UNIVERSITÀ DEGLI STUDI
DI MILANO

Algorithms for massive data
Finding similar items

VICKYSINGH BAGHEL
vickysingh.baghel@studenti.unimi.it

March 2024

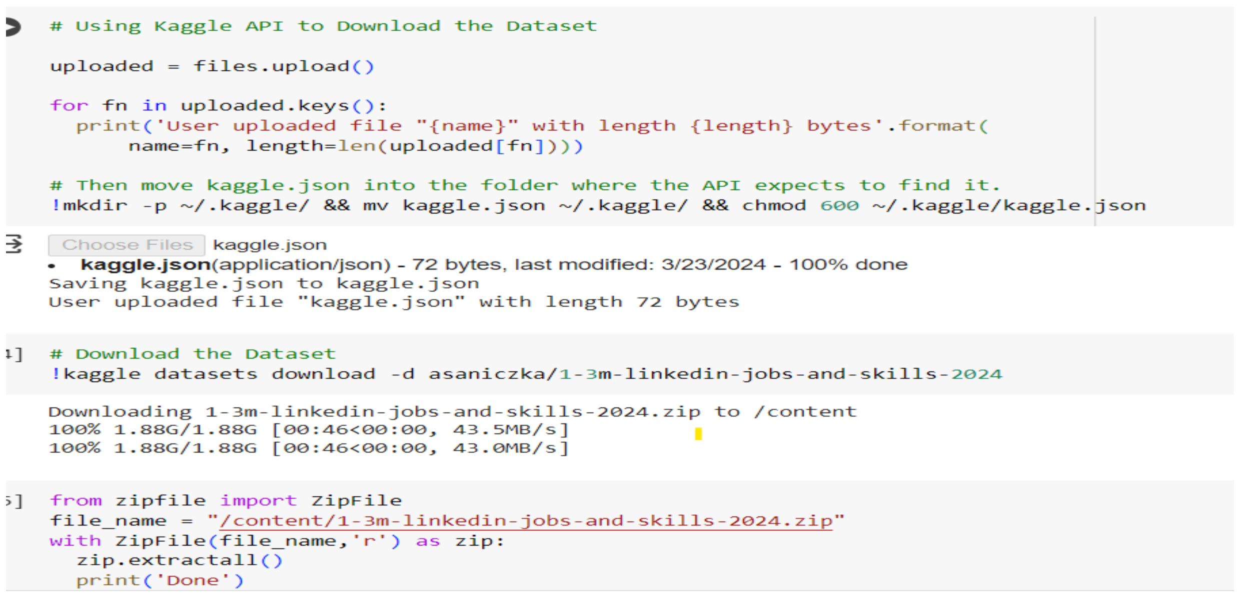
DECLARATION

I/We 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, and including any code produced using generative AI systems. 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.

Contents

1. Data Description:

The aim of the project is to implement a detector for pairs of similar elements that analyzes the data set of 1.3M Linkedin Jobs & Skills (2024) published on Kaggle. It prompts the user to upload a file, then moves the kaggle.json file to the correct folder and changes the permissions to allow access. The downloaded dataset can then be accessed and used in the rest of the code. The first step is to obtain the dataset for the “Finding Similar Items” project. Kaggle provides a direct download option via the command line using the credentials and access key. This makes it easy to access and download the dataset without having to manually search for it on the website. You can get the dataset quickly and work on your project with minimal effort.



```
# Using Kaggle API to Download the Dataset

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))

# Then move kaggle.json into the folder where the API expects to find it.
!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json
```

```
Choose Files  kaggle.json
•  kaggle.json(application/json) - 72 bytes, last modified: 3/23/2024 - 100% done
Saving kaggle.json to kaggle.json
User uploaded file "kaggle.json" with length 72 bytes
```

```
# Download the Dataset
!kaggle datasets download -d asaniczka/1-3m-linkedin-jobs-and-skills-2024
```

```
Downloading 1-3m-linkedin-jobs-and-skills-2024.zip to /content
100% 1.88G/1.88G [00:46<00:00, 43.5MB/s]
100% 1.88G/1.88G [00:46<00:00, 43.0MB/s]
```

```
from zipfile import ZipFile
file_name = "/content/1-3m-linkedin-jobs-and-skills-2024.zip"
with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')
```

Figure 1: Dataset Downloading

2. Data Preprocessing

The dataset contains approximately 48,219,735 records. We noticed that a lot of data was missing, especially in the 'Job summary' column. So we cleaned up the dataset by removing these missing entries so that we now have 15,572,837 records. Using the NLTK library, we analyzed the number of sentences in each job summary and found quite a variety, from single words to multiple sentences. To focus our analysis, we decided to filter out entries with less than 10 sentences. Although we switched to PySpark for faster processing, processing such a large dataset still took time. To speed things up, we narrowed it down to the top 500 entries with more than 10 sentences each. I then create an index for each row as a unique identity for each job description. Then I remove stop words from the sentences using the NLTK library.

```
+-----+--------------------+--------------+--------------------+--------------------+
|Index|         job_summary|sentence_count|               lower|       filtered_text|
+-----+--------------------+--------------+--------------------+--------------------+
|    1|Job Title: Senior...|            31|job title: senior...|job title: senior...|
|    2|POSITION SUMMARYT...|            25|position summaryt...|position summaryt...|
|    3|At Dominion Energ...|            11|at dominion energ...|dominion energy l...|
|    4|Overview Discover...|            24|overview discover...|overview discover...|
|    5|POSITION ELIGIBLE...|            13|position eligible...|position eligible...|
|    6|CaroMont Health i...|            16|caromont health i...|caromont health h...|
|    7|Hello, nurses! We...|            22|hello, nurses! we...|hello, nurses! re...|
|    8|Hello, nurses! We...|            21|hello, nurses! we...|hello, nurses! re...|
|    9|Overview Home Hea...|            11|overview home hea...|overview home hea...|
|   10|"Description****I...|            24|"description****i...|"description****i...|
|   11|"Description**In ...|            21|"description**in ...|"description**in ...|
|   12|Overview Advanced...|            23|overview advanced...|overview advanced...|
|   13|Job Summary:Manag...|            29|job summary:manag...|job summary:manag...|
|   14|BE PART OF SOMETH...|            37|be part of someth...|part something mo...|
|   15|Hello, nurses! We...|            22|hello, nurses! we...|hello, nurses! re...|
|   16|R-00120212 Descri...|            24|r-00120212 descri...|r-00120212 descri...|
|   17|"Summary The Iowa...|            37|"summary the iowa...|"summary iowa cit...|
|   18|"If you enjoy wor...|            21|"if you enjoy wor...|"if enjoy working...|
|   19|Position Title: C...|            49|position title: c...|position title: c...|
|   20|"Our Company**Gen...|            17|"our company**gen...|"our company**gen...|

+-----+--------------------+--------------+--------------------+----------------
```

3. Implementation:

3.1 Jaccard Similarity:

The Jaccard similarity method, which calculates how much two sets overlap by dividing the size of their intersection by the size of their union. This gives us a value between 0 and 1, where 0 means that there is no similarity and 1 means that they are identical. It is a versatile metric that works with both numeric and text data, making it ideal for our comparison task.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

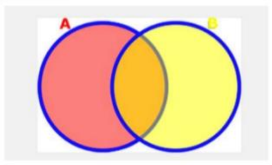


Figure 2: Jaccard Similarity formula

3.2 Minhash:

Minhash a set represented by a column of the characteristic matrix, choose a permutation of the rows. The minhash value of any column is the number of the first row in the permuted sequence in which the column has a 1.

3.3 Minhash Signature:

Technique for estimating the similarity between sets by decomposing them into shorter signatures. The process involves selecting a set of random hash functions and permuting the elements within each set according to these functions. The minimum hash value of all permutations is then chosen as the signature for that set. This compact representation enables efficient estimation of similarity between sets and is therefore particularly useful in scenarios with large datasets or high-dimensional data.

3.4 Implementation:

I started by converting the text into shingles. After calculating the shingles, I continue with the implementation. I perform a cross join operation on an index and calculate the Jaccard similarity between shingles. After calculating the similarity, I filter the results based on the index and apply another filter for records with more than 20% similarity

After calculating the Jaccard similarity based on shingles, my next step is to implement Minhash and create a signature matrix. I will then calculate the similarity of the signature matrix. This approach allows for a more efficient comparison of documents by reducing the dimensionality of the data while preserving the similarity information.

4. Result :

4.1 Jaccard similarity on shingles:

The Jaccard similarity calculated directly from shingles on job_description pairs. In each line, two indices are displayed together with the corresponding Jaccard similarity value.

```
+-----+------+-----------------------------+
|Index|Index2|jaccard_similarity_onshingles|
+-----+------+-----------------------------+
|    2|    32|                   0.94860816|
|    7|     8|                    0.7690355|
|    7|    15|                   0.77692306|
|    8|     7|                    0.7690355|
|    8|    15|                        0.872|
|   10|    74|                   0.25268817|
|   10|    82|                   0.28592163|
|   10|   102|                   0.31753555|
|   10|   167|                   0.30733946|
+-----+------+-----------------------------+
```

For example, the first row shows that 2nd job description and 32 have a Jaccard similarity of about 0.94, indicating a high degree of overlap between their respective shingle sets. Similarly, the following rows display the similarity values between each other.

4.2 Jaccard similarity on Signature matrix:

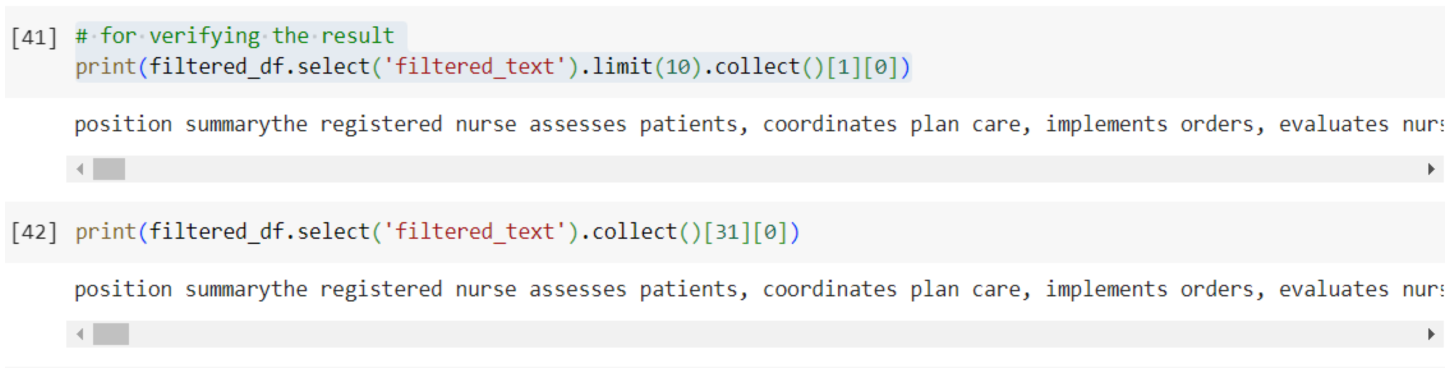
The Jaccard similarity is calculated for the resulting signature matrix, and rows whose index is not equal are filtered out. Finally, entries with a Jaccard similarity of more than 20% are retained. Overall, this method efficiently estimates the similarity of documents using MinHash signatures and Jaccard similarity.

```
Index|Index2|jaccard_similarity_signature|
+-----+------+----------------------------+
|    2|    32|                    0.924812|
|    7|     8|                   0.6516129|
|    7|    15|                  0.68421054|
|    8|     7|                   0.6516129|
|    8|    15|                   0.7777778|
|   10|   102|                  0.21904762|
|   10|   167|                  0.21327014|
|   10|   400|                  0.27363184|
|   10|   425|                  0.23671497|
|   10|   450|                  0.20754717|
|   10|   459|                  0.22488038|
|   10|   464|                  0.23671497|
|   10|   470|                  0.24271844|
|   12|    22|                  0.69536424|
|   12|    36|                  0.52380955|
|   13|    29|                   0.5802469|
|   13|    35|                   0.9541985|
|   13|    98|                   0.5802469|
|   13|    99|                   0.9541985|
|   15|     7|                  0.68421054|
+-----+------+----------------------------+
```

The Jaccard similarity values calculated based on MinHash signatures for job_description. Each row shows two jobs together with the corresponding Jaccard similarity value. For example, the first row shows that job 2 and 32 have a Jaccard similarity of about 0.92, indicating a strong overlap between their respective shingle sets.

4.3 Result Comparing with actual string:

A final comparison of the 2nd job_description with the 32nd job_description reveals a considerable overlap between the strings contained in the two indices. This result indicates that our method of calculating similarity works effectively, as it correctly identifies and quantifies the similarities between these two documents.



```
[41] # for verifying the result
     print(filtered_df.select('filtered_text').limit(10).collect()[1][0])

     position summarythe registered nurse assesses patients, coordinates plan care, implements orders, evaluates nurs
```

```
[42] print(filtered_df.select('filtered_text').collect()[31][0])

     position summarythe registered nurse assesses patients, coordinates plan care, implements orders, evaluates nurs
```

Figure 3: 2nd Index comparing with 32 Index

5. Scalability :

The analysis was performed with PySpark. PySpark is the Python API for Spark, a powerful data processing framework designed for processing large data sets. By using a distributed environment and running code on a Java virtual machine, Spark efficiently processes large datasets. The dataset was converted into a Spark DataFrame, which enables scalable data processing and analysis. In addition, the code was developed in Google Colaboratory using Jupyter Notebooks. This platform offers improved sharing and utilizes Google's servers, eliminating the need for local installations and enabling collaborative work on data analysis tasks. I used 2 approaches for calculating similarity. Both approaches offer advantages in scalability. The MinHash signature method is particularly well suited for large datasets and high dimensional data where exact similarity calculations are not feasible.

6. Conclusion:

When we compare the results of both analyses, we find that Jaccard similarity scores obtained directly from shingles tend to be higher than those derived from MinHash signatures. This discrepancy could be attributed to the inherent differences in the methods. Direct shingle-based analysis calculates similarity based on the overlap of exact shingles, while MinHash signatures provide an approximate measure, especially when a limited number of permutations are used. Despite the different results, both approaches provide valuable insights into the similarity relationships between documents, with each method having its own strengths and drawbacks.

Based on the results, we can say that minhasing is well suited for our study.

References:

Book : Mining of Massive Datasets (Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman

https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function/

Similarity of documents: Shingles,MinHashing and jaccard similarity by Khushboo Khurana.

https://spark.apache.org/docs/latest/api/python/getting_started/index.html