Cassiopee Project - Technological Change and Firm Organization

Lucien LY et Istiak KHAN
Supervised by Anahid Bauer

July 2, 2024

1 Introduction

The Cassiopee Project n°31 Technological Change and Firm Organization explores the impact of technological change, in particular artificial intelligence (AI) and natural language processing models such as GPT, on the organisation of firms. Supervised by Anahid Bauer, the project aims to understand how firms are reorganising their work structures in response to the growing adoption of these innovations.

Since the launch of ChatGPT in 2022, AI and GPT technologies have raised concerns about their potential to replace tasks performed by a variety of workers. However, studies suggest that these technologies have yet to have a significant impact on the total number of workers in establishments.

In this context, we are going to examine how companies are reorganising their work and what skills are in demand. This project proposes to investigate this reorganisation through an analysis of job vacancies, providing indications of the skills and occupations in demand and showing how the labour market is currently evolving.

2 Presentation

The aim of this project is to assess the exposure of institutions and companies to AI by analysing the content of job advertisements collected from Burning Glass Technologies.

To this end, the project will focus on classifying the skills and occupations mentioned in these adverts, in order to differentiate companies with a high use of AI and GPT from those with a low use.

More specifically, the aim is to develop text analysis programs in Python language to classify skills and occupations according to their likelihood of being replaced by these emerging technologies.

3 Scraping

We started by scraping data from the site layoffs.io, a job-tracking tool that tracks redundancies in tech-related companies. The site contains a table containing Google Sheets links with the LinkedIn profile links of the people made redundant.

#	List Name	Date Added	# People	Link	Locations	Layoff Date
1	7Shifts laid-off employees	2/2/2024	31	https://docs.google.com/spreadsheets/d/1Fc41RiVNaveHpTSCHo4x7kwq2Qtaxsrms2QVfS_854/edit#gid=0		1/1/2024
2	Flexport laid-off employees	31/1/2024	41	https://docs.google.com/spreadsheets/d/1d6ejecq8qy46SJv_VJk2TMbzT_S_77EGHPHjYMcJruw/edit?resourcekey#gid=550780554		26/1/2024
3	Block laid-off employees	31/1/2024	24	https://docs.google.com/spreadsheets/d/1z87eN8hKqflNawqSEswcOTkrPTBVHvYaSUlwmR9uCoc/edit#gid=705603816		30/1/2024
4	HubSpot laid-off employees	25/1/2024	19	https://docs.google.com/spreadsheets/d/1AzNRk1EflTBUtFl1MLqle9br27Oc0U5s7ioOhTqeKbM/edit#gid=0	Remote	24/1/2024

Figure 1: Website layoffs.io.

We then applied a Python program that stores all the LinkedIn links in a list associated with each company.

Figure 2: List containing every LinkedIn links grouped by companies.

Finally, we split each list to have just one link per line to make it easier for us to do the next task of retrieving each person's skills from their LinkedIn profile.

Now that we have all the links to the LinkedIn profiles of the people who have been made redundant, we need to find a way of scrapping their skills. To do this, we used a Google Chrome extension called Magical.

This allows us to label the 100 skills that a person can put on their profile and then retrieve them. Once this has been done, we can create an automation that will transfer the data to a Google Sheet file. The aim is to open several LinkedIn links, load them to the bottom of the page and then use the extension.

Given the number of profiles to be scanned which is around 4000, we also need to automate the opening of links. We automatically open around twenty links at a time from a webdriver and automate the click to the extension. Unfortunately, LinkedIn has a fairly strict policy on web scraping, which is why you need to regulate the number of connections to the site.

The use of a VPN is therefore necessary to avoid being banned every time (although this was the case at the beginning) but it slows down the process considerably and can also disconnect the session, making the automation work obsolete. We still managed to get a result, although it's not finished yet.

4 Burning Glass Technologies Data

The data supplied to us by **Burning Glass Technologies** (BGT) is a gigantic collection of job advertisements collected throughout the United States. BGT is a labor market analytics firm that uses artificial intelligence technologies to collect and host a massive repository of workforce and employment data.
They gathers data from approximately 50,000 online sources, including job boards and corporate websites, to create a comprehensive repository of job advertisements, each containing over 70 variables. The dataset includes several interconnected tables: Main (job descriptors), Certifications, CIP (Classification of Instructional Program codes), Degree, Major, and Skills, all of which are de-duplicated for accuracy. Data profiling assesses the dataset's quality through metrics such as completeness, value validity, consistency, uniqueness, and duplication.
The data was segmented into zipped CSV files of 200 MB each, all formatted in the same way.

The difficulties encountered were numerous. Firstly, the size of the data. There was a very large volume of data that was impossible to store locally. To get round this first problem, we used the Box platform, which enabled us to have the data available and download it when we needed to, or else load it into a Python environment. Even if we used a technique that allowed us to store temporarily a CSV file, it would take too much time.

Although, in order to facilitate the task and save some time, especially in the execution of our codes, we opted to work on solely one file since we knew that whatever results were yielded through this would be applicable to the entirety

of the data.

Then we had to compare the two versions, because there were two different versions of the data. To achieve this, a number of tests were carried out on the different characteristics of the data. The process was very simple: just convert the data into a DataFrame and then use different logical operations to compare the data, either column by column, row by row or cell by cell - testing out the equalities to find out the differences between the two DataFrames, perform subset analysis to compare only the rows or columns that are common between the two DataFrames...

The first operations did show a difference, but the distinctions weren't very clear. But if you compare the **last_updated_date** columns, you can see that version 1 dates back to January 2024, while version 2 dates back to March 2024.

Based on this indicator, the choice of database was clear, especially since the aim of our project is to study the impact of the emergence of a new technology on the labour market.

Our DataFrame is composed of 126,359 postings, 132 columns, from 7,200 different cities in America, with 6,600 companies

Our study focused on companies in the technology industry. And despite the fact that there is a column named NAICS, for North American Industry Classification System, the database was not constructed in such a way as to be able to properly filter this particular industry. In fact, the database is sometimes incomplete, leaving many boxes blank with the words 'Unclassified'. Indeed, especially in the columns : Company Name, naics contain Unclassified mentions. This left us with the difficult task of trying to filter the companies of the Technological Industry.

There were various options available to us to try and remedy this problem. The first option was to use the SpaCy library, which is an open-source natural language processing (NLP) library in Python. It is designed to be fast, efficient and easy to integrate into production applications. SpaCy is widely used for a variety of NLP tasks thanks to its ability to handle large volumes of text quickly and accurately.

More specifically, we are interested in using SpaCy for keyword matching, enabling us to try and identify keywords in job descriptions. So we asked ChatGPT to give us a list of 100 keywords that would enable us to reflect the industry landscape. The list included various fields such as Big Data, Cybersecurity, DevOps, UX/UI Designer, Wireless Networks, Python, etc., giving us a fairly exhaustive view of the world of the technology industry. The aim was then to create a Python code that would browse the job descriptions and associate a certain score based on matching with the list of keywords. Then, a minimum threshold was defined to keep the job offers coming from companies in the technology industry. After this, we had our database to work with.

5 Classifying skills

One of the main goal of the project is to classify the skills as high or low AI/GPT according to the likelihood to be replaced by those technologies.

5.1 Skill database

We do have access to a dataset of skills that was made by taking every skill in the job posting dataset via an ad-level ID.
The dataset comes with nine columns :

- id
- skill_id
- skill_name
- skill_type
- skill_category
- skill_category_name
- skill_subcategory
- skill_subcategory_name
- is_software

The goal here is to clean the database by removing the duplicates. As said before, the skills come from the job posting dataset so a **skill_name** and **skill_id** can have multiple **id**. Once it is done, we can start thinking about building our first model. We went from 7 millions rows to 22 000 which is way easier to manage.

5.2 Semi supervised learning

We have a large dataset of unique skills and we want to label them high or low. However, we can't manage to label the data manually, so what can we do ? This is where semi-supervised learning comes in. With this approach, we can train a classifier on a small amount of labeled data, and then use the classifier to make predictions on the unlabeled data. Since these predictions are likely better than random guessing, the unlabeled data predictions can be adopted as 'pseudo-labels' in subsequent iterations of the classifier.

There are multiple technique of semi-supervised learning but in our case, we will use the self-training method. The reason behind this is because this

method is one of the simplest one and it allows us to "hand-label" (i.e asking ChatGPT) a small part of the data and use it to train our model, which whill be used later in the larger dataset.
It consists of :

- Split the labeled data instances into train and test sets. Then, train a classification algorithm on the labeled training data.
- Use the trained classifier to predict class labels for all of the unlabeled data instances. Of these predicted class labels, the ones with the highest probability of being correct are adopted as 'pseudo-labels'.
- Concatenate the 'pseudo-labeled' data with the labeled training data. Retrain the classifier on the combined 'pseudo-labeled' and labeled training data.
- Use the trained classifier to predict class labels for the labeled test data instances and evaluate classifier performance.

We started by labeling 200 skills but results were not very good by the fact that we have 22 000 distinct skills in our dataset so we decided to label 1000 skills. We went from 0.9% of labeled data to 2,9%. Even if this is not much, it still helps having better performance regarding the method.

Now that we know more about semi-supervised learning and self-training, we have to chose a prediction algorithm. Our goal fits perfectly under the logistic regression since it predict binary values (let's say 1 for high and 0 for low). However, the result were not good. That's why we went for the famous random forest algorithm that is known to be very good for most applications.

Speaking of performance, it is said that it heavily depends on the quality and representativeness of the unlabeled data. Since we didn't have much time to try multiple solutions but we could have tried multiple things such as :

- Class balancing: Ensure that the added data do not imbalance the classes, as this could introduce bias.
- Model ensembles: Use multiple models to label the unlabeled data and only use the labels on which the majority of the models agree.
- Error propagation: If the model makes errors in labeling the unlabeled data, these errors can be propagated and negatively impact the overall performance.

Finally, after building our prediction model, we had to apply it in our job posting database. Therefore, we introduced a new column named **high_low_ratio** that gives us the ratio of high AI_GPT skill to low AI_GPT one. After that, we took the rows that are in the third quantile. We decided to apply it in the whole database which may have been an error since we have 126 000 rows and the process took exactly 211 minutes.

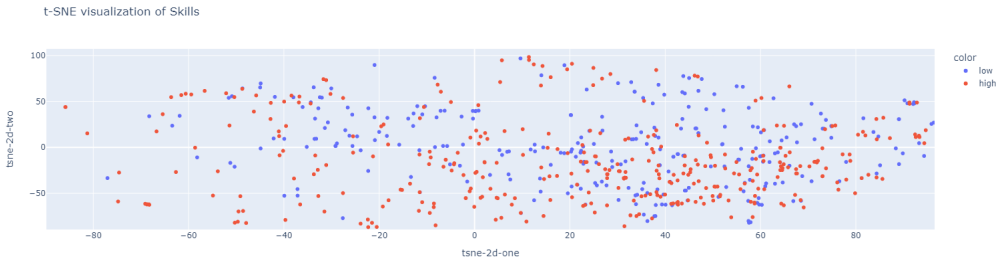

Figure 3: Distribution of the labeled skills

Classification Report:				
	precision	recall	f1-score	support
low	0.83	0.67	0.74	86
high	0.86	0.94	0.90	189
accuracy			0.85	275
macro avg	0.85	0.81	0.82	275
weighted avg	0.85	0.85	0.85	275

Figure 4: Classification report of the prediction model

5.3 Results

Firstly, we should treat cautiously the results regarding the performance of the prediction model because it is a prototype and we need to take into account the semi-supervised learning context.

We do have an acceptable accuracy regarding our model so we decided to use it on the whole database and to do a bar plot. Here again, we took the most frequent jobs that are most likely to be replaced with AI so we can have a clear view.

As we may expect, there are a lot of software engineers since most of their skills are high AI_GPT skills. We see that jobs related to IT are likely to be replaced by AI even with approximately 1/200 of the whole database.

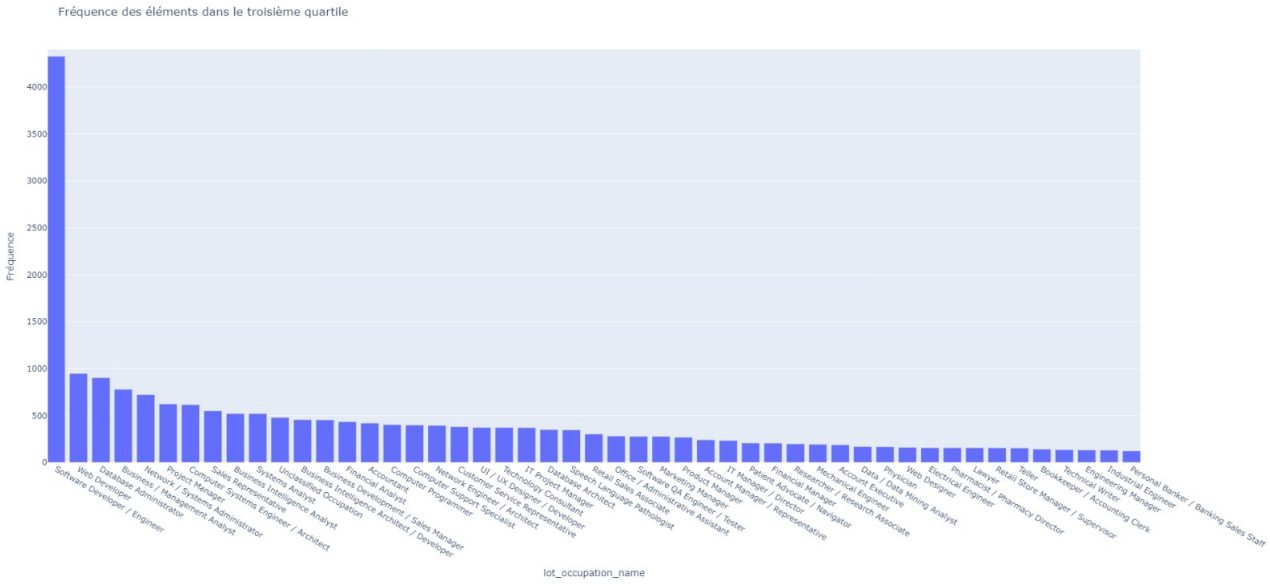

Figure 5: Bar plot

6 Classifying jobs

Another goal of the project is to classify jobs based on their skills.
The methodology employed for categorizing job titles involves several key stages: generating embeddings using BERT, clustering these embeddings, reducing their dimensions for visualization, and finally plotting the results. These stages ensure that skills are accurately categorized. First, BERT transforms skills into numerical vectors, capturing their meanings. These vectors are then clustered to group similar skills. The high-dimensional data is reduced for easier visualization, and finally, the results are plotted to visually represent the clusters, making it simpler to interpret the relationships between different skills.

The input data consists of job titles and associated skills, organized in a DataFrame. The associated skills comes from concatenating three columns from the dataset : **skill_name**, **specialized_skills_name** and **common_skills_name**. The skills for each job title are combined into a single text string to create a consolidated representation of the job's skill set.

Once the dataset is cleaned (i.e having a set of skill associated to a job name), we can start by using BERT, a language model based on the transformer architecture.
BERT (Bidirectional Encoder Representations from Transformers) is used to

transform the textual representation of skills into numerical vectors, known as embeddings. The process involves:

- Tokenization: Breaking down the combined skills text into tokens that BERT can process.
- Model Inference: Passing the tokenized text through the pre-trained BERT model to generate embedding.
- Pooling: Averaging the token embedding to obtain a single fixed-size vector for each job title, representing the semantic meaning of the combined skills.

To have groups of jobs with similar skills, we have to use clustering methods such as K-Means. The K-means algorithm is applied to the embeddings to cluster the job titles based on their skill sets. The number of clusters (set to 20 in this case) is a parameter that can be adjusted depending on the desired granularity of the categorization. Usually, e get the optimal number of cluster by using technique such as the Silhouette method or the Elbow method. The Silhouette method evaluates how similar an object is to its own cluster compared to other clusters, while the Elbow method looks for a point where the rate of decrease in clustering cost slows down, indicating the optimal number of clusters. However, both of the method gives us 2 as an optimal number but it is not coherent given the number of different job name. That's why we will continue to use 20 clusters which is a good number to visualize.

To visualize the high-dimensional embeddings, they need to be reduced to a two-dimensional space. Two techniques are used for this purpose:

- t-SNE (t-Distributed Stochastic Neighbor Embedding): This technique is particularly effective for visualizing high-dimensional data by preserving local structures, making it easier to see how individual job titles relate to their nearest neighbors.
- UMAP (Uniform Manifold Approximation and Projection): UMAP often provides a better balance between preserving local and global structures in the data, potentially offering a more coherent overall view of the clusters.

To have a good classification of jobs using skills, a large amount of cleaned data is needed.
In our case, with quite some data and a not so much cleaned data we get a good visualisation of the different clusters. We still have to keep in mind that the skills for each job are not normalized so it is fine to have the same job but not in the same coordinates

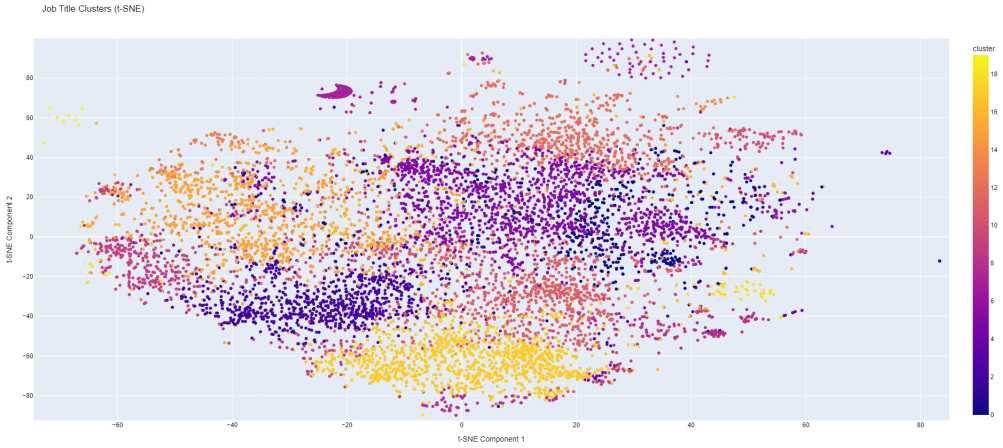

Figure 6: Visualisation of clusters using t-SNE.

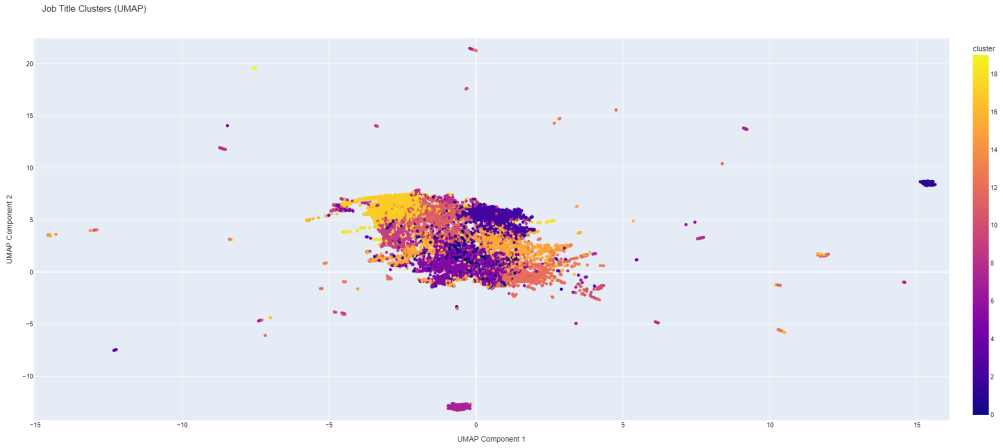

Figure 7: Visualisation of clusters using UMAP.

7 Conclusion

The Cassiopee project had multiple goals such as scraping data from layoffs and LinkedIn before getting the Burning Glass database, build different models to predict high and low AI_GPT skills or even classifying jobs.
Getting the B.G database allowed us to dive deeper in the project even with the restricted time. We tried to first classify tech companies within the database but after few trials we decided to focus on classifying skills and jobs with a restricted database. We think that the models are prototypes of what can be done later one, but we have quite some good results regarding the skill classifier. Next step is to optimize the models and to take into account the whole database by using software such as Google Vertex AI or AWS SageMaker.