Finding similar items: MeDAL

Isabella Cadisco

September 15, 2023

Abstract

Implementation of a detector of pairs of similar textual objects in a distributed environment.

1 Dataset

The MeDAL dataset is a collection of 14,393,619 articles from the biomedical domain, curated for abbreviation disambiguation and made publicly available on Kaggle. The dataset is organized into three columns: the first column reports the abstracts, the second column reports the indexes referring to the location within the text of the inserted abbreviations, while the third column reports the actual substituted words [1].

2 Problem statement

The objective of this project is to identify pairs of similar objects contained in the MeDAL dataset. Specifically, the pairs of objects whose first element, i.e., article abstract, is similar.

Regardless of the notion of similarity chosen, searching for pairs of similar objects in a set of about 1.4×10^9 results in a process of complexity $O\left(\frac{1}{2}(1.4 \times 10^9)^2\right)$, since the number of pairs to be compared is $\binom{1.4 \times 10^9}{2}$. While considering one millisecond per comparison operation, this operation would take about eleven days. Thus, the first problem encountered is time.

Second, it is necessary to define a notion of similarity: the approach that will be taken in this project is based on lexical similarity, specifically Jaccard similarity, defined as

$$J(obj_i, obj_j) = \frac{|\text{intersection}(obj_i, obj_j)|}{|\text{union}(obj_i, obj_j)|}.$$

To implement this notion of similarity, it is necessary to express textual documents as sets. This is done by adopting the shingling approach, that is, expressing each document as a set of overlapping substrings of atomic units of length k.

The resulting data structure is a matrix, called the Characteristic Matrix, which represents the presence or absence of shingles (or features) in each textual object. Each row of the matrix corresponds to a document, and each column corresponds to a shingle. The matrix cells contain binary values, indicating whether a particular shingle is present (1) or absent (0) in a document.

Considering 20 frequent characters in the English alphabet, the number of possible shingles using this approach is 20^k, a number significantly greater than the average length of abstracts contained in MeDAL, which is 643 characters. The corresponding Characteristic Matrix would be very sparse, as well as occupying k bytes for each stored shingle. Overall, the second problem that is faced is one of space.

3 Methodology

3.1 Space problem

To address the space problem, the Minhash signature technique was leveraged. Minhashing is a method that allows for efficient reduction in storage requirements while preserving Jaccard similarity. In Minhashing, permutations are simulated by employing random hash functions from a predefined set. Each set's Minhash signature is generated by mapping its elements to a number of buckets equal to the size of the set and selecting the minimum hash value among the set's members for each bucket. This process condenses the essential information about each set into a sequence of values, enabling the maintenance of a compact representation of the data while still accurately estimating set similarities.

3.2 Time problem

In addressing the time problem linked to the extensive pairwise similarity calculations, we adopted Locality-Sensitive Hashing (LSH). LSH proved effective by exploiting the locality-sensitive nature of hash functions. Hash functions used in LSH are deliberately designed to be sensitive to the local properties of the data. Specifically, they are constructed in a way that similar items are more likely to be hashed into the same bucket or band compared to dissimilar items. This inherent property allows LSH to significantly reduce the computational time needed to identify similar sets. By dividing the signatures into bands and only measuring

similarity for sets that match in at least one band, we could efficiently prune the search space. Consequently, LSH enabled us to concentrate computational efforts on potential matches, effectively alleviating the time-intensive nature of the task.

4 Proposed solution

The proposed solution is inspired by Fast Document Similarity in Python: MinHashLSH

4.1 Data organization

4.1.1 Data Acquisition and Setup

Apache Spark and PySpark were employed to enable large-scale data processing and analysis on the Hadoop Distributed File System (HDFS).

The full dataset of the MeDAL dataset, stored in a CSV file named `full_data.csv`, was downloaded. As an essential step to expedite rapid testing, a subset of this dataset was randomly sampled. The selection of this subset was driven by the need to ensure efficient analysis while capturing the fundamental characteristics of the complete dataset. To maintain consistency and reproducibility, a fixed seed was utilized for the random sampling process.

4.1.2 Data Transformation

Once the dataset was obtained, the following data transformations were applied:

- Specific columns of interest, namely 'TEXT' and 'LABEL', were selected.
- A unique identifier was assigned to each record using the `monotonically_increasing_id()` function, resulting in a new column called 'doc_id'. This identifier facilitates tracking individual records and their corresponding labels during the analysis.

4.2 Pre-processing

4.2.1 Text Pre-processing Pipeline

The pre-processing pipeline consists of the following key steps:

- **SpaCy Integration:** The SpaCy library was leveraged to enhance Natural Language Processing (NLP) capabilities. Specifically, the "en_core_web_sm" model was used to process and analyze English text.

- **NLP Pipeline:** The text data underwent a comprehensive NLP pipeline facilitated by SpaCy. This pipeline involved tokenization, part-of-speech tagging, and lemmatization.
- **Lemmatization and Stopword Removal:** Lemmatization was applied to reduce words to their base forms, ensuring consistency in the representation of words. Additionally, common stopwords and punctuation were removed from the text to focus on meaningful content.

4.2.2 User-Defined Function (UDF)

To seamlessly apply the text pre-processing pipeline to the dataset, a User-Defined Function (UDF) was created in PySpark. This UDF encapsulates the pre-processing steps and enables their application across the entire dataset.

4.2.3 Integration with PySpark

The UDF, named `preprocess_udf_spark`, was registered with PySpark, allowing for easy incorporation into the data transformation process. Consequently, a new column, 'preprocessed_text,' was generated, containing the cleaned and refined text data.

4.3 Space problem: data representation

In this section, the process of data representation is explored, involving the transformation of the pre-processed text data into a suitable format for analysis.

4.3.1 Dataframe Creation

As a first step, a new dataframe was created, containing only the 'doc id' and 'preprocessed text' columns. This simplification enabled a more focused and streamlined representation of the data.

4.3.2 Shingling

To facilitate further analysis, the text data underwent shingling. This technique involved dividing the text into overlapping substrings of a specified length (in this case, 5 characters). The result was a set of shingles for each document, representing distinct substring patterns within the text.

4.3.3 Hashing Shingles

Efficient data processing and analysis necessitated the hashing of shingles. A specific hash family was employed to generate hash values for the shingles. These hash functions were carefully crafted to produce hash values that preserve essential information while reducing the dimensionality of the data.

4.3.4 Minhashing

For dimensionality reduction and efficient similarity estimation, a minhashing approach was applied. Each document's set of hashed shingles was subjected to minhashing, where a hash function was selected from a hash family for each document

4.3.5 Interesting Functions

Two key functions are instrumental in the data representation process:

Function: get_hashed_shingles(shingles_set)

Purpose: The "get_hashed_shingles" function is responsible for hashing a set of shingles, which are substrings of text data, into a more compact representation. This process is essential for reducing dimensionality while retaining the essential information needed for similarity estimation.

Implementation:

- *Input Parameters:* The function takes a single input parameter, "shingles_set," which is a set containing the shingles extracted from a document.
- *Initialization:* Inside the function, a hash function family is created with a specific identifier (in this case, '0').
- *Hashing Shingles:* The function iterates through each shingle in the input set, and for each shingle:
  - The hash function from the hash family is selected.
  - The shingle is hashed using the selected hash function, incorporating a unique "salt" specific to that hash function to ensure distinct hash values for different shingles.
  - The resulting hash values are added to a set to ensure uniqueness.
- *Sorting:* The set of hashed values is sorted to maintain consistency in representation.

- *Result:* The function returns a list of sorted hash values representing the shingles from the input set.

Function: minhash_map(docId_ShingleSet_hFunct)

Purpose: The "minhash_map" function plays a crucial role in the minhashing process. It calculates a minimum hash value (minhash) for a document based on its set of hashed shingles. Minhashing is used for dimensionality reduction and efficient similarity estimation.

Implementation:

- *Input Parameters:* The function takes a single input parameter, "docId_ShingleSet_hFunct," which is a tuple containing:
  - Document ID.
  - Set of hashed shingles for that document.
  - Identifier for the hash function family (unique to that document).
- *Initialization:* The function initializes a variable, "min_h," to positive infinity. This variable will hold the minimum hash value encountered during the calculation.
- *Minhash Calculation:* For each hashed shingle in the input set, the function performs the following steps:
  - Selects the hash function from the hash family based on the document's hash function identifier.
  - Calculates the hash value for the shingle, incorporating the salt from the selected hash function.
  - Compares the calculated hash value with the current minimum hash value ("min_h").
  - If the calculated hash value is smaller than the current minimum hash value, it replaces "min_h" with the new value.
- *Result:* The function returns a tuple containing the document ID and the minimum hash value ("min_h") for that document.

4.4 Time problem: LSH

In this section, the LSH (Locality-Sensitive Hashing) process is discussed. A system of equations is defined and solved using the `fsolve` function from SciPy. These equations are essential for determining critical parameters for LSH, specifically the

number of bands (denoted as b) and the number of rows per band (denoted as r). These parameters are crucial for configuring the LSH process to achieve the desired threshold for candidate pair identification, which in this case, is set to 0.7.

The LSH algorithm involves mapping the document signatures into buckets using a specific hash function. Each document's signature is divided into bands, and for each band, a hash value is computed. The resulting (band, bucket) pairs are used to group similar documents efficiently. This mapping process is executed for all documents in the dataset.

Furthermore, the code generates candidate pairs of documents using a combination of the (band, bucket) pairs. These candidate pairs represent potential similar documents that warrant further analysis. The pairs are carefully generated and deduplicated to ensure that each candidate pair is unique.

4.5 Candidates' Approximate Jaccard Similarity

In this section, candidates' approximate Jaccard similarity is both calculated and analyzed.

The process initiates with the filtration of the signature matrix, specifically to encompass solely the documents identified as candidates for the forthcoming similarity comparisons.

Subsequently, the transition is made to converting the filtered signature matrix into a Pandas DataFrame, a step that greatly facilitates the manipulation of data. This transformed DataFrame conveniently houses document IDs and their respective MinHash values.

The computation of Jaccard similarity for pairs of candidate documents ensues, with an iterative journey through unique pairs of document IDs. For each such pair, their corresponding MinHash values are extracted from the DataFrame and adeptly transformed into sets. This transformation lays the groundwork for the calculation of approximate Jaccard similarity.

As for illustration, here's an exemplar of a similar pair to exemplify the concept. Notably, the *smokers* and *exposure* labels emerge as a common thread.

DOC1: the placenta plays an important role in modulating xenobiotic passage from mother to fetus studies in mice have demonstrated that placental abcb and abcg ... activity or expression between sm and nonsmokers in summary both abcb and abcg are expressed at high C2 in human placenta and are functionally active suggesting a protective role with respect to fetal SE to xenobiotics ingested by the mother

LABELS_DOC1: vesicles, smokers, smokers, levels, exposure

DOC2: it has been suggested that ingestion of polycyclic aromatic hydrocarbons pahs may contribute to the high incidence and mortality of esophageal CA in linxian china to explore this relationship a semiquantitative immunohistochemical staining method was developed for localization of pahdna ... and samples obtained at autopsy from sm and nonsmokers in the united states nuclear pahdna staining was absent from linxian samples when serial sections were incubated with normal rb SS negative control and was significantly reduced on incubation with bpdedna antiserum absorbed previously with the immunogen bpdedna these results appear to support the hypothesis that high pah exposure levels may be etiologically associated with the development of esophageal cancer in linxian

LABELS_DOC2: cancer, keratinocytes, exposure, smokers, rabbit, serum

5 Issues and further developments

The choice of k for shingle size was made considering a general rule based only on document length. Given that scientific texts tend to have words in common even in different domains, a more valid approach might have been Bag Of Words in conjunction with TD-IDF.

In addition, the banding technique employed in LSH could have been improved by using amplification techniques aimed at identifying candidate pairs with higher similarity.

References

[1] Z. Wen, X. H. Lu, and S. Reddy, "Medal: medical abbreviation disambiguation dataset for natural language understanding pretraining," *arXiv preprint arXiv:2012.13978*, 2020.

1

[1]I declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.