SI 671 Project Report
Alexander He
alexhe@umich.edu

Scraping and Clustering of Data-Centric Job Descriptions on Glassdoor

Introduction:

In the past decade, there has been an unprecedented surge in the demand for data-centric roles, reflecting the tech industry's gravitation towards data-driven decision-making. As the industry continually evolves, it's crucial to understand how these changes are reflected in job listings. This study explores the following: What patterns and insights can be derived from clustering and analyzing job listings using different text representations? How do these findings illuminate the framing of role requirements, skill demands, and other job aspects? Additionally, how do the patterns and trends in the current job listings compare with past snapshots of listings, and what implications might these differences have for job seekers and employers?

Business Problem Definition:

Navigating the tech job market is complex for job seekers and employers alike. Job seekers, especially those new to the industry or considering a career pivot, need insights into current trends and expectations in job listings. Understanding the evolving requirements, salary expectations, and role nuances can guide them in aligning their skills and applications effectively.

Employers and HR departments face the challenge of attracting top talent in a competitive landscape. Navigating these complexities requires a deep, data-driven understanding of the current job market. One approach is to analyze current job listings and compare them with past data, which helps to understand how the market has shifted. Such analyses ensure that job descriptions are clear, resonate with the desired candidates, and align with industry trends.

Job posting websites such as Glassdoor are designed to have users query for similar job postings by keywords, location, and other filters. However, relying on Glassdoor's built-in search functionality is insufficient to create a data-driven understanding of job postings, especially without access to Glassdoor's internal database or company recruitment data. Thus, web scraping is required to collect data at scale for individual analysis purposes.

Related Work:

A study by Debao et al. aimed to understand the characteristics of big data jobs through K-means text clustering. They analyzed job recruitment data, categorizing big data roles into 10 categories and examining aspects like urban distribution, salary, and educational requirements.[1] This research highlights the utility of clustering techniques in dissecting job market trends, similar to your approach, though my study uses Agglomerative Hierarchical Clustering.

A user on R-bloggers.com conducted an analysis on over 6,500 job descriptions for data roles across multiple European countries. The study aimed to understand the skillsets and technologies sought by employers for roles like data analysts, data scientists, engineers, and machine learning specialists. They found clear clusters of skillsets for different data-related roles.[2] This approach aligns closely with my project's goal to extract and analyze key attributes from job descriptions.

Both studies, while using different methodologies (K-means and text analysis alone), share a common objective with my study – analyzing job descriptions to extract meaningful patterns and insights. The difference lies in the methods

[1] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0255419
[2] https://www.r-bloggers.com/2022/04/text-analysis-of-job-descriptions-for-data-scientists-data-engineers-machine-learning-engineers-and-data-analysts/

for clustering. I aimed to strike a balance between detail retention and dimensionality reduction through various text representations.

Data Source & Description:

The data consists of job postings in the U.S. extracted with 4 queries on Glassdoor: Data Scientist, Data Analyst, Data Engineer, and Business Analyst. The earlier dataset (picklesueat/data_jobs_data) was created by a different author, which comprises a snapshot of 13612 job listings scraped from Glassdoor on 7/8/2020. The author's scraper was outdated since the website format had changed and was no longer compatible, so I re-made the scraper and fetched recent listings on Glassdoor for the same 4 queries, extracting a total of 22056 listings. For both datasets, the final size was achieved after the removal of duplicates between query results. The data columns are: Job Title, Salary Estimate, Job Description, Rating, Company Name, Location, Headquarters, Size, Founded, Type of ownership, Industry, Sector, Revenue, Competitors, Easy Apply.

Web Scraping Methodology

As mentioned previously, I re-made the scraper and fetched recent listings for the same 4 queries. The scraper was programmed to navigate the dynamic environment of Glassdoor's website, efficiently handling pagination and dynamic content to ensure stable data collection. The fields I retrieved closely following the older dataset's fields. The methodology adopted for web scraping was pivotal in gathering a robust dataset, forming the foundation for a comprehensive analysis of job market trends and patterns.

Glassdoor has a limit for how many postings it will load for a single query, which is 900. To ensure that all relevant postings in the United States are extracted, scraping was applied to queries with locations specified at the state and city levels, the latter being useful for states with over 900 postings alone, which were usually California and Texas.

Methods:

The analysis concentrated on evaluating the impact of various text representations on the agglomerative clustering of job descriptions. A comprehensive evaluation of clustering performance between text representations is achieved, controlling for dimensionality to ensure a consistent comparison. Each cluster's defining attributes were extracted, including the top keywords in job descriptions, the top job titles, companies, industries, sectors, and locations.

The same methods were also applied to a similar dataset from 2020 without any modifications. This demonstrated the ability for comparative analyses between past and present snapshots of the job market. This comparison provides a valuable historical perspective, allowing for an examination of how the tech job market and its defining characteristics have evolved over time. By juxtaposing the clustering results and key attributes from both datasets, a comprehensive view of the trends and shifts in the tech industry's job market can be achieved. Unfortunately, due to time constraints, I did not perform an actual comparative analysis using the system I created.

Zipf Plot Analysis

Zipf plot analysis was conducted on job descriptions to explore and refine their linguistic patterns. These visualizations were key in performing a comparative analysis of different n-gram configurations and stopword treatments. I evaluated the impact of lemmatization and stopword removal on the distribution word/N-gram frequencies. These frequencies were extracting using the CountVectorizer, encompassing unigrams, bigrams, and trigrams. Zipf plots were then created on a log-log scale, offering a visual representation of these frequency distributions. The objective was to strike a balance between preserving the natural linguistic structure inherent in the

job descriptions. This approach ensured that the essential characteristics job descriptions were maintained, facilitating a more accurate and meaningful subsequent
analysis.

Text Representations

To prepare the data for transformation into the different text representations, I aimed to preserve as much of the inherent structure as possible with a focus on real words. I did not remove stopwords but removed tokens containing numeric characters. Finally, I applied lemmatization so that keyword matching is simplified. The data was then transformed into the following text representations:

1. **TF-IDF with LSA**: TF-IDF (Term Frequency-Inverse Document Frequency) measures the importance of a word to a document in a collection or corpus, adjusted for the fact that some words appear more frequently in general.[3] I tested both unigrams and a combination of unigrams and bigrams. In both cases, dimensionality reduction was performed on the document-term matrix using Latent Semantic Analysis (LSA), also known as Truncated Singular Value Decomposition (SVD) in the Sci-Kit Learn API.[4] I extracted the top 400 components of the latent semantic space to balance detail retention and computational efficiency. For unigrams, this captured ~39% of the explained variance. For unigrams combined with bigrams, it captured ~20%.
2. **Word Embeddings**: Gensim's Word2Vec model[5] was trained on the preprocessed text, exploring window sizes of 2, 5, and 10 to assess the influence of local versus broader contextual information. I tested both the Continuous Bag-of-Words (CBOW) and Skip-Gram variants[6] , but the results were similar, so I just used CBOW, which was the default. The resulting embeddings were fixed at 400 dimensions, matching the dimensionality of all the representations I tested.
3. **TF-IDF Weighted Word Embeddings**: In this hybrid approach, Word2Vec embeddings were weighted by the corresponding average TF-IDF scores of each word, aiming to merge the semantic depth of embeddings with the importance weighting of TF-IDF. Once again, I tested window sizes of 2, 5, and 10 and fixed embeddings at 400 dimensions.

Agglomerative Clustering

I applied L2 normalization for each representation to ensure comparability, then applied Agglomerative Hierarchical Clustering with Ward linkage. I also tested single, average, and complete linkage, but the results were poor, so I committed to using Ward linkage only. I varied the number of clusters from 2 to 30 clusters.

I used two key metrics to evaluate the clustering quality: the Silhouette score and the Davies-Bouldin score. The Silhouette score, calculated as the mean Silhouette Coefficient of all samples, measures how well each object lies within its cluster. A high Silhouette score indicates well-separated and distinct clusters, whereas a score near 0 suggests overlapping clusters and negative values indicate potential misplacements in clustering.[7] On the other hand, the Davies-Bouldin score is defined as the average similarity measure of each cluster with its most similar cluster. This similarity is the ratio of within-cluster distances to between-cluster distances.[8] Lower Davies-Bouldin scores are desirable, as they indicate clusters that are less dispersed and more distinct from each other. This score, therefore

[3] https://www.cambridge.org/core/books/abs/mining-of-massive-datasets/data-mining/E5BFF4C1DD5A1FB946D616D619B373C2
[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
[5] https://radimrehurek.com/gensim/models/word2vec.html
[6] https://arxiv.org/pdf/1301.3781.pdf
[7] https://www.sciencedirect.com/science/article/pii/0377042787901257
[8] https://ieeexplore.ieee.org/document/4766909

provides a measure of the average 'compactness' and separation of the clusters, with lower values indicating better clustering.

Additionally, I utilized dendrograms to visualize the agglomerative process of identifying distinct clusters. To ensure a consistent comparison between text representations without being too granular, I focused on extracting 10 clusters across all text representation variants for denogram visualization and keyword extraction per cluster.

Keyword Extraction
After clustering the data, I retrieved the top keywords ranked by Word Frequency and TF-IDF score in the job descriptions of each cluster. I tested three methods: extracting all top tokens and n-grams (from unigrams to 5-grams) per cluster, extracting the same with post-clustering stopword removal, and using regex matching to select the top tokens and n-grams containing keywords from a manually-constructed set of lemmatized keywords that I expect to be relevant.

For other attributes such as job titles and companies, I extracted all top job titles and the top companies by their whole name (though job titles also underwent cleaning). This was done by creating a custom tokenizer function that simply returned the whole string without splitting.

Results

Zipf Plot Analysis
In my analysis of job descriptions using Zipf plots, I observed significant variations in the distribution of n-grams based on using different stopword lists. There are 179, 318, 378, and 119 stopwords in NLTK, Sci-Kit Learn, their union, and their intersection sets, respectively. I tested these four variants along with no stopword removal.

For unigrams, stopword removal led to a slight reduction in the number of tokens detected by the vectorizer. The counts decreased marginally from 52,449 (no stopwords) to 52,137–52,336 when stopwords were removed (**Fig. 1**). This minor change suggests that unigrams are less affected by stopword removal.

However, bigrams and trigrams saw the opposite effect. The removal of stopwords led to a notable increase in the count of unique bigrams, increasing the total from 1,376,965 without removal to between 1,611,575 and 1,657,483. The count for unique trigrams also followed this trend, rising from 5,756,088 without removal to between 5,914,353 and 6,112,661. **Fig. 1** shows the results from no stopword removal and removing the smallest stopwords set (intersection between NLTK and Sci-Kit Learn's default stopwords) for each n-gram range. Notably, removing fewer stopwords created a larger increase in artificial bigrams and trigrams than removing more stopwords. This suggests that removing stopwords disrupts common word combinations. Specifically, as we increase the number of removed stopwords, a peak in additional n-grams occurs before decreasing again. These observations underscore a critical aspect of text preprocessing. While removing stopwords can be beneficial in reducing noise and focusing on meaningful content, it can also significantly alter the natural linguistic structure of the text. This is especially true in non-unigram n-grams, where word interplay is crucial. Considering the effects on dimensionality and the creation of artificial bigrams/trigrams, I decided not to remove any stopwords before transformations into different text representations.

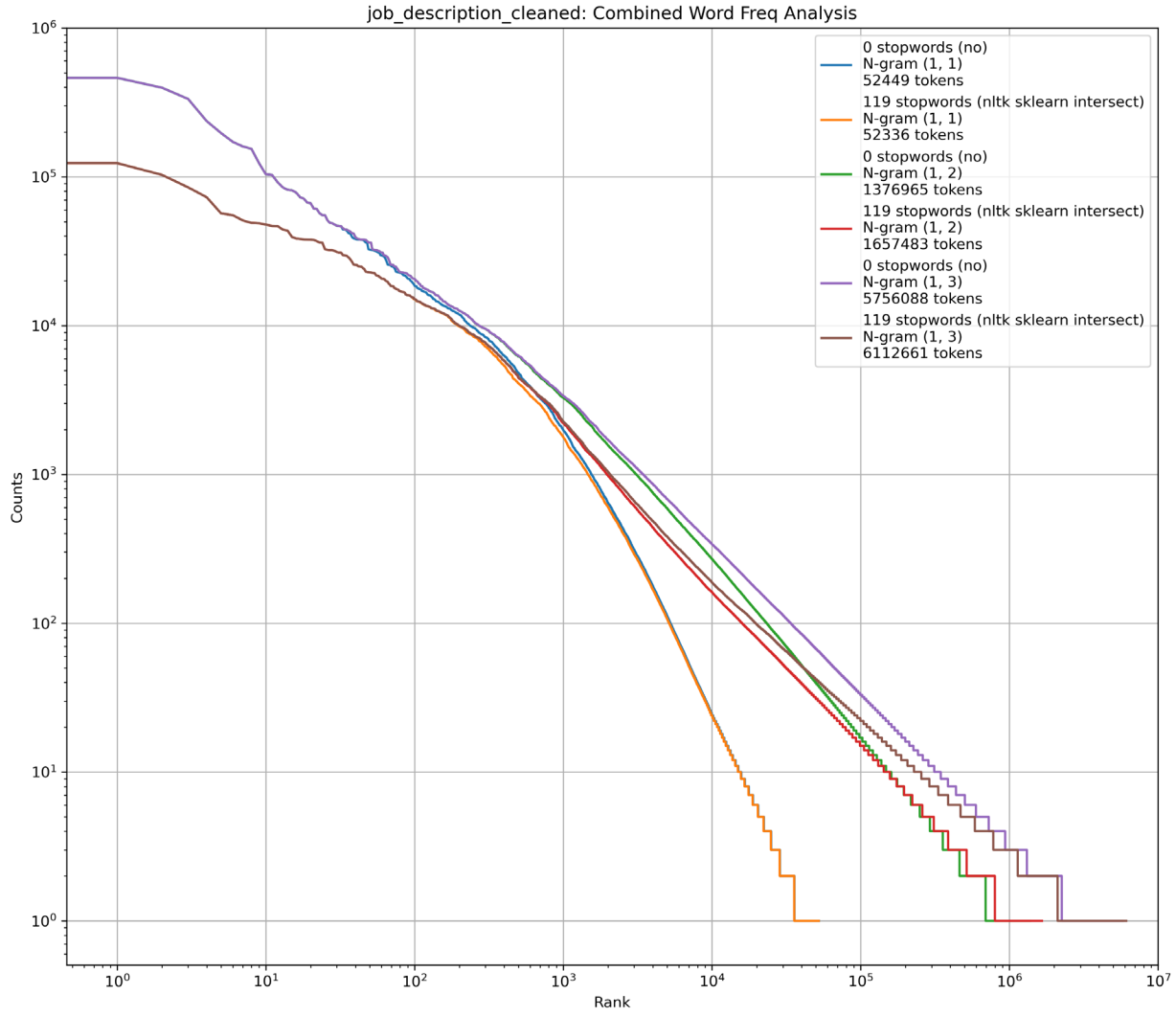


Figure 1

Comparing the different n-gram configurations, I found that the plot for unigrams alone exhibited a less linear trend, indicating a departure from the natural distribution typically observed in language data. Conversely, the plots that included both unigrams and bigrams (n-gram range 1-2), as well as those with unigrams, bigrams, and trigrams (n-gram range 1-3), showed a more linear distribution (**Fig. 1**). The plots for the 2020 data showed a very similar pattern besides a lower number of tokens due to fewer datapoints, so I will not show it here. Finally, given the marginal difference between the unigram-bigram and unigram-bigram-trigram distributions, I opted to proceed with the unigram-bigram variant to balance linguistic representation with computational efficiency,

Clustering Analysis of Different Data Representations

The effectiveness of each text representation method in clustering was distinctly observed through our evaluation metrics. Higher average Silhouette scores and lower average Davies-Bouldin scores indicate higher-quality clusters, as described previously. For my newly scraped data, the effect of the number of clusters on these metrics for all text representations is shown in **Fig. 2**. For the 2020 data, the results are shown in **Fig. 3**. The relative patterns were quite

similar, though some magnitudes are significantly different, which can be attributed to not only differences in job descriptions but also an effect from the different dataset sizes.

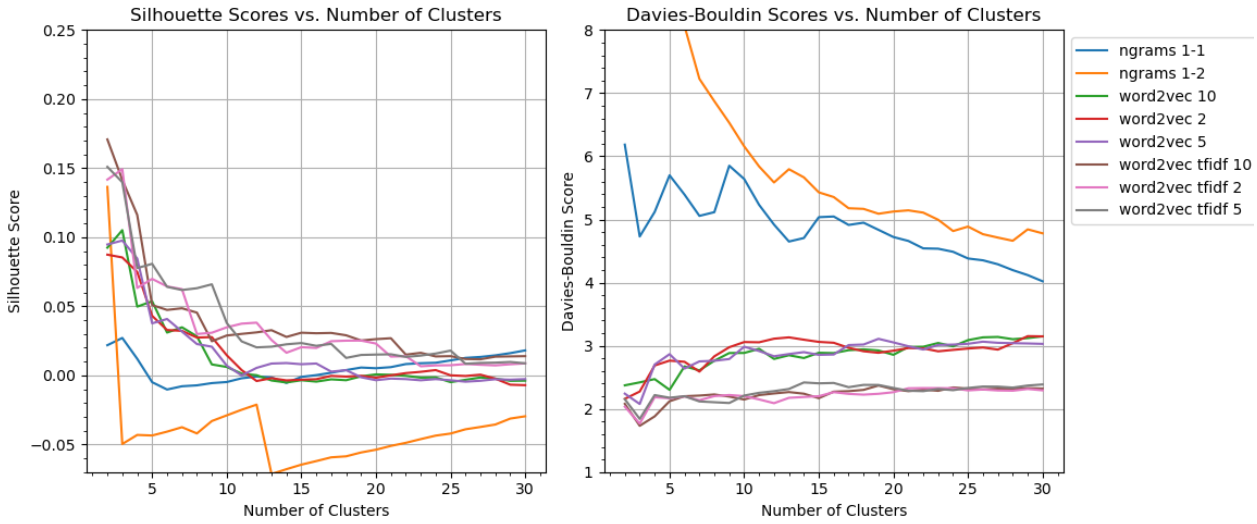


Figure 2: Clustering scores on the new dataset

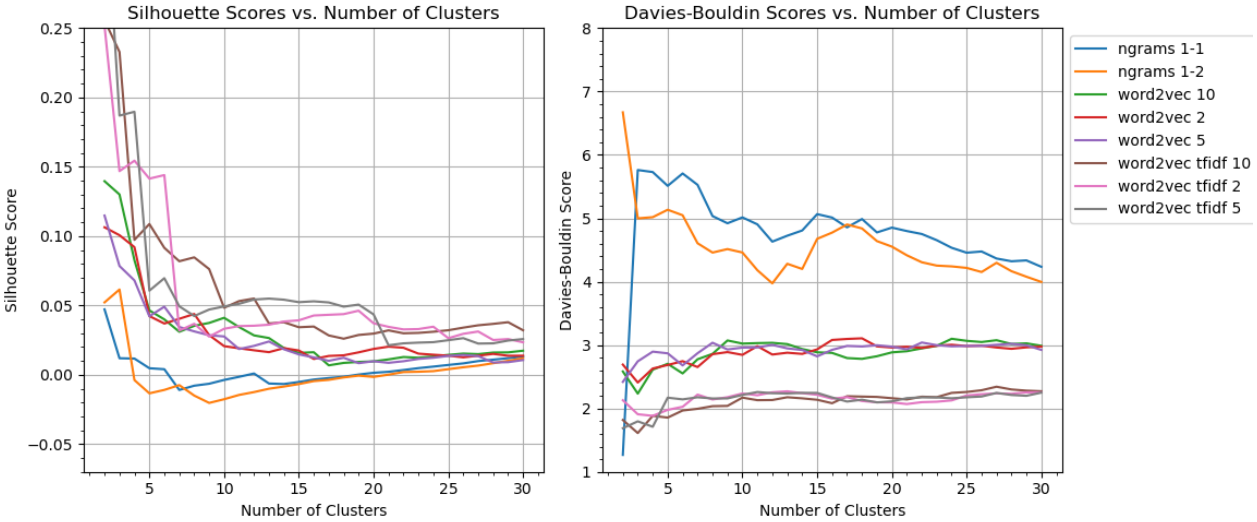


Figure 3: Clustering scores on the 2020 dataset

1. **TF-IDF with LSA**: The clustering results from unigram and unigram-bigram TF- IDF representations indicated moderate cluster definition. However, these were outperformed by other methods, as reflected in their lower (and often negative) Silhouette scores and higher Davies-Bouldin scores. While TF-IDF is adept at identifying key terms, it might fail to capture the broader semantic context inherent in job descriptions.
2. **Word Embeddings**: Clustering using Word2Vec embeddings alone demonstrated an improvement over the TF-IDF approach. The semantic coherence of clusters was enhanced, as shown by the noticeable impact on the clustering metrics, especially in the Davies-Bouldin score. The choice of window size (2, 5, and 10) slightly altered the clustering outcomes, suggesting that clusters are significantly stable between local and broader contextual information encoding.

3. **TF-IDF Weighted Word Embeddings**: This method showcased the highest silhouette scores and the lowest Davies-Bouldin scores. These results underscore the efficacy of combining a certain degree of semantic richness in embeddings with the specificity of TF-IDF, resulting in more defined and semantically meaningful clusters.

This analysis revealed that the TF-IDF Weighted Word Embeddings approach emerged as the most proficient method for clustering job descriptions. It better captured the nuanced semantic relationships within the text while emphasizing critical terms, leading to somewhat more distinct and interpretable clusters.

To highlight differences in clustering between text representations, I provided the denograms of the resulting clusters for each text representation, using unigram-bigrams for TF-IDF with LSA (Fig. 4), a window size of 5 for both raw (Fig. 5) and TF-IDF weighted (Fig. 6) word embeddings, and 10 clusters for a direct comparison without too much granularity. TF-IDF with LSA created the most imbalanced cluster sizes, ranging from 131 to 7526. Raw word embeddings had clusters with the most balanced sizes, from 1052 to 5078. TF-IDF weighted word embeddings had clusters closer in size to TF-IDF with LSA, though slightly more balanced, from 200 to 7132. Importantly, the descendant clustering splits (indicated by the dark points in the trees) are far less balanced in TF-IDF with LSA than in TF-IDF weighted word embeddings. This suggests that TF-IDF with LSA places too much importance on keyword differences, but adding TF-IDF weights to word embeddings addresses this issue by encoding both keyword differences and semantic information.

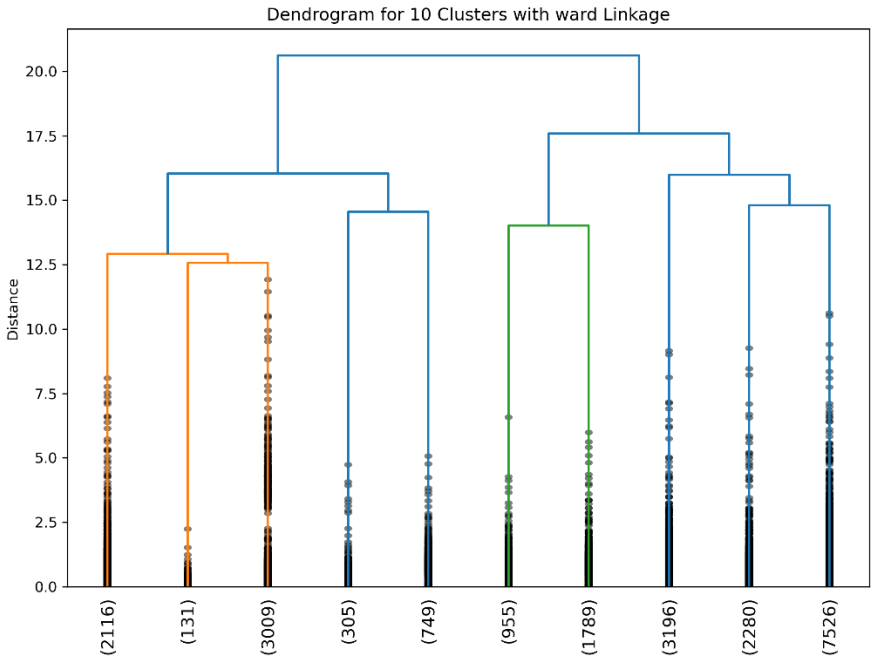


Figure 4: TF-IDF with LSA on unigram-bigrams on the new dataset

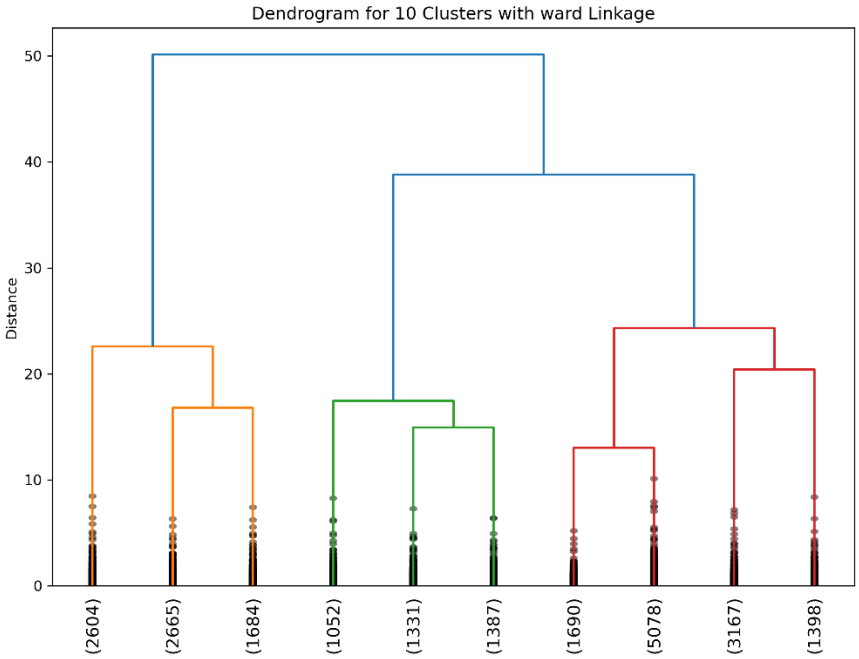


Figure 5: Raw Word Embeddings with window size 5 on the new dataset

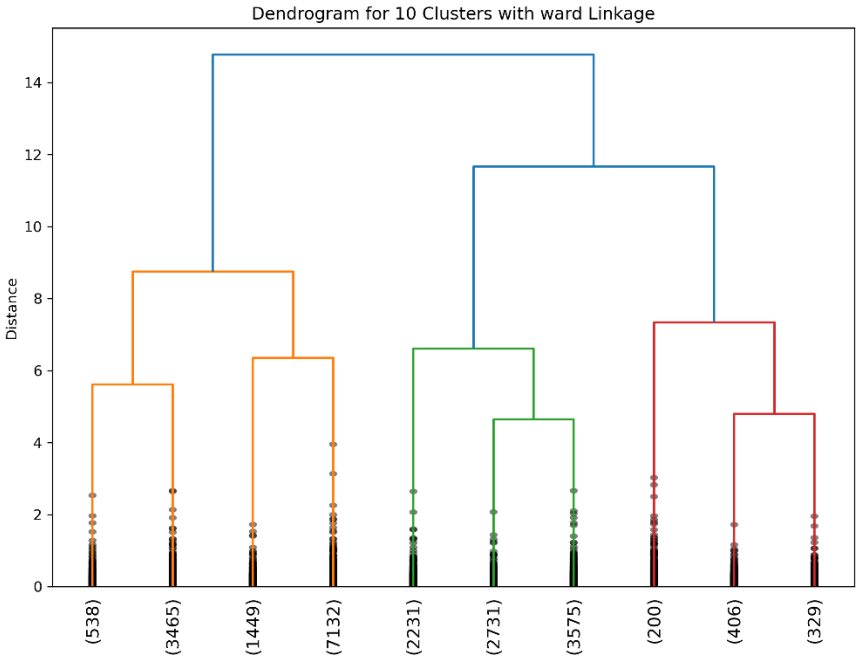
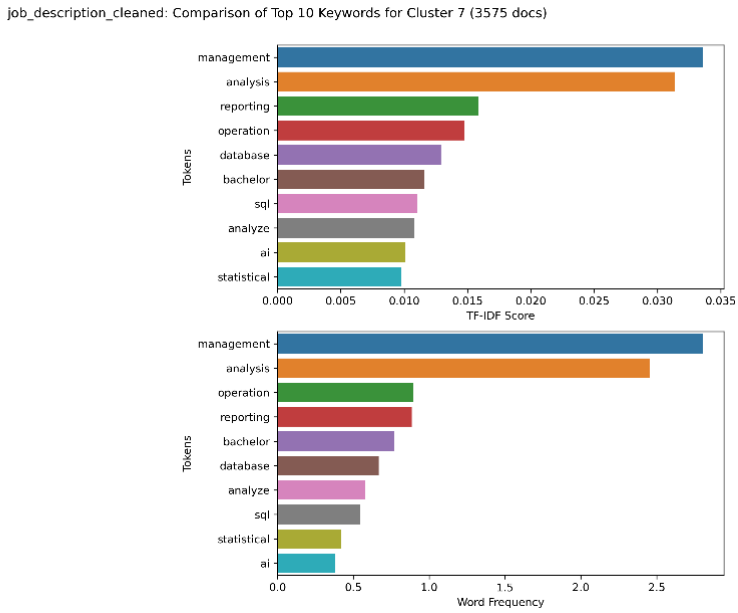
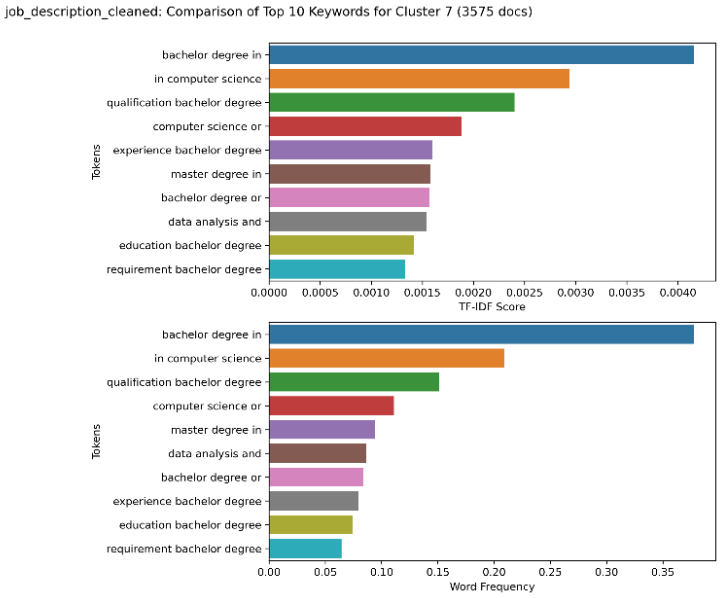


Figure 6: TF-IDF Weighted Word Embeddings with window size 5 on the new dataset

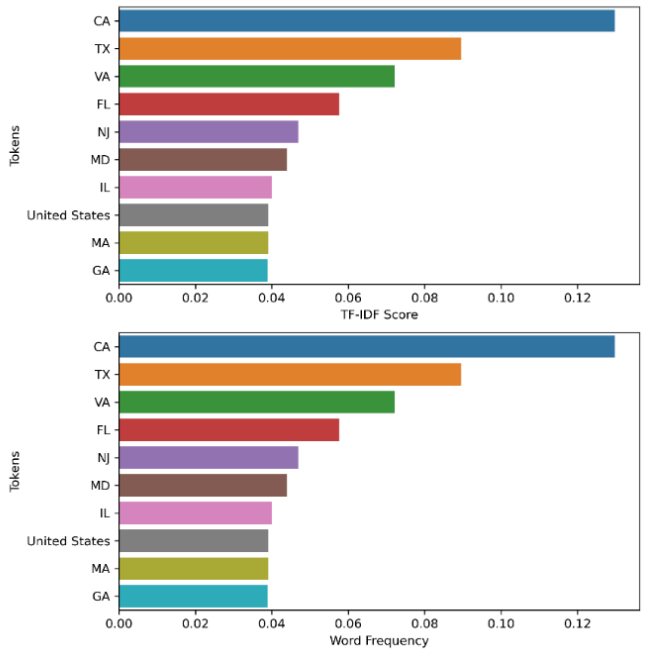
Keyword Extraction

The following are example results from keyword and attribute extraction using the best representation (TF-IDF Weighted Word Embeddings with window size 5). Again, I did not perform self-directed comparative analysis, so these are just for demonstration purposes.

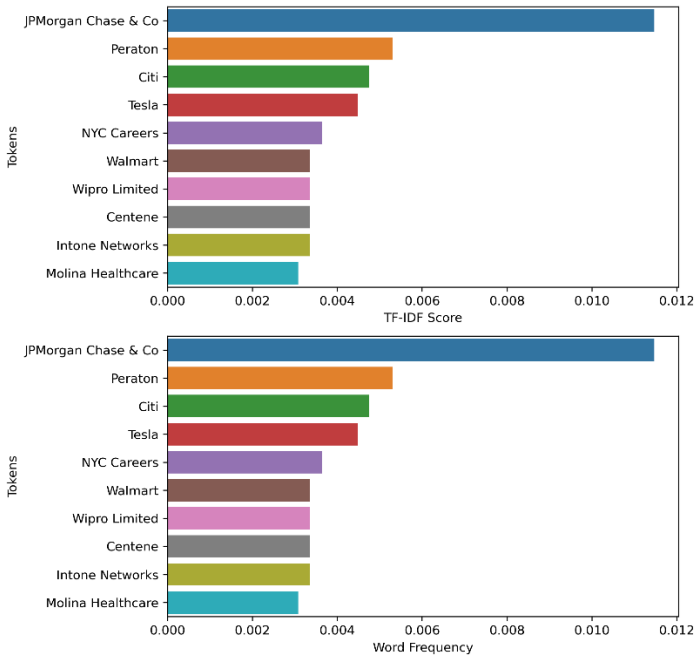


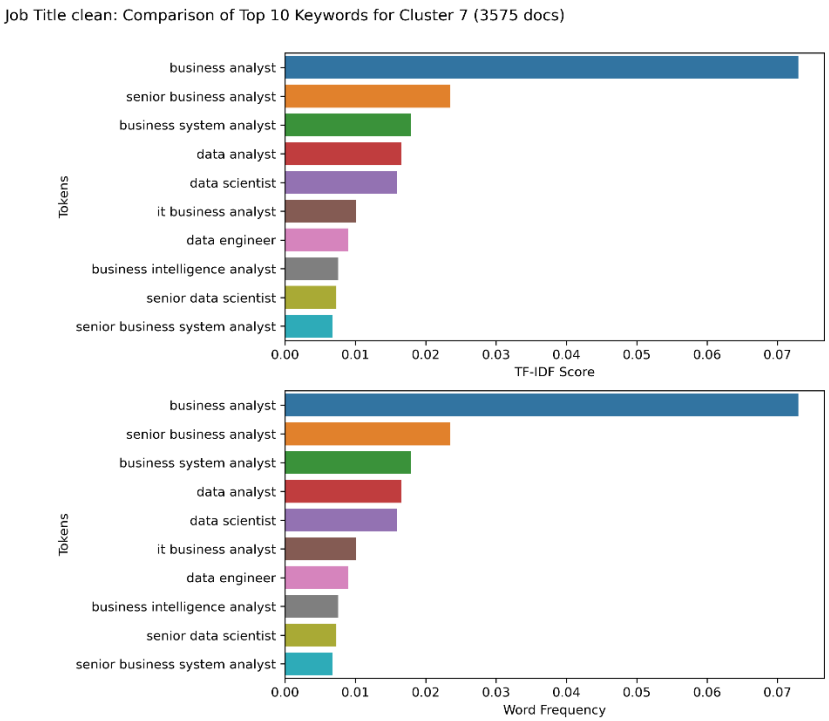


State: Comparison of Top 10 Keywords for Cluster 7 (3575 docs)



Company Name: Comparison of Top 10 Keywords for Cluster 7 (3575 docs)





Limitations and Future Work

A major limitation of my study is its ability to handle a larger variety of job descriptions. If we only cared about generalizability, we can simply extract all unique tokens in each cluster accompanied by their word frequency and TF-IDF values. However, the results are hard to use, and we would have to manually extract keywords from the output anyways, which requires domain knowledge of the positions of interest. To address this issue, I selected the top tokens and n-grams (from unigrams to 5-grams) using lemmatized regex matching on a manually constructed set of keywords that I expect to be relevant. I consider this an optimization created by my domain knowledge of job descriptions of data-centric positions. This isn't great for generalizability, but a better alternative may not exist. Furthermore, there is a high computational cost whenever non-unigrams are involved unless clever optimizations are developed. Computational costs can be lowered through stopword removal and other cleaning steps, but due to the inconsistent structure of job descriptions, this is only effective to a certain degree before information loss becomes an issue and non-unigrams become overly unnatural. Manual extraction of the "requirements" or "qualifications" section of each job description is certainly possible, but many man-hours would be required, akin to a large data annotation task.

One thing I overlooked was that I did not tune the Word2Vec models. I cannot say whether they are underfitting or overfitting, so hyperparameter adjustments and additional epochs could change clustering results. Addionally, other clustering methods can be tested, such as soft clustering with GMMs or C-means.

Multi-word buzzphrases don't always match between different job descriptions, and extensive regex matching can only do so much. To address these issues, we can implement advanced information retrieval techniques for clustering, such as learn-to-rank features, but they do come with their own set of challenges.

Another limitation is the subjective analysis required from users who wish to leverage the tool. This may require the development of more advanced features, though I believe my system is a great starting point.

Any kind of traditional time series analysis is currently not possible due to the lack of timestamps in the older dataset. Without access to Glassdoor's proprietary data and knowledge of when companies take down postings, we can employ semi-live scraping over some period of time, where exact posting dates can be inferred from the "days ago" field found in the DOM.

Code Base

https://drive.google.com/drive/folders/174BHYN0DjyduEiFzB56VkcqssS9HdoRK?usp=sharing