Information Retrieval Assignment 2: Spotify Podcast Search Engine

(Group 36)

1. Overview

Podcasts have seen a significant rise in popularity in recent years. With new podcasts being recorded at a rapid rate constantly it has become increasingly difficult to find a podcast that suits one's interests. In order to solve this problem we have decided to create a search engine for podcasts by using the Spotify Podcast Dataset [1] available online.

The project will include different stages, namely, data collection, indexing, retrieval model development and evaluation. We will aim to develop a number of retrieval models based on different techniques such as content-based filtering and collaborative filtering. This will allow us to generate recommendations and improve the quality of our search engine.

We aim to use the open source tool PyTerrier [2] for indexing purposes. Following some research we made the choice of PyTerrier as it is built on top of Apache Lucene which is a broadly used search engine and i) benefits from efficient indexing ii) can support many different retrieval models and iii) can support different types of data. Having selected an appropriate indexing tool we will then use techniques such as mean average precision (MAP) and normalised discounted cumulative gain (NDCG), f-score or precision recall to evaluate each model.

Finally, once the best-performing model has been chosen we will build a user interface for our search engine, which will make it easier for users to insert their queries and look at their search results. If time allows it we could also potentially incorporate additional features such as sorting or filtering.

2. Data and User Interface (UI)

For the dataset we will be using the Spotify data from the 2020 podcast Track by TREC. This consists of 100,000 episodes in English and Portuguese, however in this instance we will only be analysing English language podcasts. Each of the episodes contain metadata, audio files and transcribed text of the audio files. For the purposes of our project we will be focusing on only the metadata and transcripts. The 2020 track offered two tasks, the query task where applicants were required to find the most relevant podcast segments based on a simple query and then a summarisation task where users were asked to provide a summary based on that query e.g what were discussions and conversations about climate change. We will only be focusing on query based topics, however we will be performing a two stage query as follows 1. A query search to find the most relevant topics pertinent to a user defined query 2. What pieces of dialogue within a chosen podcast most accurately represent the original user query.

The metadata for the Podcast Track contains the following fields:

- Show URI
- Episode URI
- Show name
- Episode name
- Show description
- Episode description
- Publisher
- Language
- RSS link
- Duration

For the initial step we will be focusing on mainly the show description, using this to index and create a text corpus of all the documents In our dataset. This will be indexed offline and stored via some cloud based storage solution which we can access within our main

programme. Once a user query is made we can access this indexed data to query the whole dataset and identify the most relevant episodes related to that query. Our development for the Information Retrieval model will be done through a Jupyter Notebook, assessing the performance of a selection of models on the Tracks training data [3]. We can then make assessments about the relevance using the judgement file within the track. All of this data will be scraped via Beautiful Soup or Selenium and will only be used for development.

Once we have a viable well performing model we will then look at deployment of our user based input queries and UI rather than TREC's pre-defined query examples. The UI will be a front end HTML solution hosted locally at first to avoid the need to host through a cloud based platform such as GCP (Google Cloud Platform). The front end will be a simple interface allowing users to input their queries and retrieve the most relevant podcasts outputted by our model. We will create a Python backend that leverages the Flask package and REST technologies to perform a series of GET requests made by the user.

Once the initial request is made, the user will be redirected to a second page which lists the most relevant podcasts. From this page the user will have the option to select any given podcast to query the details of that podcast and return specific segments of that podcast most relevant to the initial query (another GET request). This will be based on the transcribed data which can be seen below.


```
[{"words": [{"startTime": "0.900s", "endTime": "1.400s", "word": "Welcome", "speakerTag":
1}, {"startTime": "1.400s", "endTime": "1.500s", "word": "to", "speakerTag": 1},
{"startTime": "1.500s", "endTime": "1.700s", "word": "the", "speakerTag": 1},
{"startTime": "1.700s", "endTime": "2.100s", "word": "AIA", "speakerTag": 1},
{"startTime": "2.100s", "endTime": "2.800s", "word": "Vitality", "speakerTag": 1},
{"startTime": "2.800s", "endTime": "3.100s", "word": "One", "speakerTag": 1},
{"startTime": "3.100s", "endTime": "3.400s", "word": "Minute", "speakerTag": 1},
{"startTime": "3.400s", "endTime": "4.200s", "word": "Podcast.", "speakerTag": 1},
```


Due to the Spotify dataset containing more that 1 billion transcribed words, having to index all of these may cause computational difficulties. Therefore the indexing for this section will be done in real time, and will only comprise data for the selected podcast, however we will need to monitor the performance of this as prior to testing this is unknown. There are some initial assumptions made for this section that will need to be assessed upon development. Firstly we will assume the same techniques used for indexing and retrieval as the first section. Secondly, each podcast of transcribed data will need to be split into sub sections, which now is just assumed to be at specific time intervals, e.g every 2 mins. However, in practice there is most likely a better way of doing this but will have to be assessed.

3. Architecture

Figure 1 shows a graphical representation of the architecture used for this task.

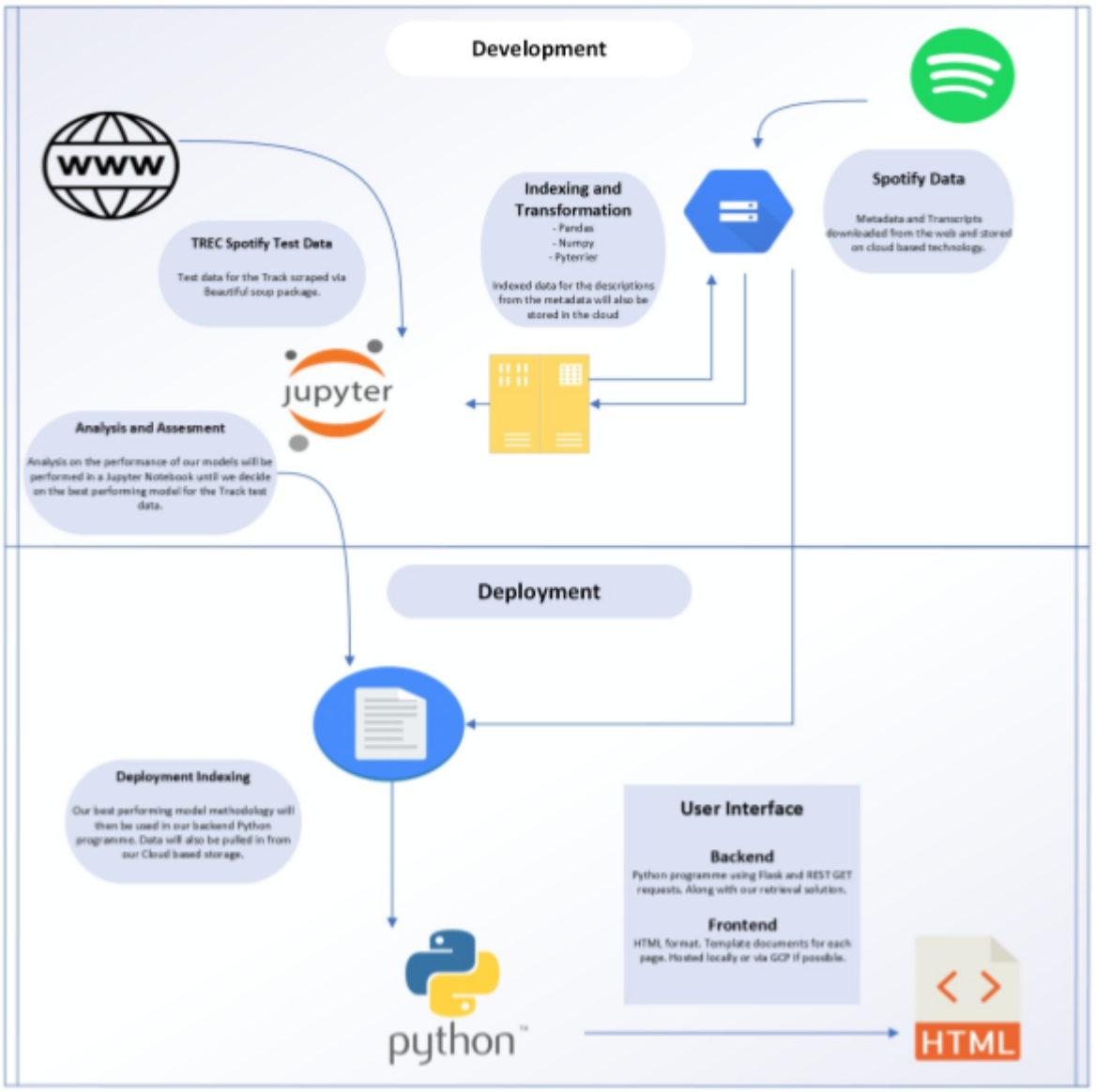


Figure 1: Podcast Search Engine Architecture

- **Spotify Cloud:** Primary source of data. Contains Spotify's Podcast dataset.
- **Spotify Data:** MetaData and Transcripts downloaded from the web and stored on cloud based technology.
- **Jupyter Notebook:** Jupyter notebooks will be used for evaluation of model performance and selection of best performing model.
- **Indexing Framework:** The indexing framework used is PyTerrier.

- **Retrieval Framework:** Data is retrieved by scraping from TREC with the use of Beautiful soup [4].
- **Evaluation Metrics [5]:**
    - **F-Score**

      (2 x precision x recall)/(precision + recall)
    - **R-Precision:**

      R-precision is used to determine the number of documents that are relevant to the query.

      R-Precision = number of relevant documents retrieved/number of relevant documents.
    - **Mean Average Precision (MAP):**

      MAP is used to determine whether a retrieved document is relevant or not.

      For a given set of queries q,

      MAP = average precision for q / number of queries.
    - **Normalised Discounted Cumulative Gain (NDCG) [6]:**

      NDCG = (actual NDCG/ideal NDCG)

4. Tools and Features

Sl.	Task	Software/Tools	Description
1	Development of the Spotify podcast search engine	Jupyter Notebook	
2	Storage and collaborative working	Google Drive and Google Colab	Easy collaboration, data storage, and computational resources.
3	Pandas, NumPy	Pandas, NumPy	**Pandas** and **Numpy** may be needed to process the data and perform computational operations on the data.

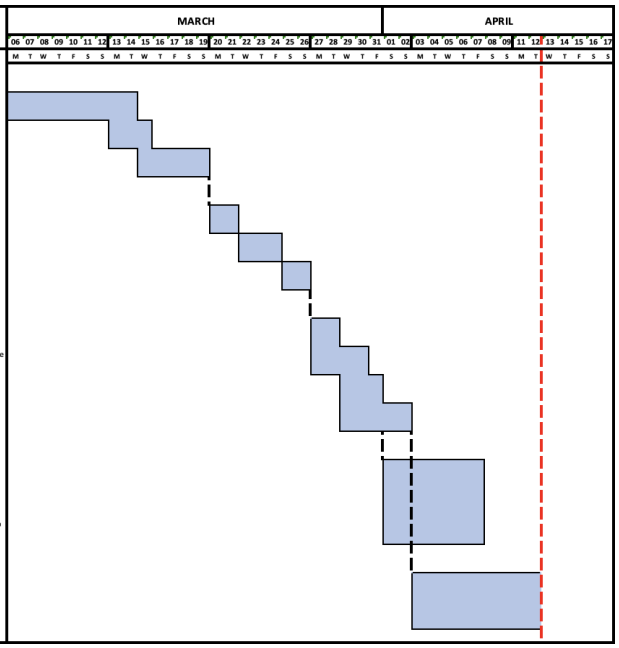
4	HTML	HTML	**HTML** is the standard markup language, it is needed to define the structure and layout. **Flask** is based on Python but also requires the use of **HTML.**
5	Web scraping	BeautifulSoup	**Scraping** is an automated way of retrieving data. In this project it will be necessary to scrape the queries from the TREC conference.
6	Data preprocessing	Re	The '**re**' library provides a simple and effective set of tools for working with regular expressions [7]. NLTK, spaCy could have been used however, as we are working on simple word processing tasks, using re is sufficient.
7	Indexing Query processing Retrieval function Relevance analysis	PyTerrier	Facilitate the implementation of the various stages of processing a search query, including tokenization, stemming, reverse index search, result retrieval, evaluation and analysis. **PyTerrier** offers great flexibility and is compatible with many other data processing tools in Python. Whoosh and Elasticsearch were considered, but **PyTerrier** is simpler to set up and use for small-scale projects and is more suitable for experimenting and evaluating different search models.
8	User interface	Flask	**Flask** to develop a user interface for the search engine.

5. Roles and Responsibilities

Member	Roles
Angelos, T	Retrieval framework, Query representation, Report writing
Daniel, C	Spotify data collection, Design of the database, Data pre-processing, UI development, Evaluation
Kuna, N	Query processing, Report writing
Lucas, B	New library learning, Indexing, Relevance Analysis, Evaluation, compliance with deadlines

6. Time plan (Gantt Chart)

Sl	Task	Timeline	Member	Responsibility
1.	Dataset	2 weeks		
1.1.	Spotify data collection	7 days	Dan	Collect Spotify podcast data, including titles, descriptions, tags and categories
1.2.	Design of the database	3 days	Dan	Organized in a way that allows for efficient retrieval and manipulation of information.
1.3.	Data pre-processing	5 days	Dan	Cleaning and standardising the data collected to prepare it for analysis
2.	Indexing	1 week		
2.1.	Tokenization	2 days	Lucas	Break down the text into words or terms
2.2.	Model selection	3 days	All	Selecting the most appropriate algorithm or set of algorithms to use for ranking the search results
2.3.	Inverted indexing	2 days	Lucas	Create an index that associates with each term of the corpus the list of positions where this term appears in each document
3.	Query processing	6 days		
3.1.	Parsing	2 days	Kuna	Analyse the syntax of the query to extract key terms and search operators.
3.2.	Query parser	4 days	Kuna	Break down the query into meaningful terms, eliminate unnecessary words and organise them into an appropriate data structure
3.3.	Query representation	3 days	Angelos	Transform the query into a standardised representation that allows it to be compared with indexed documents
3.4.	Retrieval framework	4 days	Angelos	Select relevant documents according to the query
4.	Evaluation	1 week		
4.1.	Relevance Analysis	1 week	Lucas & Dan	Assess the relevance of the selected documents to the user and rank them in order of relevance
4.2.	Analysis of the results	1 week	Lucas & Dan	Analyse to understand the research results and identify areas for improvement
4.3.	Presentation of results	1 week	Lucas & Dan	Understand the performance of the research system and make decisions about how to use it
5.	Report and presentation	9 days		
5.1.	Write up	9 days	All	
5.2.	Presentation	9 days	All	



7. References

1. *Podcasts dataset*. Available at: https://www.podcastsdataset.byspotify.com/ (Accessed: March 8, 2023).
2. *Welcome to Pyterrier's documentation!¶ Welcome to PyTerrier's documentation! - PyTerrier 0.9.2 documentation*. Available at: https://pyterrier.readthedocs.io/en/latest/index.html (Accessed: March 8, 2023).
3. (no date) *Text retrieval conference (TREC) podcaststrack*. Available at: https://trec.nist.gov/data/podcast.html (Accessed: March 8, 2023).
4. *Beautiful Soup documentation¶ Beautiful Soup Documentation - Beautiful Soup 4.4.0 documentation*. Available at: https://beautiful-soup-4.readthedocs.io/en/latest/ (Accessed: March 8, 2023).
5. Chandekar, P. (2021) *Evaluate your recommendation engine using NDCG*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1#:~:text=NDCG%20is%20a%20measure%20of,to%20evaluate%20a%20recommendation%20engine. (Accessed: March 8, 2023).
6. *Common evaluation measures - NIST*. Available at: https://trec.nist.gov/pubs/trec14/appendices/CE.MEASURES05.pdf (Accessed: March 8, 2023).
7. *RE - regular expression operations Python documentation*. Available at: https://docs.python.org/3/library/re.html (Accessed: March 8, 2023).