Final Project Report - CP423

Authors

Robert Mazza | 190778040

Josh Degazio | 190560510

Adrian Alting-Mees | 190743560

Contribution

Due to the nature of the final assignment there aren’t distinct questions each member worked on, however we can break the contribution down in terms of options (1 - 7)

Option 1: Adrian, Josh

Option 2: Adrian, Josh

Option 3: Adrian, Robert

Option 4: Adrian, Robert

Option 5: Adrian, Robert, Josh

Option 6: Josh, Adrian, Robert

Option 7: Adrian

Installation Instructions:

Make sure you have the following packages installed:

```
pip install requests
pip install hashlib
pip install beautifulsoup4
pip install sklearn
pip install joblib
pip install soundex
```

Sources.txt

If you would like to change the root links for the crawler, make sure to preserve the format of:

`Topic, Link` Look at the given sources.txt for reference.

Option 1: Collect New Documents

Usage Instructions:

Run the program and after being prompted, enter ‘1’. Now wait as the crawler goes to work going through all the links in the sources.txt file. Depending on the amount of links in the sources.txt and the depth of the crawl the execution time will be drastically effected. Less links and lower depth will be fastest. You can watch the crawling process through the crawl.log file.

Important Functions:

collect_documents(url, topic, depth)

This code defines a function called collect_documents that collects new documents from a given URL for a specified topic and depth. The function retrieves the webpage's content using the requests library before parsing the HTML using BeautifulSoup. The text is then extracted from the content, any extra white space is removed, and the text is saved in a file with the content's SHA256 hash as the filename. The function also adds the following information to the crawl.log file: topic, link's URL, URL hash value, and date. The function is then called repeatedly on any links found on the crawled page that match the original URL. The maximum depth of recursion is constrained by the MAX_DEPTH variable.

Option 2: Index Documents

Usage Instructions:

After crawling a decent amount of documents you may enter ‘2’ at the main menu to begin the process of indexing all the stored documents.

Important Functions:

index_documents()

This code creates an inverted index of the downloaded documents and saves it as invertedindex.txt through the function index_documents(). A data structure known as an inverted index stores a mapping between words in documents and the documents in which they appear. A dictionary mapping and an empty dictionary inverted_index are first initialised by the function to store the inverted index and the mapping between document hash and document ID, respectively. The number of documents that have been indexed is tracked using the variable N.

The function then iterates through each topic folder in the data directory, reading each document's contents as it goes and computing the hash value of the document's information. The mapping dictionary is then updated with a new record that contains the hash value and the document's unique document ID H{N}.

The function then extracts the words from the document using regular expression and updates the inverted index by including each word's appearance in the document. The document hash and the number of times the word appears in the document are used to represent the word's appearance as a tuple.

The document hash and the number of times the word appears in the document are combined to form a tuple to represent a word's appearance.

Option 3: Search For A Query

Usage Instructions:

After crawling and indexing, enter ‘3’ on the main menu then you will be prompted to enter a search query.

Important Functions:

search_query()

The function search_query() starts by initializing two dictionaries words and soundex_words. Each term in the corpus is mapped to a list of tuples by the inverted_index, a defaultdict object, where each tuple contains the term frequency in that document and the document ID. The function then creates the inverted_index dictionary after reading the invertedindex.txt file, which contains the inverted index.

The mapping.txt file, which contains a mapping between document IDs and their corresponding topics, is then read by the function. It reads the user's query input, breaks it up into words, and looks for documents that contain at least one of the search terms. The term frequency of each term in the query is then calculated for each document that was retrieved.

The function uses the TfidfVectorizer to vectorize the query and the documents after retrieving the pertinent ones. It determines the cosine similarity between the vectors representing the query and each document, then ranks the documents according to this score. Finally, based on their similarity score, it prints the top 3 relevant documents.

Option 4: Train ML classifier

Usage Instructions:

After searching for a query, enter ‘4’ on the main menu to begin training a classifier using the collected information. This will be saved as classifier.model.

Important Functions:

train_classifier()

This function is designed to train a classifier using a dataset of documents. The documents are stored in subdirectories within a parent directory called "data". The function uses the TfidfVectorizer to convert the raw text content of the documents into a numerical vector representation, which can then be used as input data to train the classifier. The classifier used in this function is a Support Vector Machine (SVM) with a probability parameter set to True.

Once the documents have been vectorized and split into training and testing sets, the SVM classifier is trained on the training data. The trained classifier is then saved as a file called "classifier.model", and the TfidfVectorizer object is saved as "tfidf_vectorizer.joblib".

Finally, the function evaluates the performance of the trained classifier by making predictions on the testing data and printing the accuracy score and confusion matrix. It then prints a message indicating that the classifier has been trained, and calls the "main()" function to continue execution of the program. Overall, this function automates

the process of training a classifier on a given dataset of documents, and saves the trained model for future use.

Option 5: Predict A Link

Usage Instructions:

After training the classifier, enter ‘5’ on the menu to predict the topic of a given link.

Important Functions:

predict_link()

The function "predict_link()" is designed to use a trained classifier to predict the topic of a given link. It first prompts the user to enter a link to predict, and then uses the requests library to retrieve the content of the web page associated with the link. The content is then pre-processed to remove extra whitespace and non-textual elements.

The function then loads the saved TfidfVectorizer object, which was trained on the same dataset as the classifier. The content of the web page is vectorized using the loaded TfidfVectorizer object.

Next, the saved classifier model is loaded using joblib. The trained classifier model then makes a prediction on the vectorized link to determine its most probable topic, and also returns the confidence score for that prediction. The topic and confidence score are printed to the console.

Finally, the function prints a message indicating that the prediction has been made, and then calls the "main()" function to continue the execution of the program. Overall, this function uses a trained classifier to predict the topic of a given web page and returns the confidence score associated with that prediction.

Option 6: Your Story!

Usage Instructions:

Enter ‘6’ in the main menu.

Important Functions:

your_story()

It just prints the text file.

Option 7: Exit

Usage Instructions:

Enter ‘7’ in the main menu.

Important Functions:

sys.exit()

exits the program.