CP423 Final Project

Search engine program implemented in Python.

Overview: This program is a basic search engine implemented in Python. It includes different fundamental components we talked in class for building Indexing and Query Processing pipelines. The program starts from the command line using `python search_engine.py`. The user is prompted with the following message, and they can type the appropriate number in the console and press enter to select the option they want. The program will then prompt the user for the appropriate input and will perform the selected task. The program will continue to prompt the user for input until they select the option to exit the program.

```
Select an option:
1- Collect new documents.
2- Index documents.
3- Search for a query.
4- Train ML classifier.
5- Predict a link.
6- Your story!
7- Exit
```

The program collects documents from websites under all of the three program topics of Astronomy, Economy, and Health. The list of source websites can be modified in `source.txt`.

Program Instructions

Install Dependencies

```
pip install -r requirements.txt
```

Run Program

```
python search_engine.py
```

Running the program displays a brief welcome message. After which, the user enters the option menu event loop, as shown below.

▼ Option menu console output

```
Select an option:
1- Collect new documents.
2- Index documents.
3- Search for a query.
4- Train ML classifier.
5- Predict a link.
6- Your story!
7- Exit
```

1- Collect new documents

The program collects new documents for each of our topics in our search engine. We use the `source.txt` file to begin crawling the links for new pages on the website as our trusted source. This is done for the `TOPIC_DOCUMENT_LIMIT` constant in `constants.py`. The collected documents are then placed in the `data` directory in the appropriate subdirectory according to the topic. When a document is collected, we use its URL to generate a hash, which is then used as the filename for the text file containing the document. The crawler works by iterating over all the sources to collect documents from each of the trusted sources, with the aim of achieving a nicely distributed collection of documents.

▼ Output

```
Enter option: 1
Collecting new documents...
Crawling Astronomy...
Finished crawling Astronomy
Crawling Economy...
Finished crawling Economy
Crawling Health...
Finished crawling Health
```

After the task is complete, the program re-enters the option menu event loop.

2- Index documents

The program iterates over the `data` directory and all the topic subdirectories, and after tokenizing, indexes all of the tokens. Along with each token, a list of occurrences is stored, which includes the document hash, the frequency of the token in the document, and the topic of the document. Since the actual hashes are very long a short form is used for each hash which can be found in `mapping.json` located in the `Index Data` directory. The program then saves the inverted index (`inverted_index.json`) and the mapping to the `Index Data` directory. After this, the program uses `crawler.log` to create a mapping of the hashes to the corresponding URLs (to be saved in `hash_to_url.json`), which are displayed when querying. The outline of the occurrences field in the inverted index includes all occurrences of the token within documents, as well as the frequency of the token in the document. The soundex code for each token is also stored in the inverted index, which can be used for spell correction during the querying process.

▼ Output

```
Enter option: 2
Indexing documents...
Loading existing inverted index...
Inverted index saved to Index Data\inverted_index.json
Mapping saved to Index Data\mapping.json
Hash to url mapping saved to Index Data\hash_to_url.json
```

After the task is complete, the program re-enters the option menu event loop.

3- Search for a query

The program prompts the user to enter a search query. The users query will be spell corrected using soundex similarity, edit distance and term frequency are used for fallback conditions. The program will then perform the term-at-a-time algorithm and use the inverted index to display the top 3 highest ranked documents (ranked using cosine similarity) that contain any of the spell corrected query terms. A link to the web page is given for each document, as well as a snippet of matching text blocks with query terms highlighted in unique colours.

▼ Example output (screenshot)

After the task is complete, the program re-enters the option menu event loop.

4- Train ML classifier

We have chosen the KNN model for training as we found it to be the most effective in predicting and classifying new links into their respective topics. We based this decision on our evaluation metrics, including cross-validation and classification metrics. The program begins by collecting all the documents and placing them into a Pandas dataframe, which includes their text content. The text content is then preprocessed through tokenization and removal of stopwords. Afterward, the data is split into training and testing sets, with 80% used for training and 20% for testing. Next, the TFIDF vectorizer is fitted to the training data and used to vectorize both the training and test sets. The vectorizer is saved for future use in predictions. The KNN model is trained using a grid search to identify the best parameters, in this case the neighborhood size. Once this is done, the model is tested on the test set to make predictions and calculate the relevant metrics.

▼ Example output

```
Enter option: 4
Training ML Classifier...
Training KNN Model
Best parameters: {'n_neighbors': 7}
Best Score: 0.9884472049689441
Test Metrics
            precision    recall  f1-score   support

         0       0.97      1.00      0.98        31
         1       1.00      0.92      0.96        25
         2       0.97      1.00      0.98        31

  accuracy                           0.98        87
 macro avg       0.98      0.97      0.98        87
weighted avg       0.98      0.98      0.98        87
```

After the task is complete, the program re-enters the option menu event loop.

5- Predict a link

The program prompts the user to input a link, which is then crawled to extract its textual content. The content is then vectorized using the same vectorizer that was utilized during classifier training. The program then loads the saved classifier and uses it to make a classification prediction, printing each of the three program topics along with a corresponding probability/confidence score indicating the likelihood of the link belonging to that topic.

▼ Example output

```
Enter option: 5
Predicting a link...
Enter link: http://sten.astronomycafe.net/2023/03/
```

```
Predictions
Astronomy: 100.0%
Health: 0.0%
Economy: 0.0%
```

After the task is complete, the program re-enters the option menu event loop.

6- Your story!

The program displays a brief message containing general information about the search engine and details about how we have applied our knowledge from the course to build the various components of this system.

▼ Output

```
Enter option: 6
This search engine program is designed to provide users with the basic
functionality for collecting, indexing, classifying, and searching for documents
while leveraging our knowledge of search engine architecture and information
retrieval gained from this course. We applied course concepts including web and
document crawling, text processing, indexing, query processing, classification and
retrieval models to develop its various components.

For instance, to collect new documents, we implemented a web crawler algorithm
that uses BeautifulSoup to parse text, store documents and extract links from web
pages. To index documents, we used an inverted index data structure that stores
the frequency of each term in the document to enable fast retrieval of search
results. Additionally, we used the term-at-a-time algorithm to retrieve documents
related to the query, and applied text processing techniques such as tokenization,
stemming, stop-word removal, and spell correction to improve the accuracy of
search engine results.

We applied our knowledge of supervised machine learning methods to train a
classifier model capable of classifying documents into one of the three program
topics. To train the model, we utilized a K-Nearest Neighbors (KNN) classifier
based on the text content of both labeled source pages and internal links found
within them. This model is used to identify which documents are most relevant to
the user's query. Additionally, the same classifier model is used for the link
prediction feature to predict which of the three topics a given link belongs to.

Overall, this simple search engine program is the culmination of our in-depth
understanding of search engine architecture and information retrieval techniques
that we gained throughout this course. Our ability to effectively implement
various algorithms and processes is reflected in the program's performance and
utility.

Thank you for taking the time to explore our program,
- Herteg and Kelvin <3
```

After the task is complete, the program re-enters the option menu event loop.

7- Exit

The option menu event loop will conclude, exiting the progam.

▼ Output

```
Enter option: 7
Exiting search engine...
```

Contributors

- [Herteg Kohar](#)
- [Kelvin Kellner](#)

That concludes our project!

Thank you for taking the time to read our project report. We hope you enjoyed it!