CSCE 5290: Natural Language Processing
Project Increment 2

Project Increment 2

Project Title
Resume and Job Posting Keyword and Skill Extractor

GitHub Link
https://github.com/Zedoumous/NLP-5290---Project-Group-17

Team Members - Group 17
Tyler Parks - tylerparks@my.unt.edu
George Thomas - georgethomas2@my.unt.edu
Jofiel Gomez - jofielgomez@my.unt.edu
Tyrell Richardson - TyrellRichardson@my.unt.edu

Informational Abstract
Natural Language Processing, Kaggle, LinkedIn, Skill Extraction, Skill Identification, Skill Comparisons, Job Postings, Resumes, APIs, Scraping, CSO, YAKE, KeyBERT, TextRank, GENSIM, Similarity Scoring, Average Scoring, Architecture Diagrams, Normalization

Introduction

Navigating the job market is difficult as a new graduate. There is a misalignment between academic coursework and expectations from hiring managers. This project is meant to parse through job applications to extract the key skills a student needs to be successful. This allows students to use this key information to expand their studies to include these skills and allow professors to get a deep understanding of the current job market and plan their coursework accordingly.

There are billions, if not trillions of webpages, resumes, files, etc. with key information attached to them. Most of these files/pages, often hold information that can be considered unimportant to a user if they are looking for specific information. For this project case specifically, skill extraction from web pages like job postings or from resumes, can be useful for skipping irrelevant or overwhelming information. For example, if an employer is looking to hire about 10 employees out of 100 applicants. It would be inefficient to scroll through every file to look for the section that lists the employee's skill. The employer would have to do this 100 times. If the employer had a program that could go through an employee's resume and instantly extract information about the skills on their resume, the employer could instantly determine if they are a candidate for the job much sooner, and more efficiently. That is the significance of the project. By creating a program that extracts information quickly, efficiently, and accurately, we can save a lot of users time and effort.

Background

This project is similar work to what all professional businesses are implementing when it comes to resumes. Companies are too busy with other important work and projects and do not have the time and ability to go through the resumes of potential, future employees. They have the ability to use third party companies to complete the process of finding employees, but it will cost them a good amount of money. They have now started to use natural language processing in order to create a tool that can help them go through resumes, one at a time or multiple, and find keywords of highly potential candidates. Similar projects such as keyword extractors have contributed to designing and implementing these features into our own project.

Your Model

Detailed Program End-to-End Architecture

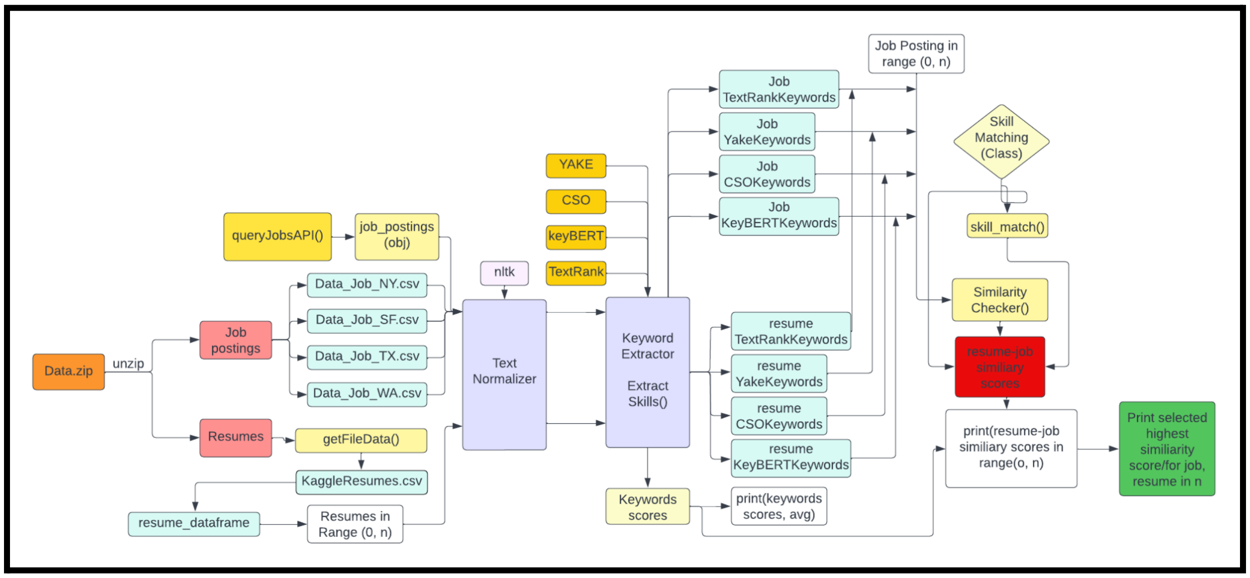

Figure 001: Detailed Program End-to-End Architecture

A compressed Data.zip which holds two folders for resume and job postings is unzipped, separated, and parsed by the getFileData() function. The job postings object and resume dataframe are then passed through the text normalizer function as a corpus for pre-processing using the nltk library. The function includes removing numbers, punctuation marks, white space and lower casing the text going through. The text from each of these corpuses are then passed through the keyword Extractors: KeyBERT, CSO, Yake, and TextRank. Each of these extractor functions print out the keywords and their scores, as well as the averages for the lists of scores. This is then cross referenced to evaluate the most effective keyword extractor.

Each extractor shows a good amount of uniqueness that a person using this tool for a job could fully focus on a certain extractor that fits their needs respectively. The keyword extractions (extracted skills) are then passed to objects that hold the extracted keywords for each iteration of job posting and resume. These lists are then passed through the skill matching object where each job posting is evaluated for similarity of each iteration of a resume, then prints the results indicating which resume-job posting pair has the highest score.

Overhead Keyword Extraction Architecture

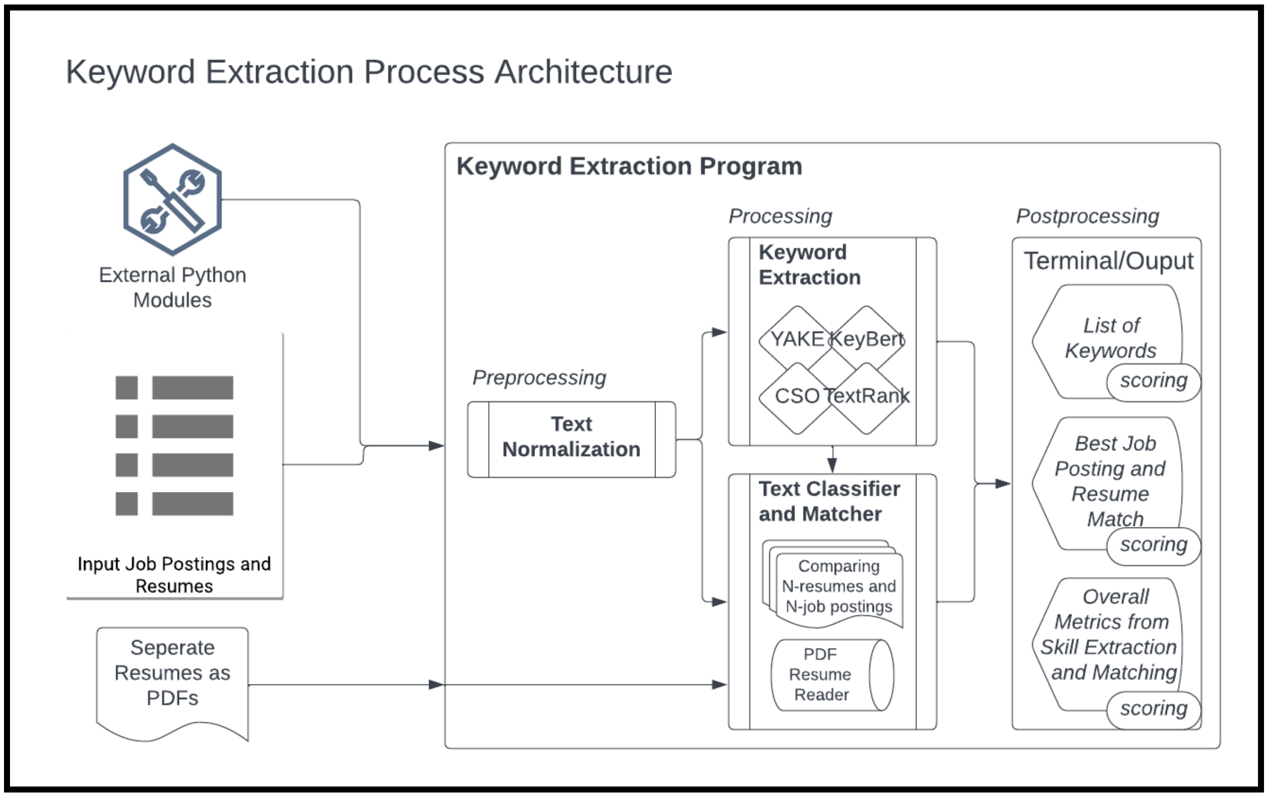
Keyword Extraction Process Architecture
External Python Modules
Input Job Postings and Resumes
Seperate Resumes as PDFs
Keyword Extraction Program
Preprocessing
Text Normalization
Processing
Keyword Extraction
YAKE
KeyBert
CSO
TextRank
Text Classifier and Matcher
Comparing N-resumes and N-job postings
PDF Resume Reader
Postprocessing
Terminal/Ouput
List of Keywords
scoring
Best Job Posting and Resume Match
scoring
Overall Metrics from Skill Extraction and Matching
scoring

Figure 002: Overhead Keyword Extraction Architecture

The above diagram represents the entire keyword extraction process from input to output. After input job postings and resumes are normalized and cleaned, our skill extraction tools process the raw corpi and determine which keywords out of the large input to return to the program. Once returned, our matching tool compares the list of skills acquired from the job postings to input resumes to find a best matching candidate. This matching along with all of these steps accuracy scores can be seen in the output terminal.

Workflow Diagram

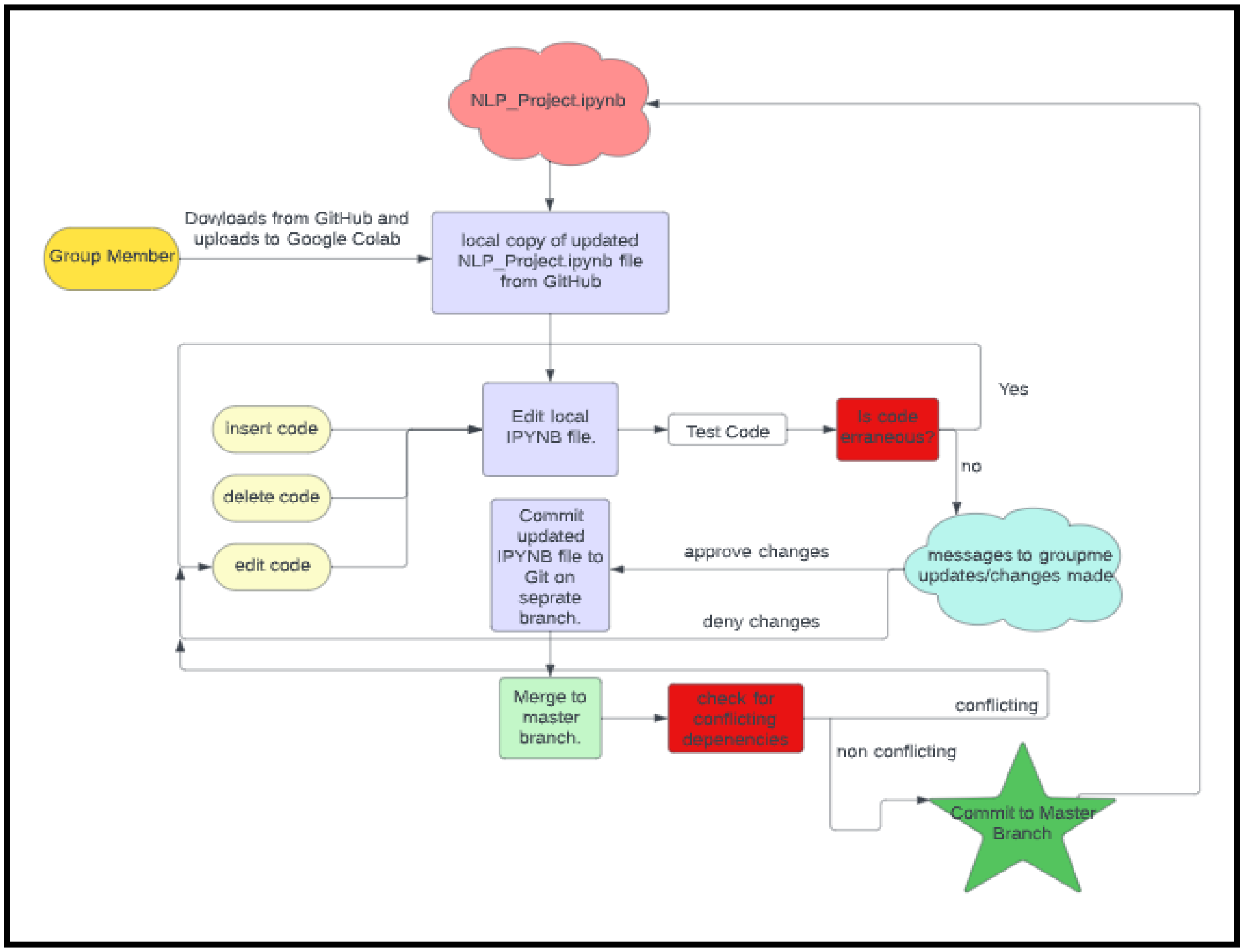
NLP_Project.ipynb
Group Member
Dowloads from GitHub and uploads to Google Colab
local copy of updated NLP_Project.ipynb file from GitHub
insert code
delete code
edit code
Edit local IPYNB file.
Test Code
Is code erraneous?
Yes
no
messages to groupme updates/changes made
approve changes
deny changes
Commit updated IPYNB file to Git on seprate branch.
Merge to master branch.
check for conflicting depenencies
conflicting
non conflicting
Commit to Master Branch

Figure 003: Workflow Diagram

This diagram represents the workflow our group followed throughout our project's development process. In the beginning, we opted to create our program in Python; however, we soon decided that (for dependencies sake), we would switch to using Google Colab and Jupyter files instead. This is where we each would edit, upload, and review each other code on GitHub and repeat this process.

Dataset(s)

The Datasets used in the project were samples of web-scraped job postings and resumes. In Colab we used a zipped “Data” folder that contained two separate folders containing the job postings and the resume postings.

- Uploaded Datasets
    - Job Postings Dataset
        - Description
            - Dataset containing four subsets of job postings from different regions. This set includes a wide variety of information associated with that job posting.
        - Size: 10MB
        - Source: Kaggle
        - Columns: Job_title, Company, State/City, Min_Salary, Max_Salary, Job_Desc, Industry, Date_posted, Valid_until, Job_type, Rating
        - Number of Records:
            - Data_Job_NY.csv: 900
            - Data_Job_SF.csv: 889
            - Data_Job_TX.csv: 643
            - Data_Job_WA.csv: 892
            - **Total**: 3,324

    - Resume Dataset
        - Description
            - Dataset containing just under 1,000 categorized resumes from the fields of computer and data science.
        - Size: 3MB
        - Source: Kaggle
        - Columns: Category, Resume
        - Number of Records:
            - 962

Detail Design of Features

- Resume/Job Posting Identification
    - The program reads in files (.csv/.txt/.pdf) and uses text classification to check if the file is related to resumes or job posting. Most of our input data is prelabeled.

- Information Extraction
    - The program can identify skills within a specific unit of the dataset (a single job posting and or resume). This is used for the alignment analysis with identified skills of a matching Job Posting or Dataset.

    - We are using a diverse portfolio of keyword extraction tools and text classifiers…
        - keyBert
        - Computer Science Ontology
        - YAKE
        - TextRank

      …in order to extract the most telling and *intentional* pieces of data within the text. This includes skill words, requirements, spotlighted text, and more.

- Preprocessing
    - Our group needed to normalize the incoming resumes and job postings in order to ensure that our skill extraction and matching algorithms were receiving the cleanest data possible. Listed below are some of the normalization methods we employed:
        - Removed English Stopwords using NLTK
        - Converted to all lowercase
        - Removed digits
        - Remove most extraneous punctuation
        - Removed extraneous whitespaces
            - Tabs
            - Newlines
        - Tokenization of the corpi

- Skill Classification and Matching
    - Each skill is assigned a score which determines how relevant that skill is to the entire document, i.e. how well that skill represents the corpus as a whole.
    - This is accomplished by using keyword extraction tools on both the resume and the job postings. We then compute cosine similarity using the GENSIM library for every token in each document and assign them a similarity score. We then choose the resume keywords and job posting document pairing with the largest score.

- Resume, Job Posting Matching
    - This feature allows a user to upload a resume in PDF format and run it against live Job Postings to find keywords that most fit the career they are looking for.
    - If we continue this feature in the future, we would implement a web GUI to upload a resume and use an improved method to visualize matching results.

Analysis of Data

- Data Pre-processing (Normalization)
  The data used in the Application are a series of CSV files that are required to be uploaded to the Colab Notebook. Our program also pulls job postings from an external API and combines that information with the input job postings from Kaggle. This data is first cleaned using the aforementioned methods:
    - Removed English Stopwords using NLTK: **Reduced number of non-key words within the text.**
    - Converted to all lowercase: **Cleaning**
    - Removed digits: **Cleaning**
    - Remove most extraneous punctuation: **Cleaning**
    - Removed extraneous whitespaces: **Cleaning**
        - Tabs
        - Newlines
    - Tokenization of the corpi: **Analytics and I/O to other functions**

Resume Data

Our resume dataset is very simple to use and read. Shown below, each resume is categorized and provides a full raw copy of the resume text.

Category	**Resume**
Data Science	Skills * Programming Languages: Python (pandas, numpy, scipy, scikit-learn, matplotlib), Sql, Java, JavaScript/JQuery. * Machine learning: Regression, SVM, NaÃ¯ve Bayes, KNN, Random…
…	…

Figure 004: Resume Data

Job Posting Data
The job posting data is full of information including the job posting description, which contains the skills that we are looking for. Many of the job postings we use require this data column.

Job_title	Company	State	City	Min_ Salary	Max_ Salary	Job_Desc	Industry	Rating
Chief Marketing Officer (CMO)	National Debt Relief	NY	New York	n/a	n/a	Who We're Looking For…	Finance and Data Science	4.0
…	…	…	…	…	…	…	…	…

Figure 005: Job Posting Data

Implementation

Our group implemented the 'Resume and Job Posting Keyword and Skill Extractor' project using Python via Google Colab and Jupyter files. Python and Colab are great for their functionality in the world of NLP applications. Many of the tools that we used in previous class assignments are featured in this project as well.

Note: Follow along using the uploaded `NLP_Project.ipynb` *file, as many of the steps in this section correlate to the table of contents in that Colab environment.*

1. *Data Retrieval*
   Inside of the *Driver Code* Section, we use a helper function named `getFileData()` with two parameters as input, file name and document type, to fetch our Kaggle datasets. These .csv files are converted into Pandas Dataframes and then passed to the skill extractors for further use.

2. *API Setup*
   In this project we utilized the USA Jobs API. In order to connect we needed to generate a private key and use our email address for the headers in the GET request. The end point we used was "api/search". According to the documentation[8], we have a number of parameters we can pass in the GET request. Using the requests library, we make the query and get a JSON object. Once converted, it is an array that we can loop through to find data on the returned job postings including but not limited to: location, pay grade, Job Title, and Job Descriptions. In our project, we focused on the last two in our matching algorithm.

3. *Skill Extraction Functions*
   a. KeyBERT
      This tool uses a pre-trained KeyBERT model named `all-mpnet-base-v2`. This model includes many different flag parameters that allow the user to customize and test the KeyBERT model.

   b. CSO
      The Computer Science Ontology is a collection of over 400,000 different CS topics and skills. This tool connects and compares a given corpus with that collection and returns computer science keywords found within that corpus.

   c. YAKE
      This tool is similar to KeyBERT in that it uses many different flags; however, this tool is based on algorithmic reduction of keywords.

   d. TextRank
      This tool is based off of PageRank (A tool for ranking web pages in online search result queries) and is an extractive and and unsupervised text summarization technique.

4. *Skill Matching*
   This class ‘SkillMatching’ is the processing unit and storage for all process job postings and resumes that go through our program. The class has the ability to match two sets of skills, generate similarity scores for those skills, and display the results of its processing. This matcher is where each skill extractor is executed and results returned as well.

5. *Driver Code*
   This section of our program fetches the dataset and begins skill extraction on a defined number of job postings and resumes samples. Each skill extraction tool is given a score along with each resume/job posting comparison.

6. *Resume/JP Matching*
   Using GENSIM, our program is able to cross-reference each extracted skill from a given job posting and resume. Our resume/job posting matching algorithm takes advantage of GENSIM’s cosine similarity scoring to enable such comparisons. Once a number of resumes and job postings have been compared, a best contender is stored to show to the user.

7. *Resume Upload*
   The ‘Resume Upload’ section allows a user to upload a .pdf file (one is provided) to the Colab instance. This file is ‘mined’ using a Python dependency called `pdfminer3` for the skills located within. These skills are used to compare against known skills taken from job postings previously extracted. The program finds the best job posting match for the input resume and returns that data.

Results

Sample Scores (keywords) - Job Posting #1

keyBERT Extractions		CSO Extractions		YAKE Extractions		TextRank Extractions	
Law Enforcement	0.5326	Information Management	n/a	Fully remote position	0.0019	management	0.2693
Provide Data Management	0.4489	Data Management	n/a	Analytical support technology	0.0028	office	0.2398
Technology branch office	0.4378	Electronic Records	n/a	Support technology branch	0.0028	trust	0.2138
Scientist provide	0.3831	Records Management	n/a	Specific body worn	0.0028	data	0.2133

Figure 006: Sample Scores (keywords) - Job Posting #1

Sample Scores (keywords) - Resume #1

keyBERT Extractions		CSO Extractions		YAKE Extractions		TextRank Extractions	
Text analytics	0.5406	css	n/a	Coding topic modeling	.00019	information	0.2170
categories	0.3252	Decision trees	n/a	Tools technology python	.00020	analytics	0.2170
standards	0.2127	cyber-attack s	n/a	Predictive coding topic	.00022	Details data	0.1702
Deep learning education	0.2003	vehicles	n/a	Science assurance associate	.00023	models	0.1702

Figure 007: Sample Scores (keywords) - Resume #1

These tables are samples of the first few epochs of the varying keyword extractors, with the sample words and scores they pulled. The first table is a sample of a Job posting, and the second is a sample of a resume. These scores displayed are then put through a checksimiliarty()

function that checks the similarity of every word in this list for the resume and job posting and returns an average score.

Similarity Mean Scores

Skill Extractor Tool	**Similarity Mean**
KeyBERT match Similarity Mean:	0.3252
CSO match Similarity Mean:	0.2003
Yake match Similarity Mean:	0.2170
TextRank match Similarity Mean:	0.2693

Figure 008: Similarity Mean Scores

Another Method of skill matching we experimented with was with the creation of a skill matching class. This class was an exaggeration of the previously used method. This method would take in the extracted keywords just as the similaritychecker() would and enter the output of the extracted skills into a dictionary which was then passed through the skill matching(obj)'s skill_match and sim_score() methods. Both of these functions within the class match and score correlating job posting and resume skills and returns matching job postings for each epoch of resumes.

Matching Job Posting for Resume 1

Skill Extractor Method	**Type of Resume**	**Matching Job Posting Text and Skill Words**
keywordsBERT	Data Scientist	data scientist provide data management analytical support technology branch office office public trust specific body worn camera electronic records management systems office created strengthen public trust national park services law enforcement programs transparency availability accessibility information fully remote position
keywordsCSO	Data Scientist	position located department health human services hhs centers medicare medicaid services cms office burden reduction health informatics obrhi emerging innovations group data scientist gs design develop implement analytical statistical programming mechanisms necessary collect organize analyze interpret unique highly specialized data sets

keywordsYAKE	Data Scientist	position located department health human services hhs centers medicare medicaid services cms office burden reduction health informatics obrhi emerging innovations group data scientist gs design develop implement analytical statistical programming mechanisms necessary collect organize analyze interpret unique highly specialized data sets

Figure 009: Matching Job Posting for Resume 1

Matching Job Posting for Resume 2

Skill Extractor Method	**Type of Resume**	**Matching Job Posting Text and Skill Words**
keywordsBERT	Data Scientist	data scientist provide data management analytical support technology branch office office public trust specific body worn camera electronic records management systems office created strengthen public trust national park services law enforcement programs transparency availability accessibility information fully remote position
keywordsCSO	Data Scientist	position located department health human services hhs centers medicare medicaid services cms office burden reduction health informatics obrhi emerging innovations group data scientist gs design develop implement analytical statistical programming mechanisms necessary collect organize analyze interpret unique highly specialized data sets
keywordsYAKE	Data Scientist	data scientist provide data management analytical support technology branch office office public trust specific body worn camera electronic records management systems office created strengthen public trust national park services law enforcement programs transparency availability accessibility information fully remote position

Figure 010: Matching Job Posting for Resume 2

Project Management

Find below our group's Completed Responsibility Table and individual contributions to the project as a whole. These descriptions detail each team member's contribution of overall work from end-to-end on our project.

Completed Responsibility Table *(Task, Person)*

Finished Feature	**Description**	**Team Member**
Initial Collection of Small Kaggle Datasets	Identified small datasets of job postings and resumes to be used for testing.	Tyler Parks
External Job Posting Collection via API or Scraping	A scraper that collects raw text data from specified URLs.	George Thomas
Normalization of Input Text	Format input text into the most basic form for optimal use for language model and text classifiers.	Jofiel Gomez and Tyrell Richardson
Skill Extraction using **KeyBERT**	Extracts keywords from input text using the KeyBert Tool	Tyler Parks
Skill Extraction using **CSO Classifier**	Extracts keywords from input text using the CSO Classifier	Tyler Parks
Skill Extraction using **Yake Classifier**	Extracts keywords from input text using the Yake Classifier.	George Thomas
Skill Extraction using **TextRank Classifier**	Extracts keywords from input text using the TextRank Classifier.	Tyrell Richardson
Comparing Resume and Job Posting to find 'Best Match'	A feature that will compare the analysis of different resumes and job postings to match similar documents.	George Thomas
Skill Classification/Clustering	Clustering the skills we extract from each posting/resume.	Tyrell Richardson
Identifying and determining neutral input as being either a Resume or a Job Posting	Document classification.	Tyrell Richardson
Utilizing Jobs API to 'match' resumes	Grabbing job postings from online sources based on matches and values we generate	George Thomas
Adding Scoring Metrics for	Provides scores for each resume-job posting	George Thomas and

Resume and Job Postings Matching	skills comparison, used as a measure for similarity.	Tyler Parks
Adding Scoring Metrics for Skill Extractor Analysis	Provides an accuracy score for each skill extractor based on the scores of each individual skill.	Tyrell Richardson

Figure 011: Completed Responsibility Table

Individual Contributions

- George (25%):
  - External Job Posting Collection via API or Scraping
  - Skill Extraction using **Yake Classifier**
  - Utilizing Jobs API to 'match' resumes
  - Comparing Resume and Job Posting to find 'Best Match'
  - Adding Scoring Metrics for Resume and Job Postings Matching

- Tyler (25%):
  - Initial Collection of Small Kaggle Datasets
  - Skill Extraction using **CSO Classifier**
  - Skill Extraction using **KeyBERT**
  - Adding Scoring Metrics for Resume and Job Postings Matching

- Tyrell (25%):
  - Adding Scoring Metrics for Skill Extractor Analysis
  - Identifying and determining neutral input as being either a Resume or a Job Posting
  - Text Normalization
  - Created Github Repo
  - Skill Classification/Clustering
  - Skill Extraction using **TextRank Classifier**

- Jofiel (25%):
  - Text Normalization
  - Keyword Extraction Tool collaborator
  - Text Selection

- **Total**, *for Increment 2* (*max 100%*): **100%!**

Issues and Concerns

Below describes unsolved issues that our group encountered and discovered over the course of development.

- Issue #1:
  Finding a reliable method for measuring the accuracy of each text classifier from new job postings. The current metric (accuracy scores) are collected from averages, scores generated from the skill extraction tools, and GENSIM similarity scores; while these are accurate metrics, our program would likely benefit from measuring an F1 score and using a confusion matrix of some sort.

- Issue #2:
  Finding a notebook/UI for program execution outside of Google Colab. As Colab was great in providing a modular way to create our program, it would have been nice to execute in a different environment for testing.

- Issue #3:
  Implementing the Linkedin API to scrape job postings from the Linkedin website. The current USA Jobs API is robust but is limited to public sector job postings.

- Issue #4:
  Automating this process would ideally lead to Skill Matching-as-a-service -- where users can compare their resumes and job postings against the other on the spot.

Future Work:

This program can be improved by using a combination of the different skill extraction tools to formulate an accurate depiction of the job posting through a list of skills. This would greatly improve the accuracy of our comparison model and thus improve the final result when testing against input resumes.

References/Bibliography

[1] arianpasquali and vitordouzi, “Yake,” *PyPI*. [Online]. Available: https://pypi.org/project/yake/. [Accessed: 01-Nov-2022].

[2] “Keybert,” *PyPI*. [Online]. Available: https://pypi.org/project/keybert/. [Accessed: 01-Nov-2022].

[3] “Yake,” *PyPI*. [Online]. Available: https://pypi.org/project/yake/. [Accessed: 01-Nov-2022].

[4] “cso-classifier,” *PyPI*. [Online]. Available: https://pypi.org/project/cso-classifier/. [Accessed: 01-Nov-2022].

[5] P. Joshi, “Automatic text summarization using TextRank algorithm,” *Analytics Vidhya*, 15-Jun-2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/. [Accessed: 03-Nov-2022].

[6] E. Fontana, “Bow + tf-IDF in python for unsupervised learning task,” *Medium*, 14-Sep-2020. [Online]. Available: https://medium.com/betacom/bow-tf-idf-in-python-for-unsupervised-learning-task-88f3b63ccd6d. [Accessed: 03-Nov-2022].

[7] “Gensim: Topic modeling for humans,” *Radim ÅehÅ-Åek: Machine learning consulting*, 06-May-2022. [Online]. Available: https://radimrehurek.com/gensim/. [Accessed: 03-Dec-2022].

[8] “USA Jobs API Reference - GET /api/Search” [Online]. Available: https://developer.usajobs.gov/API-Reference/GET-api-Search. [Accessed: 01-Nov-2022].

[9] “Shape America's future,” *Job Search*. [Online]. Available: https://www.usajobs.gov/. [Accessed: 03-Dec-2022].