FINDING SIMILAR ITEMS

Algorithms for Massive Data project

Tejaswini Yadav Nakka (06651A) Sai Spandana Adivishnu (03180A)

April 2024

Abstract

In today's paced world navigating the online job market can be, like trying to find a needle in a haystack due to the sheer volume of opportunities inundating job search platforms. Our project addresses this issue directly by offering a solution that simplifies the process of finding job listings. We delved into the array of job postings within the LinkedIn Jobs & Skills dataset, utilizing sophisticated data processing techniques such as shingling, MinHashing and Local Sensitive Hashing all facilitated by PySpark's capacity to effortlessly manage large datasets. This approach does not facilitate job seekers in identifying positions that align with their skills and career goals efficiently. Also assists recruiters in pinpointing candidates whose backgrounds match the roles they aim to fill. Through our project we aim to streamline the journey towards connections transforming the quest for an ideal job or candidate from a daunting endeavour, into a targeted and effective pursuit.

Contents

1 Introduction

The current job market is filled with in numerous job opportunities. Especially the online job search apps/websites are where the recruiters and job seekers are helping each other. Even though, these job search sites open a gateway to opportunities. They also make one's life harder by pouring all the opportunities at once where the candidates have to read the job description and handpick the most relevant job. So, we propose a solution for detecting similar job description by using advanced data processing techniques. Focusing on the job summary column in the job_summary.csv file downloaded from the ≪LinkedIn Jobs & Skills≫ dataset, published on Kaggle, and released under the ODC-By license, Version 1.0. The primary objective is to identify pairs (or sets of higher cardinalities) of similar job descriptions providing valuable insights for both recruiters and job seekers.

The initial step of our approach involves preprocessing the text data extracted from job postings, where we use various techniques such as data processing, shingling, MinHash signature generation and finally the Local sensitive hashing with the help of jaccard similarity. These techniques help us to capture the data in a format which is suitable for comparison, which makes it easier to analyse the similarities between different job summaries and organize them in a way that makes it quick and easy to find the ones that are similar to each other.

This project helps the recruiters by simplifying the process to searching candidates who apply for similar job descriptions as theirs. And helps the job seekers to search the job posting with relevant job opportunities that matches their skills and qualifications or the type of job they're looking for.

2 The dataset and data preparation

To find the similar items in large models, we imported the LinkedIn dataset from the Kaggle created by asaniczka (2024). Our main aim is to find the similar items sets from the job_summary.csv file. The dataset consists of 5.1 GB of LinkedIn job summary along with the job links. The text to feed the algorithms are in the job summary column from the desired csv file and we are using words as items.

Each row having the job description/summary collected by the author. We aim to find the recurrent combination of words inside each sentence in the summary, in order to find the similar job description. Text data from the diverse sources like LinkedIn can be messy, unstructured and highly varied. Common issues include inconsistent formatting, spelling errors, presence of special characters. The goal of preprocessing is to standardize this text data to a format that's ideal for the analysis, particularly for the similarity detection. We are using the pyspark for the algorithms we used in the analysis. The main idea behind using the pyspark is, it is designed to handle big data that doesn't fit into the memory of a single machine. It processes data in a distributed manner across multiple nodes.

After checking and making sure that there are no null values. We have moved forward with preprocessing the text by performing Tokenization, lowercasing, and stripping the punctuation. Initially, the text is split into individual words or tokens. We used the 'split()' method to split the text based on whitespace, which creates a list of tokens. Later, each token is converted to lowercase using 'lower()' method. This ensures that the words Job and job are treated as the same word. Punctuation marks like commas, periods, exclamation marks, etc., are removed from the tokens. By using a technique called list comprehension, a simple way of writing a loop which goes through each word and cleaning it. The punctuation marks are removed at the end or beginning of the word.

We added a column named "preprocessed_text" by creating a custom function that tokenizes,cleans and Transforms each job description from the "job_summary" column into a suitable format for analysis. With the implementation of a custom text processing function called Preprocess_text using Pyspark to sanitize and tokenize the data we defined it as a user defined function (UDF) named preprocess_text_udf. This UDF allows us to apply the function to every entry in the "column'. By taking this approach we can effectively. Format text data within a Spark DataFrame preparing it for analysis or machine learning tasks. This method offers an scalable solution, for incorporating text preprocessing techniques into our data processing workflows.

3 Finding Similar items:Theoretical Framework

After the data preprocessing, we extract the feature from the raw data. From the textbook Mining of Massive Datasets, written by A.Rajaraman and J.Ullman, To obtain the features we implement algorithms like Local sensitive hashing (LSH), that allows us to focus on pairs that likely to be similar, without having to look at all pairs. There are three essential steps for finding the similarity in documents that are Shingling, Min-Hashing and finally Locality-Sensitive Hashing. We considered the example of Identifying similar news articles as a reference from the textbook.

3.1 Shingling

Shingling is a technique used in text processing to break down documents into sets of overlapping tokens or sequences, commonly referred as shingles. For instance, slicing up a document into consecutive word or character sequences to capture its essence. These slices or shingles help in comparing documents for similarity, especially in tasks like duplicate detection or near- duplicate detection.

3.1.1 K-Shingles

A K-shingles (or K-gram) is a sequence of k tokens extracted from a document, where token can be a character, word, or other units of text, depending on the specific application. You must indicate the k large enough that the probability of any given shingles appearing in any given document is low. Generally, we consider K=5 for short documents and k=10 for long documents. For instance: If k=3 and the text is "TEXT MINING", if we consider 3- shingles as a character based then the text would be ["tex"," ext", "xt ", "t m", " mi", "min", "ini", "nin", and "ing"], Or if k=2 if we consider the k as word based for the text "text mining is fun", the 2-shingles would be ["text mining", "mining is", "is fun"]. K-shingles are primarily used to assess the similarity between documents, By comparing the sets of shingles form two documents, one can estimate how similar the documents are. This particularly useful in applications like plagiarism detection. The next step involves MinHashing

3.2 MinHashing

MinHasing is a technique designed to efficiently estimate the jaccard similarity between sets, making it an invaluable tool in handing large -scale data where direct computation of similarities is computationally prohibitive.

3.2.1 Jaccard Similarity

Jaccard similarity is a measure used to quantify the similarity between two sets. It's defined as the size of the intersection divided by the size of the union of the two sets.

$$Jaccard\_Similarity(S,T) = \frac{|S \cap T|}{|S \cup T|} \tag{1}$$

Where S and T are sets. The jaccard similarity ranges from 0 to 1, where 0 means no similarity(ie., the sets do not share elements) and 1 means that the sets are identical.

MinHashing provides a way to estimate the Jaccard similarity without needing to directly compute the intersection and union sizes of the sets, which can be expensive for large datasets. It does this by converting sets into compact signatures while approximately preserving their similarity.

It involves using a collection of hash functions. Each function can hash the elements of a set into a large number of buckets, with the key property being that each hash function randomly permutes the elements. For a given set, its MinHash signature is created by applying each hash function to every element in the set and recording the minimum hash value obtained for each function. The collection of these minimum values across all used hash functions forms the MinHash signature of the set. The similarity between two sets can then be estimated by comparing their MinHash signatures. Specifically, the estimate of the Jaccard similarity is the fraction of hash functions for which the two sets have the same minimum hash value. If two sets are similar, their MinHash signatures will have a higher proportion of hash values in common.

The fascinating of MinHash lies in its foundation in probability. The probability that a specific hash function will produce the same minimum value for two sets is equal to the Jaccard similarity of those sets. Thus, by taking the average agreement across multiple hash functions, we get a good estimate of the true Jaccard similarity.

After generating MinHash signatures for sets of data- a step that effectively condenses the original information into a more manageable condenses the original information into a more manageable form while preserving the essence of similarity is Local Sensitive hashing.

3.3 Local Sensitive Hashing

The primary goal of LSH is to maximize the probability that similar items (based on Jaccard similarity or another similarity metric) hash items several times in such a way that similar items are more likely to be hashed to the same bucket, while dissimilar items hash to different buckets. This approach significantly reduces the computational burden by limiting similarity comparisons to items within the same bucket rather than across the entire dataset.

One common strategy in LSH, especially when dealing with MinHash signatures, is to divide each signature into smaller segments or "bands." The idea is that if two signatures are similar, they are likely to be identical, or at least very close, in at least one of these bands. Each band of the MinHash signature is hashed using a hash function that maps the band to a bucket in a hash table. Items whose bands hash to the same bucket are considered "candidate pairs" for being similar. By only comparing items within the same buckets, LSH drastically narrows down the pool of comparisons needed to identify similar items. This step is where the efficiency gain comes from, as it circumvents the need for pairwise comparisons across the entire dataset.
The banding and hashing process in LSH is designed to ensure that items with high Jaccard similarity (as approximated by their MinHash signatures) end up in the same bucket. The parameters of LSH (e.g., the number of bands and the size of each band) can be tuned based on the desired similarity threshold, balancing the trade-off between false positives and false negatives.

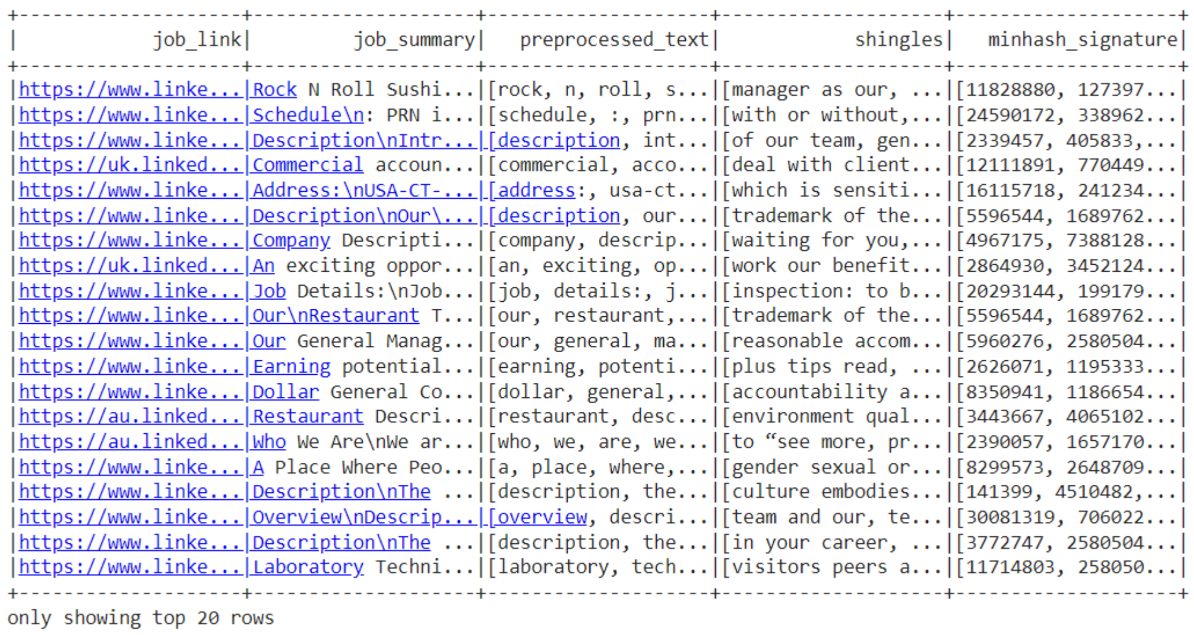
4 Algorithm implementation and Results

In our project, as we discussed in the previous chapter, we created a function (generate_shingles) that takes a piece of text as input and break it into shingles of a specified length, where we selected 3 as the specific length. But this can be changed depending on the analysis. Next the function slides a window of the specific length (in our case, k=3) over the words, creating a shingle for each window by concatenating the tokens. These shingles are stored to ensure uniqueness, as redundancy is not useful for the analysis. After completion, the function gives set of unique shingles, which can be further used for comparison.

We defined a User Defined Function (UDF) named generate_shingles_udf which takes text as input, joins the tokens into a single string and then applies the generate_shingles function to create shingles. Shingles are essentially small, overlapping subsequence's of words extracted from the text. By using this UDF, we generate shingles for each entry in the "preprocessed_text" column of our Data Frame.

After shingling we created a MinHash signatures for a given set of shingles. After that we specifies the number of permutations to use for generating the MinHash signatures in our case we opt 128. More permutations generally provide a more accurate approximation of Jaccard similarity but at the cost of increased computational overhead. we indicated the parameter 'pd' to control the output format. If pd=1, then function returns the MinHash object itself. If pd=0, it returns the hash values as a list. The shingles are encoded and updated into the MinHash object, which then either returns this object or its hash values as list, based on the 'pd' parameter.

After this step we created User- Defined Function (UDF) that allowing it to be applied to the dataframe column containing sets of singles. This UDF returns an array of integer hash values, constituting the MinHash signature. The UDF is registered to work with Spark's dataframe API, enabling the processing of each row's singles in parallel across a spark cluster.



```
+--------------------+--------------------+--------------------+--------------------+--------------------+
|            job_link|         job_summary|   preprocessed_text|            shingles|   minhash_signature|
+--------------------+--------------------+--------------------+--------------------+--------------------+
|https://www.linke...|Rock N Roll Sushi...|[rock, n, roll, s...|[manager as our, ...|[11828880, 127397...|
|https://www.linke...|Schedule\n: PRN i...|[schedule, :, prn...|[with or without,...|[24590172, 338962...|
|https://www.linke...|Description\nIntr...|[description, int...|[of our team, gen...|[2339457, 405833,...|
|https://uk.linked...|Commercial accoun...|[commercial, acco...|[deal with client...|[12111891, 770449...|
|https://www.linke...|Address:\nUSA-CT-...|[address:, usa-ct...|[which is sensiti...|[16115718, 241234...|
|https://www.linke...|Description\nOur\...|[description, our...|[trademark of the...|[5596544, 1689762...|
|https://www.linke...|Company Descripti...|[company, descrip...|[waiting for you,...|[4967175, 7388128...|
|https://uk.linked...|An exciting oppor...|[an, exciting, op...|[work our benefit...|[2864930, 3452124...|
|https://www.linke...|Job Details:\nJob...|[job, details:, j...|[inspection: to b...|[20293144, 199179...|
|https://www.linke...|Our\nRestaurant T...|[our, restaurant,...|[trademark of the...|[5596544, 1689762...|
|https://www.linke...|Our General Manag...|[our, general, ma...|[reasonable accom...|[5960276, 2580504...|
|https://www.linke...|Earning potential...|[earning, potenti...|[plus tips read, ...|[2626071, 1195333...|
|https://www.linke...|Dollar General Co...|[dollar, general,...|[accountability a...|[8350941, 1186654...|
|https://au.linked...|Restaurant Descri...|[restaurant, desc...|[environment qual...|[3443667, 4065102...|
|https://au.linked...|Who We Are\nWe ar...|[who, we, are, we...|[to “see more, pr...|[2390057, 1657170...|
|https://www.linke...|A Place Where Peo...|[a, place, where,...|[gender sexual or...|[8299573, 2648709...|
|https://www.linke...|Description\nThe ...|[description, the...|[culture embodies...|[141399, 4510482,...|
|https://www.linke...|Overview\nDescrip...|[overview, descri...|[team and our, te...|[30081319, 706022...|
|https://www.linke...|Description\nThe ...|[description, the...|[in your career, ...|[3772747, 2580504...|
|https://www.linke...|Laboratory Techni...|[laboratory, tech...|[visitors peers a...|[11714803, 258050...|
+--------------------+--------------------+--------------------+--------------------+--------------------+
only showing top 20 rows
```

Figure 1: Creating Minhash Signatures based on shingles

Each 'minhash_signature' in the output represents a compact summary of the job description's content, specifically its shingles. By comparing these signatures, one can efficiently estimate the similarity between job descriptions without directly comparing the original text.

Using the Create_lsh_index function, we set up a MinHash Lsh index, for efficiently searching similarities among job description in our dataset. We initiated with an empty LSH index and set the parameters for Jaccard similarity and the number of permutations for calculating Minhash. Then this function iterates through each entry in the dataset and extracts job summary text. Then with the help of functions like generate_shingles and generate_minhash_signature which we have defined before, the lsh index function creates minhash signatures for each job summary. These signatures, along with their corresponding row indices, are carefully stored in a dictionary called minhashes, serving as a reference for future analysis. In the same time, MinHash signatures are incorporated into the LSH index, forming a framework ready for quick similarity queries. By doing so, this function efficiently helps in identifying similar job descriptions in the dataset.

We defined another function called find_similar_jobs which is used to identify similar job descriptions using LSH. This helps in iterating through each job description in the dataset, generating shingles and their corresponding MinHash signatures. These signatures are then used to query the LSH index to retrieve candidate pairs that potentially have similar descriptions. For every candidate pair, the jaccard similarity coefficient is calculated between their minhash signatures. If the jaccard coefficient exceeds a predefined threshold (0.5) indicating certain level of similarity, the pair is added to the list of similar pairs. Finally, this function helps finding the overlapping content helping both recruiters and job seekers.

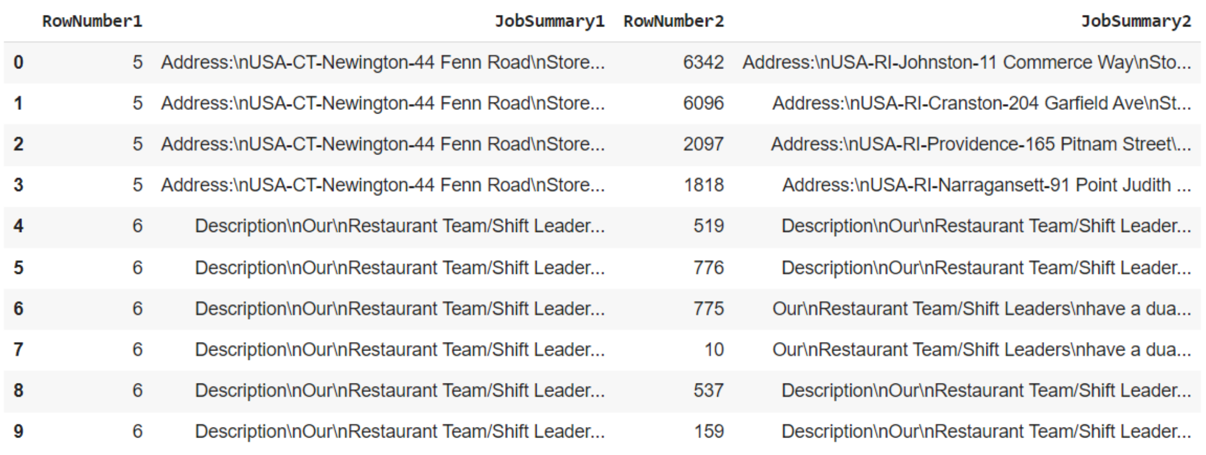
Finally, we succeeded in processing a large dataset stored in a CSV file in chunks, with each chunk containing a specified number of rows (10,000 in this case). It initializes an LSH index and an empty dictionary to store MinHash signatures. It iterates over each chunk, creating MinHash signatures for each row's job summary and inserting them into the LSH index. Then, it searches for similar job descriptions within the chunk using the LSH index and MinHash signatures.

If similar pairs are found, they are added to a DataFrame (JobSummary 2) to keep track of them. Finally, the code prints the DataFrame containing the similar job pairs found in the chunk.

In our case the table shows pairs of job summaries, each identified by 'RowNumber1' and 'RowNumber2', which link to specific job descriptions (JobSummary1 and JobSummary2). This structured format is indicative of a comparison or analysis between job postings, possibly to identify similarities or differences in job roles, responsibilities, or locations.

4.1 Observations

The first few rows (0-3) all share the same RowNumber1 (5) but are compared with different RowNumber2s, indicating that the job summary from RowNumber1 is being compared against multiple other job summaries and found similar job summaries in where the row number 2 are 6342,6096,2097,1818. Rows 4-9, where RowNumber1 is 6, involve job summaries that seem related to restaurant team or shift leaders. This could indicate an effort to match similar job roles across a particular category of jobs.



	RowNumber1	JobSummary1	RowNumber2	JobSummary2
0	5	Address:\nUSA-CT-Newington-44 Fenn Road\nStore...	6342	Address:\nUSA-RI-Johnston-11 Commerce Way\nSto...
1	5	Address:\nUSA-CT-Newington-44 Fenn Road\nStore...	6096	Address:\nUSA-RI-Cranston-204 Garfield Ave\nSt...
2	5	Address:\nUSA-CT-Newington-44 Fenn Road\nStore...	2097	Address:\nUSA-RI-Providence-165 Pitnam Street\...
3	5	Address:\nUSA-CT-Newington-44 Fenn Road\nStore...	1818	Address:\nUSA-RI-Narragansett-91 Point Judith ...
4	6	Description\nOur\nRestaurant Team/Shift Leader...	519	Description\nOur\nRestaurant Team/Shift Leader...
5	6	Description\nOur\nRestaurant Team/Shift Leader...	776	Description\nOur\nRestaurant Team/Shift Leader...
6	6	Description\nOur\nRestaurant Team/Shift Leader...	775	Our\nRestaurant Team/Shift Leaders\nhave a dua...
7	6	Description\nOur\nRestaurant Team/Shift Leader...	10	Our\nRestaurant Team/Shift Leaders\nhave a dua...
8	6	Description\nOur\nRestaurant Team/Shift Leader...	537	Description\nOur\nRestaurant Team/Shift Leader...
9	6	Description\nOur\nRestaurant Team/Shift Leader...	159	Description\nOur\nRestaurant Team/Shift Leader...

Figure 2: Similar Job Summaries

5 Conclusion

This project employs a smart strategy to handle large datasets effectively by breaking them into smaller, manageable chunks. This not only optimizes performance but also ensures scalability, allowing it to handle even massive datasets easily. By combining Locality Sensitive Hashing (LSH) and MinHash signatures, the code quickly identifies similar job descriptions with great accuracy. This innovative approach transforms the way we approach job matching and talent acquisition, offering recruiters and job seekers a powerful tool to uncover hidden connections and make better decisions. Simply put, this code marks a groundbreaking leap forward in data-driven recruitment. Simplifying the process to searching candidates who apply for similar job descriptions as theirs. And helps the job seekers to search the job posting with relevant job opportunities that matches their skills and qualifications or the type of job they're looking for.

6 APPENDIX

6.1 GitHub Link

You can find python files on given GitHub link
`https://github.com/SpandanaTejaswini/Finding-Similar-Items-.git`

7 References

- Mining of Massive Datasets, written by A. Rajaraman and J. Ullman
- DataSet Link: `https://www.kaggle.com/datasets/asaniczka/1-3m-linkedin-jobs-and-skills-2024.`
- `https://spark.apache.org/docs/latest/api/python/getting_started/install.html`