UNIVERSITÀ DEGLI STUDI DI MILANO

Algorithms for Massive Data Analysis Paper
Finding similar items project
Privacy, Data Protection, and Massive Data Analysis in emerging scenarios

Ekaterina Sergeeva
ekaterina.sergeeva@studenti.unimi.it

November 2024

Abstract

The goal of this project was to develop a detector to identify similar items by analyzing rows in the job summary column of a CSV file, which includes descriptions of job postings from LinkedIn platform. To achieve this, a MapReduce framework was employed to compute Locality-Sensitive Hashing (LSH) through Minhash signatures using Apache Spark.

Contents

1 Dataset downloading and Preprocessing

The project's objective was to develop a detector capable of identifying similar items. The dataset was downloaded via the Kaggle API directly on the Google Colab platform, containing two columns with links and descriptions of LinkedIn job postings. Following the installation of required packages, library imports, and the initialization of the Spark session, two approaches were applied to prepare data for analysis.

Firstly, a schema was defined to specify the exact data types of the job link and job summary columns for Spark. Additionally, Spark was configured to correctly handle special characters, quotes, and multiline fields to prevent content from merging across the two columns, which occurred during the initial attempt. Since this approach did not fully satisfy the preprocessing need, a second solution was implemented. This involved managing null values, replacing newline characters (\n) with spaces, and splitting each job description into an array of words. This final step allowed for handling problematic rows while retaining job descriptions containing more than five words. The following is an example of the Spark DataFrame structure:

```
+--------------------+------+
|           job_words|doc_id|
+--------------------+------+
|[Rock, N, Roll, S...|     0|
|[Schedule, :, PRN...|     1|
|["Description, In...|     2|
|[, we, encourage,...|     3|
|[Commercial, acco...|     4|
|[Address:, USA-CT...|     5|
|[Description, Our...|     6|
|[Company, Descrip...|     7|
|[An, exciting, op...|     8|
|[Job, Details:, J...|     9|
|[Our, Restaurant,...|    10|
|[Our, General, Ma...|    11|
|[Earning, potenti...|    12|
|[Dollar, General,...|    13|
|[Restaurant, Desc...|    14|
|[Who, We, Are, We...|    15|
|[A, Place, Where,...|    16|
|[Description, The...|    17|
|["Overview, Descr...|    18|
|[, the, Team, Mem...|    19|
+--------------------+------+
only showing top 20 rows
```

Figure 1: Example of Spark Dataframe

To enhance computational efficiency, a sample of the dataset (0.0005) was used, as processing the full dataset with Spark would require a substantial amount of time. Each job description was preprocessed using the NLTK package to remove punctuation, stopwords, extra spaces, and to tokenize and convert all text to lowercase. Although some approaches include stopwords in shingle creation, they were excluded in this case.

After preprocessing, an RDD (Resilient Distributed Dataset) was created. An RDD is a fundamental data structure in Apache Spark, representing data as a collection that can be processed in parallel across clusters. It is immutable, meaning transformations generate a new RDD; it is also partitioned and fault-tolerant, allowing Spark to recover from node failures. In this project, the RDD was derived from the DataFrame by using the .rdd attribute, where each row corresponds to a record in the DataFrame. A transformation was then applied using .map, which applies a lambda function to each RDD row, outputting a tuple containing the document ID and the associated with it job description text.

2 Implementation of the algorithm

In the the algorithm's implementation, a Minhash signature was generated for each job description to identify candidate pairs of similar descriptions by measuring their Jaccard similarity.

2.1 Shingling

In this dataset, a shingle is a substring of length k derived from the words in each job description. The shingle length was chosen based on the nature of the job descriptions, ultimately set to 7. Shingling was performed on the RDD using .flatMap, which applies a lambda function to each element and flattens the output. For each document, this function generated a list of tuples containing the document ID and each shingle extracted from the sliced words, ensuring full coverage across the document.

The resulting output was a flattened list of tuples with document IDs and corresponding shingles. To optimize computation, these shingles were then hashed to fit within a 32-bit integer. This produced another RDD where each output tuple

consisted of the document ID and the hashed shingle value. All hashed shingles were then combined with each unique document, collected as a single list to streamline further processing.

2.2 Minhashing

To compute the Minhash signature, several random hash functions were generated to simulate random permutations. These hash functions, created using random parameters a, b, and c, were stored in a list. For each shingle, each hash function computed a hash value, and the smallest values for each hash function were retained. This process produced Minhash signatures for each document, resulting in a compact representation of the sets of hashed shingles.

2.3 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) is an algorithm designed to efficiently locate similar pairs by examining only those likely to be similar, rather than comparing every possible pair. This is achieved by hashing similar items into the same buckets. In this project, LSH was implemented using the MinHashLSH class in Apache PySpark. Minhash values were converted to vectors, and the model was fitted to a DataFrame.

The banding technique was employed, with the number of bands set to 8. This parameter is essential as it determines the partitioning of the Minhash signature; each band contains a row of the signature, and if two sets are similar, at least one of their bands should hash to the same bucket. A threshold, determined by the universal formula, based on the bands and rows, was set around 0.84. However, it was subsequently manually lowered to 0.6 in order to obtain more relevant results.

To find similar job descriptions, the MinHashLSHModel in Apache PySpark was applied, using the approxSimilarityJoinmethod, which identifies pairs of rows with distances below the defined threshold. Duplicate pairs and those that did not meet the similarity threshold were removed. The primary objective was to identify pairs of documents with similar Minhash signatures by comparing each document's signature with those of all others through the joined DataFrame created by the MinHashLSH application.

3 Results

Lastly, a Pandas DataFrame was created with three columns: two columns representing the document IDs and a third column displaying the Jaccard distance between them, illustrating that the task was successfully accomplished. A visual example is provided below.

	doc_id1	doc_id2	jaccard_distance
0	106539	1194679	0.484848
1	125621	637191	0.390000
2	125621	1178559	0.380000
3	249104	443170	0.360000
4	443170	1445985	0.524390
...	...	...	...
3862	524299	1614871	0.500000
3863	637191	1572681	0.456790
3864	1214720	1445985	0.552941
3865	906430	973215	0.576923
3866	853215	1440865	0.350515

3867 rows × 3 columns

Figure 2: Jaccard Dataframe

References

[1] A. Rajaraman e J. Ullman, Mining of Massive Datasets (2017)