirms the moderate similarity suggested from the cosine similarity.

For the second test description with cosine similarity 0.81 the corresponding description is `a super, veggie-packed salad.  courtesy of rachael ray.`: the test sentence mentions "good healthy low fat" and the suggested sentence contains "veggie-packed salad" which implies the same healthy, low fat characteristics, and hence confirms the suggested higher similarity from the cosine similarity.

Problem 3

Part 1

We begin with loading the data and removing the reviews which do not have a rating or the review states "No Review Text", we are left with 703 reviews.

After that we are pre-processing the reviews by converting the texts to lowercase, removing numbers and symbols, removing stop words, and stemming. This would help us clean the "noise" from the text, and keep mainly meaningful stemmed words. Then, we are creating a column `sentiments` such that the ratings 4 and 5 give it a positive sentiment and a value of 1, and the rest as negative sentiment and a value of 0.

TF-IDF is performed on the reviews by using `TfidfVectorizer` from `sklearn.feature_extraction.text` with `max_features=100`.

Now we can continue with the model which would classify our reviews to be either positive or negative via modelling the `sentiments` column: a Logistic Regression. We are going to use only two columns from the data: the processed reviews (as features after applying TF-IDF to the processed reviews) and the created sentiments columns (as a target column).

First, we have to split the dataset into training and testing sets with the function `train_test_split(features, sentiments, test_size=0.3, random_state=42)` from `sklearn.model_selection`, where `features` is the result obtained from applying TF-IDF to the reviews, and `sentiments` is the column we created in the above steps to model. Now, the logistic regression can be fitted to the train data, and after that a prediction on the test data can be done, allowing to compute an accuracy metric. The accuracy achieved is approximately 0.83 indicating that the model is correctly predicting the outcome 83% of the time, and this can be considered a good level of accuracy.

Part 2

If we apply the trained model from `Part 1` to "My experience at Starbucks is great" and "The Americano tastes awful" we obtain the corresponding results: "positive" and "negative", which aligns with the sentences: "great" implying a positive sentiment, and "awful" implying a negative sentiment.