Algorithms for massive data (DSE) 2023/2024

Project: Finding Similar Items

Task: The task is to implement a detector of pairs (or sets of higher cardinality) of similar job descriptions, analyzing the «LinkedIn Jobs & Skills» dataset, published on Kaggle. The detector must consider the job_summary column of the job_summary.csv file and output the pairs/sets of summaries inferred as similar[1].

Student: W

Project outputs: 1. Report. 2. Github codes[2].

Contents

[1] https://malchiodi.di.unimi.it/teaching/AMD-DSE/2023-24/, where the pandas version code and spark version code are included. For clarity and brevity, the codes which generate various comparisons adjusting certain parameters are not included, since they are derived from the base codes published. Most of the additional tests codes generate partial results and could be listed in the further
[2] https://github.com/StevenFromUnimiMIEDSE/UNIMI-DSE-Algorithm_for_massive_dataset/tree/main

Dataset overview

The job_summary.csv contains 1.29 million job links and its corresponding job summaries. The data is organized in two columns, one as job_link and the other as job_summary. In both columns, the textual data contains characters, words, symbols, etc.

General reference note

This project has taken the teaching materials of professor Malchiodi[3], and additional teaching materials indicated by the course[4]. The coding part has taken official documentation of pandas[5], spark[6][7], etc, as base, and improved during the iterative process. Although the final version of the code is on colab, the test on local machine has offered certain insight to understand the limitation of local computation[8], where debugging references have been taken from various sources[9].

Project General Design

General flow of analysis

The data processing based on pandas and spark are both used in this project. The former offers easy data manipulation framework and possibility to get quick insight on sample. The later offers tools to scale up the data processing, especially when sample ratio goes above 0.1%[10]. Based on different tests, distributed computing capability of spark would be the solution to scale up the similarity finding algorithm.

Either using pandas or spark, the general analytical workflow does not vary at high level. In general the procedure consists of, setting up proper environment variable for python (more important in local machine, and simpler in Colab environment) to work, setting up Kaggle environment/API, loading dataset, examining property of the data, cleaning data,

[3] https://malchiodi.di.unimi.it/teaching/AMD-DSE/2023-24/
[4] http://infolab.stanford.edu/~ullman/mmds.html
[5] https://pandas.pydata.org/docs/
[6] https://spark.apache.org/docs/latest/, the shown citing sources may not exhaust in detail level, but most of the code structures are inferenced from official documentations.
[7] Examples: https://spark.apache.org/documentation.html, https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html, https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.show.html, etc.
[8] In general, the configuration in local machine is more complicated than cloud environment. Below certain threshold the local machine has some advantages, but this conclusion requires further tests and logging of processing statistics.
[9] The debug references are taken from stakflow, indicating the difficulty of implementing similar algorithm in local machine. A few examples are (frequent error in local environment): debug ref 1: https://stackoverflow.com/questions/70571389/py4jjavaerror-an-error-occurred-while-calling-zorg-apache-spark-api-python-pyt/76042851#76042851 debug ref 2: https://stackoverflow.com/questions/30013254/can-anyone-explain-my-apache-spark-error-sparkexception-job-aborted-due-to-stag
[10] Latest colab execution on pandas version code, shows below resource usage level: System RAM 8.2 / 12.7 GB Disk 35.1 / 107.7 GB.

similarity analysis(different techniques may differ from each other about analytical details[11]), visualization of analytical results(text information, visual graphical information, etc).

After local machine[12] trial and test[13], it is obvious that, pandas has limited ability to handle the entire or large portion of the dataset in local environment, while spark can handle medium range(from 10k to 100k records[14]) with proper setting(config, in most spark configuration parameter, to optimize local computation).

After utilizing colab[15], the computation speed and efficiency improved(in general). The fastest pandas version code execution reached 3 to 4 minutes on 0.1% sample fraction size(1297 records), considering additional query and visual graphical processes. While the spark version has handled 1% sample fraction size, although the execution time is much higher due to SQL manipulation and UDF efficiency drags.

Cloud, pandas, spark conclusion

Finally, I would assume that, using pandas first to test algorithms in memory, could get result faster, then using spark to scale up the project, would be an ideal choice to this project.

Project code

The project code is saved in https://github.com/StevenFromUnimiMIEDSE/UNIMI-DSE-Algorithm_for_massive_dataset, with option to run directly in colab.

Theoretical background of jaccard similarity, shingling, min hashing, LSH[16]

1. **Jaccard similarity[17]**

   The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:

$$\mathrm{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

   Jaccard distance:

[11] Although in this project, the core algorithm to determine similarity is LSH which simulates Jaccard similarity, and, for test purpose, cosine and other similarity algorithms were tested, and no significant differences were found.
[12] Processor Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz 3.00 GHz, Onboard RAM 32.0 GB (31.8 GB available), System Type 64-bit operating system, x64-based processor
[13] Without techniques like spark SQL, local machine struggles to complete analysis of large dataset. Storage insufficiency, unsupported operation exception, etc, are common errors during runtime.
[14] In local machine, the maximum volume which was successfully processed reached 100k records. In colab, the maximum volume handled successfully reached 1% of the entire dataset, and further tests could be done in the future.
[15] Python 3 Google Compute Engine backend, CPU, 12.7G RAM, 107.7G Disk
[16] J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org
[17] Reference: same as point 9 and other teaching materials from the course

$$d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

This is the similarity measure for shingles.

2. **3 Steps to find similar pairs**[18]
   - **Shingling:** Convert documents to sets
     - Using hashing to assign each shingle an ID
   - **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
     - Using **similarity preserving hashing** to generate signatures with property
       $$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2).$$
     - We used hashing to get around generating random permutations
   - **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
     - Using hashing to find **candidate pairs** of similarity ≥ **s**

Similarity search by pandas[19]

1. Kaggle environment variable setting, data download
Setting up environment variables to get access to Kaggle API. Download the datasets, unzip for further analysis. Load necessary dependencies.

2. Analysis[20]

2.1. Load the data and check properties.

2.2. Process text to lower case, remove simbols, etc
Get shingles from the processed text, Separate entire text to pieces of equal length character chunks.

2.3. Hash each shingle[21] into a hash value, apply minhash

2.4. Setting up bands and rows parameters and calculate threshold of similarity, to minhash value vectors

2.5. Transform minhash values according to ban and row parameters, apply LSH, to band hashed values

2.6. Creating buckets and find candidate pairs

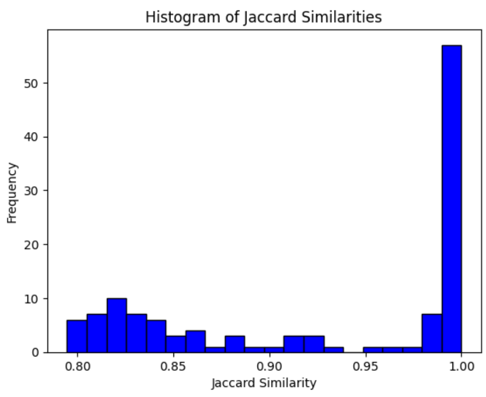
2.7. Calculate pair wise Jaccard similarity value on shingle level for demonstration, show similar pair examples, visualize histogram and heat map

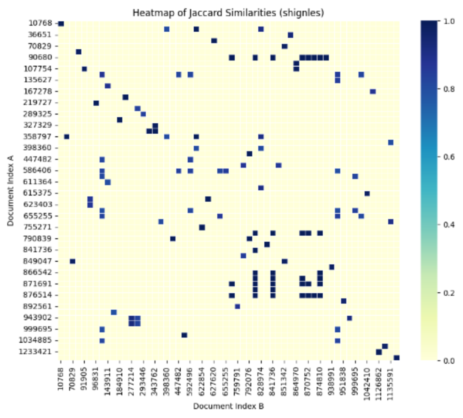
[18] Reference: same as point 9 and other teaching materials from the course

[19] Either in pandas version or spark version, certain functions have undergone multiple iterations and could not be the most efficient version.

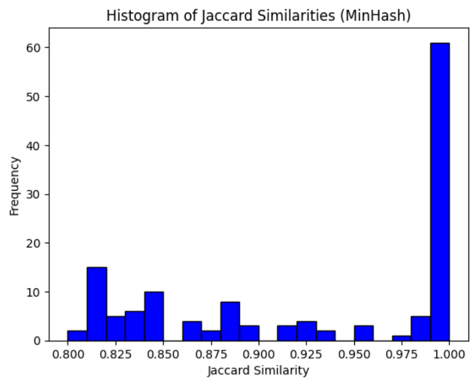
[20] Parameter settings are not comprehensively listed here for brevity

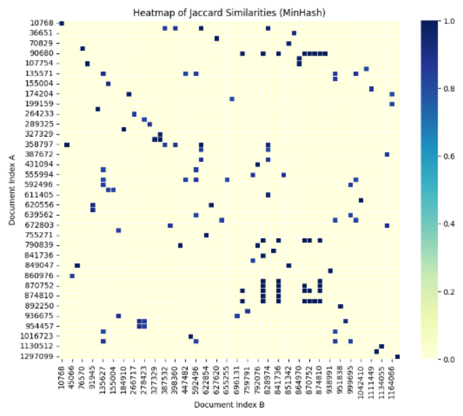
[21] The final length of shingles is referred from the lecture material, since k=10 could be suitable to handle long context.





2.8. Calculate pair wise Jaccard similarity value on minhash level for demonstration, show similar pair examples, visualize histogram and heat map





2.9. Compare sample dataset and similarity pairs statistical information[22]

In the case of 0.1% sample fraction size, similarity level indicates that LSH algorithms generate similar pairs that match the inherent similarity threshold. In addition, similar pairs from shingle level is 123, 0.948% of sampled dataset. On minhash level the candidate pair count is 134, 1.03% of the sampled dataset. As tested and expected, jacard similarity on minhash level, where data is compressed more than shingle level, could generate sparse similarity clusters as shown in the histogram, and there is potential risk of false positive pairs since the count is higher than the shingle level similar pair counts.

Then, after examining band level similarity, the count is 89 pairs, which shows its inclination to eliminate false positives. But, such result could not be concluded without further tests. For example, after checking some sample pairs, there is chance where LSH loses some information of similar pairs, and can not identify the similar pairs.

Similarity search by Spark

1. Data downloading

It is the same as pandas version.

[22] similar pairs info is saved in the notebook

2. **Kaggle environment variable setting, data download**
   Setting up environment variables to get access to Kaggle API. Download the datasets, unzip for further analysis. Load necessary dependencies.
3. **Spark environment setup**
   Setting up environment variables to get initiate spark session. In local environment, this setting would best utilize local compute efficiently. In cloud environment, it could be optional in most cases[23].
4. **Analysis**
   - 4.1. Define schema, sample data, constructing datafrome.
   - 4.2. Process text information.
   - 4.3. Get shingles, Separate entire text to pieces of equal length character chunks.
   - 4.4. Minhash shingles to hash values.
     Hash shingles to hash values. The hash functions could vary.
   - 4.5. Transform minhash values according to band and row parameters, apply LSH, to hash banded minhash values to band hashes.
   - 4.6. Creating buckets on band hashes and find candidate pairs in each bucket.
   - 4.7. Calculate pair wise Jaccard similarity value on shingle level for demonstration, show similar pair examples.
   - 4.8. Visualizations, additional checks are omitted due to computation process complexity of SQL query and inefficiency[24].

Scaling records/tests

In google cloud, execution time of the above referred codes ranges from 3 minutes to 2 hours in general, with sample fraction from 0.0001% to 1%. The pandas version usually takes more than 3 minutes to execute all the code, while spark version usually takes 20 minutes or more[25] to execute all the codes(without additional visualization, but with additional stepwise checking).
In local machin, execution time of the above referred codes ranges from 5 hours to 48+ hours runtime. Most of the tests(around 20+ attempts) failed for different configuration reasons and local compute constraints. With sample fraction set as lower or equal to 10%, where partial results could be generated by the full code execution is expected to be much longer.

[23] During various tests, when colab environment's compute quota changes, setting memory allowance, and other parameters could improve the general performance.
[24] Organizing data and utilizing UDF could drag the algorithm efficiency. Since a sample property has been examined, in spark session this part is omitted.
[25] https://stackoverflow.com/questions/45311122/why-is-sparks-show-function-very-slow, this link is one possible explanation, regarding how spark show() works and the computation workflow of spark.

Conclusion

Algorithms and packages - Pandas and Spark differences

By comparing pandas and spark processing of the dataset, spark has shown better performance and adaptability to large datasets with distributed calculation design and parallel processing advantage, while pandas still provides sufficient support to small to medium size data handling and offers in memory calculation capability.

In addition, more details in the runtime should be paid attention to. In the similarity search and calculation phases, leveraging the spark related functions could provide better code structure and produce results in a relatively more sufficient way. It is worth noting that, UDF settings could be less efficient than built in function[26].

Analytical results

In previous session, and in the code notebooks. Roughly estimating, the similar pairs job is around 1% of the total job pairs.

Further improvements

Range of scaling

Further research or improvements of the codes to improve the scaling efficiency could be done in the future, especially, when the sample ratio goes above 0.02. And, different tools(hash function, built in functions in spark, and other packages other than spark) to scale computation could be used to achieve better performance.

Parameters tuning

Distribution of similarity scores among pairs could be further analyzed. As tested on shingle size variations[27], the change of each step wise parameters could impact the eventual distribution of similarity scores, which include but not limited to, shingle length, hashing bands of minhash, hash function number and number of bands, etc. In the situation where built in models of spark are used, computation efficiency could improve.

In another batch of algorithm tests, parameters of the algorithms are tested(n-gram, num_hashes, band_size). With a 100 sample size, the jaccard similarity distribution is sensitive to changes of n gram setting.

[26] General testing conclusion. The final version utilizes more UDF than other versions, for clarity of step wise computations.
[27] Figure 3. Due to the computation intensity, such tests are only tested on small sets.

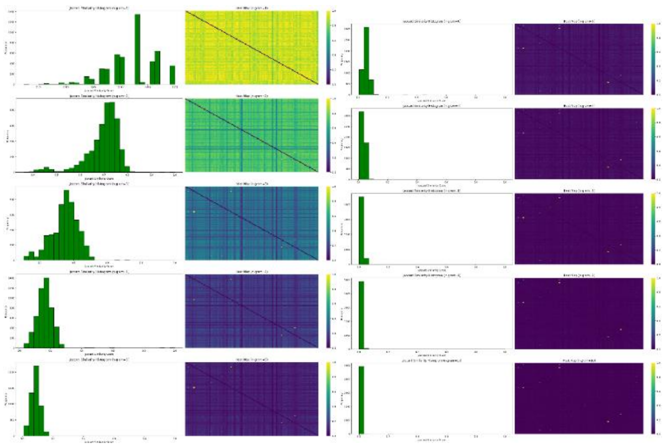


Figure 1 100 sample test results on granular parameter settings change (n-gram) of the similarity algorithm, similarity score distribution in histogram and heat map.

Hashing/data compression method variations

Apart from the standard process, tests on hashing concatenated minhashes to signature set, then hashed to bucket id, or hashing summed minhashes to signature set, then hashed to buckets, were tested. Some similar items could be identified, but the threshold control is not embedded in the workflow.

Other algorithms

In addition to local machine tests, other tests are done before reaching the final version of test and codes. It is interesting to explore more ways of measuring similarity[28], checking the distribution of similarity values[29]. In above figures, some tests were focusing on shingling on words instead of n-gram(fixed token size).

In combination with other techniques like PCA, clustering, more insight could be generated from the analysis on the target dataset. Although potential computation overhead increase is not trivial.

Furthermore, in the future exploration on such dataset, based on implementing different similarity calculation, searching for semantic representations of the dataset could show more properties of the dataset.

Explorations on distributions and visuals

Before the final versions of the code, different techniques have been tried to understand the similarity search choices. These could be further exploration point to compare the different similarity comparison strategies[30].

[28] Jaccard distance, cosine similarity, Euclidian distance, etc.
[29] Histogram of similarity value, heat map to show density of similar pairs.
[30] Cosine similarities, Euclidean and other distance measurements with clustering techniques, etc

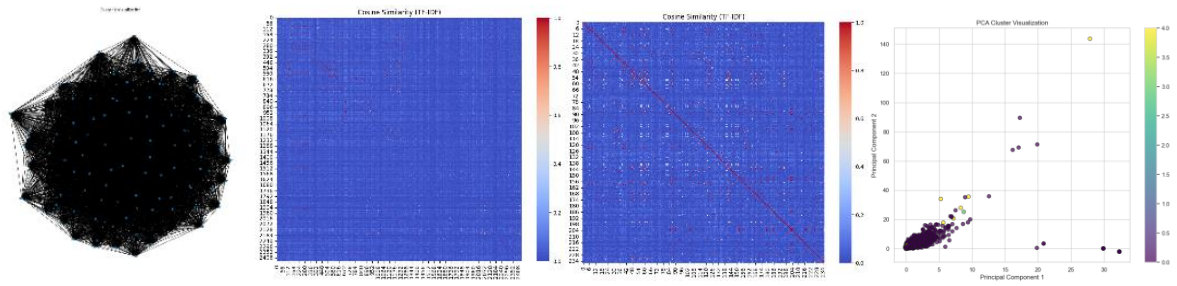


Figure 2 different visualization choices tested in local machine, which could be further implemented on large dataset and cloud environment[31], which includes relationship graphing, cosine similarity, Euclidean distance, PCA.

As shown above, there could be various ways to visualize the similarity cluster, or generate similarity calculation results.

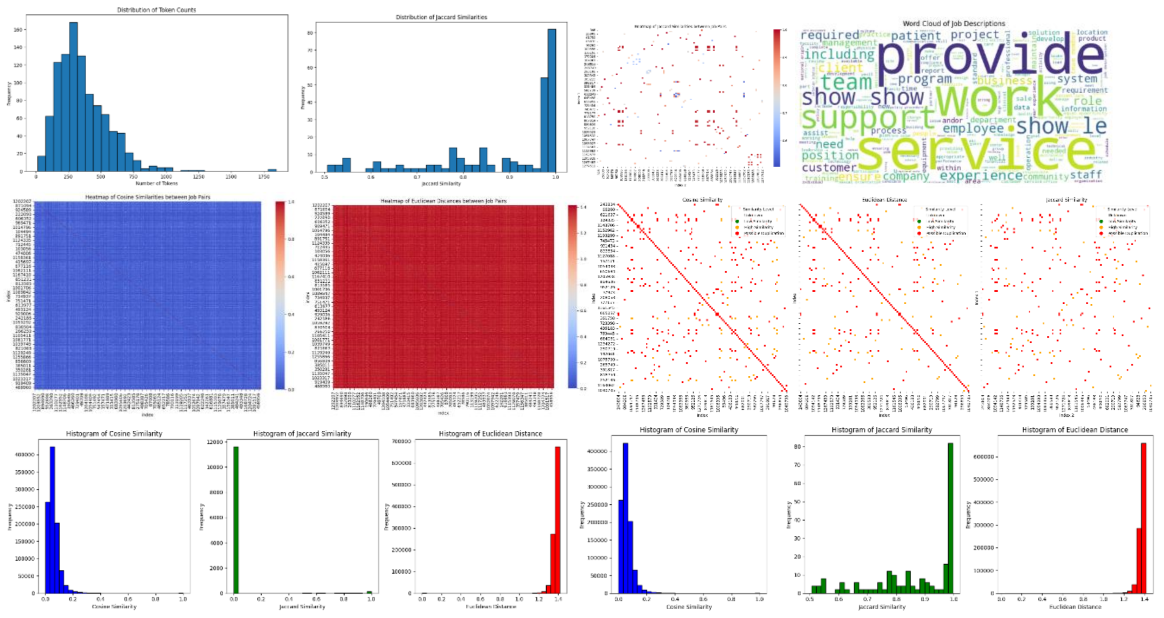


Figure 3 token counts and Jaccard similarity value histogram distribution; Heatmap of Jaccard Similarities between Job Pairs; word cloud; heat map of similarity under different metrics, cosine, Euclidean distance; arbitrary range heat map of similarity under different metrics(duplicates, similar, etc); Histgram of different similarity metrics; without and with extreme value filters. All based on test of sample size 1000 and different variations of original codes.

Computation efficiency

To test algorithms covered in this report and code or different similarity search models(especially when expanding to more complexed ML models), code/program design[32] is crucial to save time and resources.

The colab free tier or home workstation machines can offer limited availability and computes, hence a better designed code, or even different algorithm structure could potentially improve the general computation performances.

Considering choices of detail algorithm sections, although shingle wise calculation of similarity could be computationally expensive and not suitable for huge data scale

[31] Since most of the visualization option requires pandas data frame which takes high volume of memory, sampling could be a necessary approximation technique.

[32] The current code has not exhausted options to substitute all UDF to built in functions.

situation, it is worth noting that, such similarity analysis could be useful to analyze small scale or sampled data.

Use case

The clearer use case description could lead the research to more focused target(example, distinguish duplication, high similarity, low similarity[33], or finding semantic meaning dispersion using other ML models).

[33] Example, if the jaccard similarity is close to 1, it is potential duplication, from different job posters. So potential duplication, high similarity, low similarity, could offer more insights to the target datasets.

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.