CP 423 Text Retrieval & Search Engine – Assignment 3

Herteg Kohar (190605160), Kelvin Kellner (190668940)

What each member contributed:

- Herteg Kohar (190605160) - Q1
- Kelvin Kellner (190668940) - Q2

Q1:

How we understand the question:

- We read the documentation from sklearn. We investigated cross-validation and all the model configurations to understand how to instantiate each model and fit the training data. A module called joblib was used to store the model and the vectorizer and the documentation was read in order to understand how to utilize the module.

How we implemented the question:

- Part 1
    - Once the datasets to be included were chosen the datasets are then loaded using pandas and the chosen datasets are then concatenated together in order to create our dataset. Now we split the dataset into two parts 80% for training and 20% for testing. This is done with nltk to tokenize and then remove stopwords. Once this is done, we use the TfidfVectorizer to vectorize the text we fit it with the training set. The vectorizer is then saved to a file to use later for predicting new text. We get vectorized data for both the train and test set. After that, we use our training set for cross-validation to fit our models. We use a 5-fold cross-validation and report the mean accuracy, precision, recall and f1-score for each model's cross validation. The specified model from the user is the one that goes through cross-validation and the best estimator from the folds is used to test the model. The 20% test set is then used and predicted one we then return the metrics for the test set and the confusion matrix. The confusion matrix is then displayed from matplotlib and shown in text in the console as well. The model is saved to a file as well for future predictions. A JSON (JavaScript Object Notation) settings file is then saved which has the included datasets and chosen model.
- Part 2
    - In the next step, we set up the steps in order to predict new text. The program first checks to see if the vectorizer and the model files are present before continuing. The sentence that is entered for prediction is then tokenized and the stopwords are removed. The vectorizer from the training is then loaded from the file as well as the model. The text is then vectorized and the prediction can be made. Once the prediction is made it is shown in the console.

Any challenges/issues that we had:

- Trying to figure out a good we to split the data in order to perform the cross validation while keeping enough data to make test predictions on was a challenge. We were able to overcome this by emailing the Professor and asking for clarification and guidance and we were able to come up with a good training and testing methodology.

Q2:

How we understand the question:

- We read the documentation from sklearn. We used examples from the internet to learn how to tweak hyperparameters. A module called pickle was used to store the processed article data frames to save computation on consecutive program calls. A module called joblib was used to store the model and the vectorizer and the documentation was read in order to understand how to utilize the module.

How we implement the question:

- The dataset was loaded into the program. All articles have a header separated by an empty line. For each article, the header was removed, it was tokenized, and deleted for stop words. Next, a TF-IDF vector was created for the entire corpus. Then, the program calls the appropriate function for the desired clustering model, which will create the user's preferred number of clusters and make a cluster prediction for each article in the dataset. The performance scores of the model are calculated and displayed in the console for the user to see.

Any challenges/issues that we had:

- Processing the data was a bit challenging, the header format is not consistent between documents, and the method we had chosen might not remove 100% of the header every time.
- The data set is quite large, the WHC and AC models take a long time to load. We tried tweaking the hyperparameters or using PCA, but performance did not improve noticeably, so we chose to revert our changes.
- The DBSCAN model is the only model that determines the number of clusters automatically, and for whatever reason, the model always decides to group all the data into a single cluster. The reason for this is not clear to us, once again, we tinkered with hyperparameters but were not able to find a solution and are not sure of the implications behind this result.
- TF-IDF has the option to use max_features, min_df, and max_df parameters which we used to increase performance speed, we used a grid search to find the optimal values for the best score.