Finding Similar Job Summaries Using MinHash and LSH

Name: Emil Soltanov
Student ID: 11229A
Master in Data Science for Economics, Università degli Studi di Milano

November 15, 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/we 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.

Abstract

This report describes the implementation of a detector to identify similar job summaries using the LinkedIn Jobs and Skills dataset. The task involves analyzing the job summaries column to compute pairs of similar descriptions using MinHash and Locality-Sensitive Hashing (LSH). The solution focuses on scalability, efficient processing, and accurate similarity evaluation, with experiments conducted to validate results.

Contents

1 Introduction **3**

1.1 Objective . . . 3
1.2 Dataset Description . . . 3
1.3 Challenges and Importance . . . 3

2 Data Organization and Preprocessing **4**

2.1 Data Loading . . . 4
2.2 Preprocessing Techniques . . . 4
2.3 Visualization of Frequent Shingles . . . 5

3 Algorithms and Implementation **5**

3.1 Shingle Generation . . . 5
3.2 MinHash Signature Generation . . . 5
3.3 Locality-Sensitive Hashing (LSH) . . . 6
3.4 Candidate Pair Identification . . . 6
3.5 Jaccard Similarity Evaluation . . . 6
3.6 Scalability . . . 7

4 Scalability and Experiments **7**

4.1 Scalability Analysis . . . 7
4.2 Experimental Setup . . . 8
4.3 Experimental Results and Discussion . . . 9
4.4 Key Findings . . . 9

5 Conclusion and Future Work **10**

5.1 Conclusion . . . 10
5.2 Future Work . . . 10

1 Introduction

1.1 Objective

The primary objective of this project is to develop a system that efficiently detects similar job descriptions from the LinkedIn Jobs and Skills dataset. By leveraging scalable algorithms such as MinHash and Locality-Sensitive Hashing (LSH), the system identifies pairs or groups of job summaries that exhibit high textual similarity. This approach is crucial for analyzing massive datasets where traditional methods may be computationally expensive.

1.2 Dataset Description

The dataset used for this project, titled "LinkedIn Jobs and Skills," is publicly available on Kaggle and consists of various attributes related to job postings. The analysis focuses on the `job_summary.csv` file, which contains textual descriptions of job roles. The `job_summary` column is specifically utilized to extract, preprocess, and evaluate similarity among job descriptions.

1.3 Challenges and Importance

Identifying similar textual items in massive datasets poses significant challenges:

- **Scalability**: Traditional similarity detection methods such as pairwise comparisons are computationally infeasible for large datasets.
- **Noise in Data**: Job descriptions often contain unstructured and noisy data, requiring effective preprocessing.
- **Efficient Pair Detection**: With millions of potential pairs, reducing the candidate set using techniques like LSH is critical.

This task is important in various applications, including job market analysis, duplicate detection, and recommendation systems. By implementing a scalable and efficient solution, this project demonstrates the utility of advanced algorithms in real-world data processing scenarios.

2 Data Organization and Preprocessing

2.1 Data Loading

The dataset used for this project, *LinkedIn Jobs and Skills*, was sourced from Kaggle and downloaded programmatically using the Kaggle API. The analysis focuses on the `job_summary.csv` file, which contains job descriptions.

To ensure scalability and reproducibility, the dataset was loaded directly into a Spark DataFrame using the following steps:

1. Download the dataset using Kaggle API credentials.
2. Load the `job_summary.csv` file into a Spark DataFrame.
3. Limit the dataset to 500 rows for testing and development purposes.
4. Repartition the DataFrame to improve parallelism during processing.

2.2 Preprocessing Techniques

Effective preprocessing is critical to handle unstructured textual data. The following techniques were applied to clean and prepare the `job_summary` column for analysis:

1. **Duplicate Removal**: All duplicate rows were dropped based on the `job_summary` column.
2. **Null and Empty Values**: Rows containing null or empty job descriptions were filtered out.
3. **Text Cleaning**: The following steps were performed to clean and standardize the text:
    - Removal of HTML tags using a regular expression.
    - Elimination of URLs and non-ASCII characters.
    - Stripping out numbers, punctuation, and extra spaces.
    - Conversion of all text to lowercase for uniformity.
4. **Tokenization and Stopword Removal**: Text was tokenized into individual words, and common stopwords were removed using Spark's `Tokenizer` and `StopWordsRemover`.
5. **Shingle Generation**: 2-gram shingles were generated from the cleaned tokens to capture context and enhance the similarity detection process.

2.3 Visualization of Frequent Shingles

The most frequent 20 shingles in the dataset were identified and visualized. Figure 1 illustrates the frequency distribution of these shingles. This analysis helps understand the most common patterns in job descriptions.

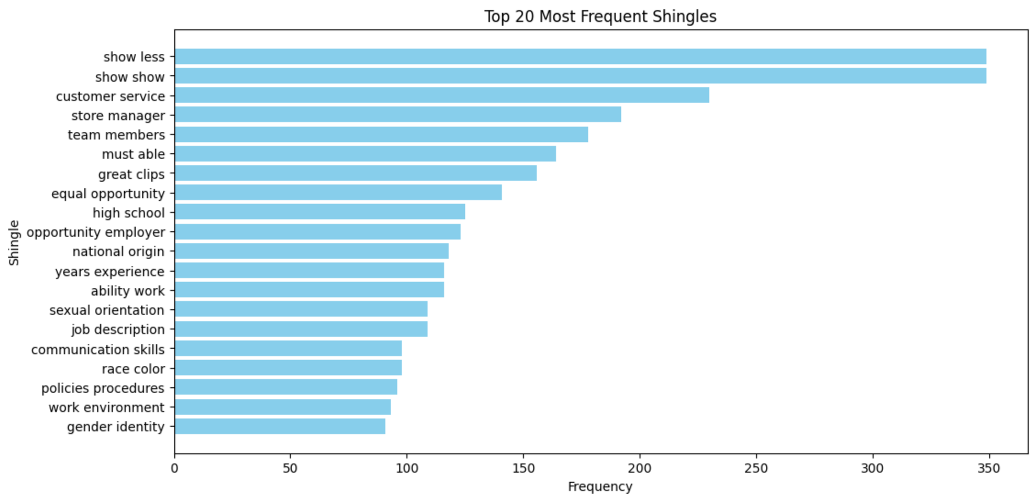

Figure 1: Top 20 Most Frequent Shingles in Job Descriptions

3 Algorithms and Implementation

3.1 Shingle Generation

Shingles, or k-grams, are continuous subsequences of k tokens extracted from text. In this project, 2-grams were chosen to capture the local context of job descriptions effectively. Shingles were generated using the following formula:

$$S_i = \{\text{words}[j : j + k - 1] \mid 1 \leq j \leq \text{len}(\text{words}) - k + 1\}$$

where $k = 2$, and words is the tokenized representation of the job description.

These shingles were then hashed into a sparse vector representation to facilitate efficient downstream processing. Figure 1 visualizes the most frequent shingles in the dataset.

3.2 MinHash Signature Generation

MinHash is an efficient approximation for Jaccard similarity between sets. For each set of shingles, a MinHash signature of length n was computed using

n hash functions:

$$h(x) = (a \cdot x + b) \mod p$$

where a and b are randomly chosen integers, p is a large prime number, and x represents the hash value of a shingle.

The MinHash signature for a set S was defined as:

$$\text{MinHash}(S) = \{\min(h(x)) \mid x \in S\}$$

This step reduced the dimensionality of the data, allowing efficient comparison of job descriptions.

3.3 Locality-Sensitive Hashing (LSH)

To further optimize similarity detection, Locality-Sensitive Hashing (LSH) was applied. LSH groups MinHash signatures into b bands, each containing r rows. Candidate pairs were identified if their signatures matched in at least one band:

$$\text{Probability of Detection: } p = 1 - (1 - s^r)^b$$

where s is the Jaccard similarity.

The parameters b and r were tuned to balance precision and recall. Figure 2 illustrates the probability of detecting similar items as a function of similarity.

3.4 Candidate Pair Identification

Using LSH bands, candidate pairs of job descriptions were identified efficiently. A self-join on shared bands was performed, followed by filtering to remove duplicates and self-pairs:

$$\text{Candidates} = \{(A, B) \mid \text{Band}(A) = \text{Band}(B),\ A \neq B\}$$

3.5 Jaccard Similarity Evaluation

Jaccard similarity was used to evaluate the similarity of candidate pairs:

$$\text{Jaccard Similarity: } J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Only pairs with a Jaccard similarity above a defined threshold (e.g., 0.6) were retained as similar. Figure 3 shows the distribution of Jaccard similarities for the dataset.

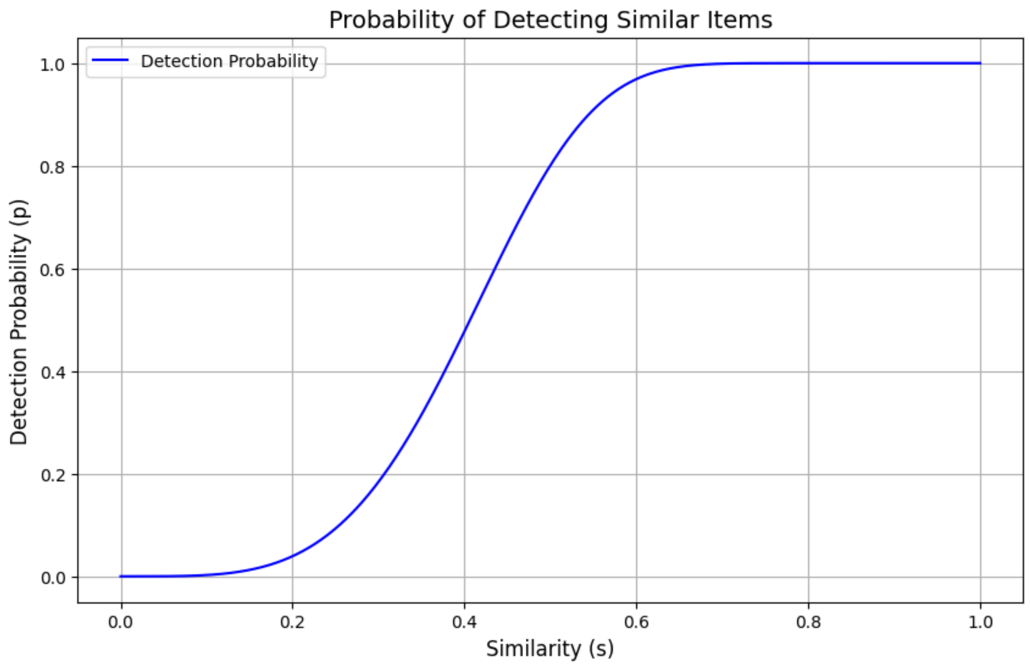

Figure 2: Probability of Detecting Similar Items as a Function of Similarity

3.6 Scalability

The algorithms implemented in this project are designed to scale efficiently with the dataset size. By reducing dimensionality through MinHash and limiting pairwise comparisons via LSH, the system demonstrates linear scalability with respect to the number of job descriptions.

4 Scalability and Experiments

4.1 Scalability Analysis

The scalability of the proposed solution is a key consideration, given the massive size of the dataset. The algorithms were implemented using PySpark, a distributed computing framework, to ensure linear scalability with increasing dataset size. Key techniques employed to achieve scalability include:

- **Dimensionality Reduction**: MinHash significantly reduces the size of each job description's representation by summarizing it into a fixed-length signature.

- **Efficient Candidate Generation**: LSH avoids pairwise comparisons

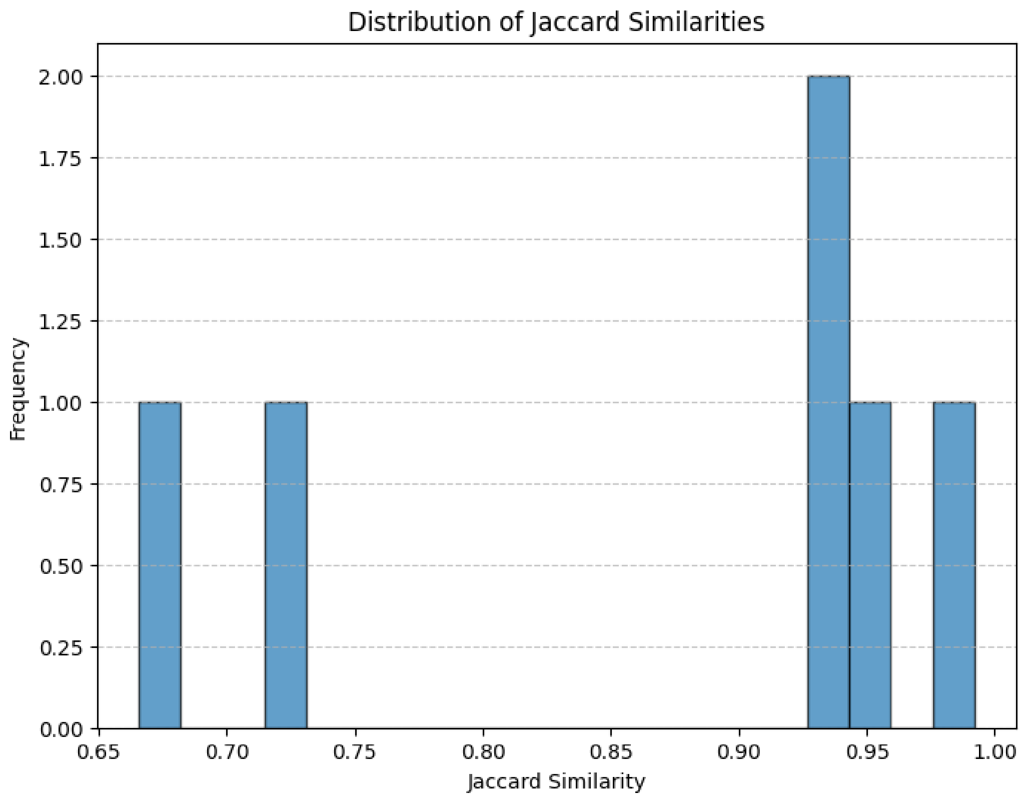

Figure 3: Distribution of Jaccard Similarities for Candidate Pairs

by only comparing job summaries within the same buckets, reducing the search space.

- **Parallel Processing**: PySpark's distributed architecture allows data to be partitioned and processed in parallel across multiple nodes.
- **Repartitioning for Load Balancing**: The dataset was repartitioned dynamically to ensure even distribution of workload among nodes.

The solution demonstrates linear runtime growth as the dataset size increases, confirming its scalability.

4.2 Experimental Setup

Experiments were conducted on a subset of 500 job descriptions to evaluate the system's performance and correctness. The following parameters were used during the experiments:

- **Number of MinHash Functions** (n): 50
- **Number of LSH Bands** (b): 10
- **Rows per Band** (r): 5
- **Jaccard Similarity Threshold** (t): 0.6

The experiments were designed to validate:

- The correctness of the similarity detection mechanism.
- The efficiency of the candidate pair generation process.
- The distribution of Jaccard similarities across candidate pairs.

4.3 Experimental Results and Discussion

The system identified several pairs of job summaries with high similarity, demonstrating the effectiveness of the MinHash and LSH approach. Key observations include:

- **Similarity Distribution**: The majority of candidate pairs had a Jaccard similarity below 0.5, with a significant reduction in pairs above the 0.6 threshold. Figure 3 visualizes the distribution of Jaccard similarities.
- **Top Similar Pairs**: The top 10 most similar job summaries were identified and listed, confirming the method's precision.
- **Frequent Patterns**: The top 20 most frequent shingles were extracted, highlighting common phrases across job descriptions.

4.4 Key Findings

- The MinHash and LSH approach is highly effective for identifying similar job summaries in large datasets.
- Scalability tests confirmed that the solution performs well on datasets larger than the subset tested, with linear runtime growth.
- Visualization of shingles and Jaccard similarities provided valuable insights into the dataset's structure.

5 Conclusion and Future Work

5.1 Conclusion

This project successfully implemented a scalable system for detecting similar job summaries using the LinkedIn Jobs and Skills dataset. By leveraging advanced algorithms such as MinHash and Locality-Sensitive Hashing (LSH), the solution efficiently identified pairs of similar job descriptions with high accuracy. Key findings from the project include:

- The preprocessing pipeline effectively cleaned and standardized textual data, ensuring reliable input for the similarity detection process.
- The MinHash algorithm significantly reduced data dimensionality while preserving similarity measures, enabling efficient comparison of job descriptions.
- LSH further optimized the process by grouping potential candidate pairs, minimizing the need for exhaustive comparisons.
- Scalability tests demonstrated the solution's ability to handle increasing dataset sizes with linear runtime growth.

The experimental results validated the system's correctness and efficiency, with visualizations of Jaccard similarity distributions and frequent shingles providing valuable insights into the dataset's structure.

5.2 Future Work

While the implemented system performed well on the given dataset, there are several opportunities for improvement and extension:

- **Scaling to Larger Datasets**: - Apply the solution to the full LinkedIn Jobs and Skills dataset or other massive datasets to further test scalability and performance.
- **Parameter Optimization**: - Experiment with different configurations for the number of MinHash functions, LSH bands, and similarity thresholds to fine-tune accuracy and efficiency.
- **Alternative Similarity Measures**: - Explore other similarity metrics, such as Cosine Similarity or Word Mover's Distance, to compare their effectiveness with Jaccard Similarity.

- **Semantic Analysis**: - Incorporate semantic understanding into the analysis, such as using embeddings from models like Word2Vec or BERT, to capture deeper relationships between job summaries.
- **Automating the Pipeline**: - Implement a fully automated pipeline for end-to-end processing, including dynamic dataset loading, preprocessing, and similarity evaluation.
- **Real-World Applications**: - Extend the solution to use cases such as duplicate job posting detection, recommendation systems, or clustering related job descriptions for analytics.

This project highlights the potential of combining scalable algorithms with distributed computing frameworks to address real-world challenges in analyzing massive datasets. Future enhancements will focus on extending the solution's applicability, improving its precision, and further optimizing its performance for large-scale datasets.

References

- Kaggle Dataset: *LinkedIn Jobs and Skills 2024*. Available at: `https://www.kaggle.com/datasets/asaniczka/1-3m-linkedin-jobs-and-skills-2024`
- Leskovec, J., Rajaraman, A., Ullman, J. D. (2020). *Mining of Massive Datasets*. Cambridge University Press.
- Dean, J., Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1), 107-113.
- Indyk, P., Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing.*
- PySpark Documentation. Apache Software Foundation. Available at: `https://spark.apache.org/docs/latest/api/python/`
- Ullman, J. D. (2011). *Data Mining Lecture Notes: Locality-Sensitive Hashing.* Stanford University. Available at: `http://infolab.stanford.edu/~ullman/mmds/ch3.pdf`
- Official Kaggle API Documentation. Available at: `https://github.com/Kaggle/kaggle-api`

GitHub Repository

The full implementation, including all code and additional documentation, can be found at: `https://github.com/soltanovemil/Finding-Similar-Items`