JOB RECOMMENDATION SYSTEM BASED ON NATURAL LANGUAGE PROCESSING

STUDENT NAME:	Zixuan Hou
STUDENT NUMBER:	14427261

29/5/2024

Content

Overview

In today's fast-evolving job market, finding jobs that match one's skills and interests has become increasingly important and challenging. As the RecSys Challenge 2016 highlighted, developing sophisticated job recommendation systems is crucial for enhancing the user experience in job search platforms (Abel et al., 2016). Knowing which jobs match one's skill set in a timely manner can not only significantly improve the success rate of job hunting, but also help one's career planning. Therefore, it is particularly important to develop a system that can automatically identify personal skills and recommend matching jobs.

This project integrates NLP and machine learning technologies to develop a job recommendation system. By crawling job postings on the Indeed website and simulating browser operations using the Selenium automation framework, job-related data can be efficiently collected. In addition, a variety of data preprocessing techniques, such as lemmatization, stop word removal, and TF-IDF conversion, are used to help us extract meaningful features from the original text and provide input for subsequent machine learning models.

Then, by training a variety of machine learning models including random forests, support vector machines, and neural networks, the system can more accurately predict and recommend jobs that best suit the user's skills. Through the implementation of this project, an effective tool is provided for job seekers to analyze and use job information on the market in a scientific way, helping them find the most suitable jobs in a highly competitive employment environment.

Method and Results

WebCrawler

Indeed was chosen as the main data source for this project. The Indeed website covers positions in multiple industries and professional fields, which makes it an ideal data source to obtain descriptions and demand data for various occupations.

To efficiently collect data from the Indeed website, I used Selenium, an automated testing tool that provides a way to simulate web browser operations. With Selenium, we can automatically navigate web pages and simulate user interactions such as clicking and scrolling, so as to effectively crawl job information.

First configure Selenium's WebDriver and set up the Firefox browser.

```
# Set the Firefox driver path and Firefox executable file path
driver_path = './geckodriver'
firefox_binary_path = 'C:/Program Files/Mozilla Firefox/firefox.exe'
```

```
    # Setting up Firefox
    options = webdriver.FirefoxOptions()
    binary = FirefoxBinary(firefox_binary_path)
    options.binary = binary
```

Code 1: Setting up Firefox

Dynamically construct query URLs based on the type of job to be crawled so that the corresponding job listing page is loaded.

```
    # Setting the crawl URL
    url = f'https://www.indeed.com/jobs?q={job_query}&l='
    driver.get(url)
    time.sleep(5)  # Waiting for the page to load
```

Code 2: Setting url

In this project, I selected 101 different job types for data crawling and analysis. This selection was based on the following considerations:

(1) Industry representativeness: The selected jobs cover multiple industries from technology, medical care, education to management, ensuring the diversity and representativeness of the data set.

(2) Occupational diversity: Including different positions from entry-level to senior management, such as network engineers, financial analysts, human resources managers, etc., helps us analyze and understand the skill requirements and responsibilities of positions at different levels.

(3) Skill compatibility: These positions have a certain degree of overlap in skill requirements, allowing us to explore the commonality of skills between different positions and their potential impact on career paths. This is particularly important for designing a cross-professional recommendation system.

Through a comprehensive analysis of these 100 positions, users can find jobs that best match their skills and career interests in a complex and changing career environment.

```
# Define a list of job types to crawl
job_queries = [
    'data+scientist', 'software+engineer', 'project+manager', 'product+manager',
    'business+analyst', 'web+developer', 'graphic+designer', 'network+engineer',
    'systems+administrator', 'database+administrator', 'quality+assurance',
    'technical+support', 'data+analyst', 'security+analyst', 'data+engineer',
    'machine+learning+engineer', 'devops+engineer', 'cloud+architect', 'it+manager',
    'seo+specialist', 'digital+marketer', 'content+writer', 'sales+manager',
    'financial+analyst', 'accountant', 'hr+manager', 'marketing+manager',
    'customer+service+representative', 'operations+manager', 'logistics+coordinator',
    'procurement+manager', 'supply+chain+analyst', 'legal+assistant', 'paralegal',
    'nurse', 'medical+assistant', 'pharmacist', 'physical+therapist',
    'research+scientist', 'lab+technician', 'teacher', 'school+principal',
    'instructional+designer', 'education+consultant', 'library+assistant',
    'counselor', 'psychologist', 'social+worker', 'therapist',
    'engineer', 'mechanical+engineer', 'civil+engineer', 'electrical+engineer',
    'chemical+engineer', 'environmental+engineer', 'aerospace+engineer',
    'biomedical+engineer', 'architect', 'urban+planner',
    'construction+manager', 'electrician', 'plumber', 'carpenter',
    'welder', 'machinist', 'automotive+technician', 'hvac+technician',
    'chef', 'restaurant+manager', 'bartender', 'waiter',
    'event+planner', 'travel+agent', 'tour+guide', 'flight+attendant',
    'pilot', 'aircraft+mechanic', 'bus+driver', 'truck+driver',
    'mechanic', 'dispatcher', 'safety+manager', 'safety+specialist',
    'janitor', 'custodian', 'groundskeeper', 'landscaper',
    'security+guard', 'firefighter', 'police+officer', 'detective',
    'correctional+officer', 'emergency+medical+technician', 'paramedic',
    'fitness+trainer', 'personal+trainer', 'massage+therapist',
    'esthetician', 'hair+stylist', 'cosmetologist', 'barber'
]
```

Code 3: List of job types

Load the job listing page using WebDriver, then parse the page elements to extract information such as job title, company name, location, and posting date. For each job detail, click the job link to access the detailed description page.

```
# Creating a WebDriver Instance
driver = webdriver.Firefox(executable_path=driver_path, options=options)
wait = WebDriverWait(driver, 10)

while job_count < num_jobs_to_scrape:
    # Capture job cards
    job_cards = driver.find_elements(By.CSS_SELECTOR, '.job_seen_beacon')

    for job_card in job_cards:
        if job_count >= num_jobs_to_scrape:
            break
        try:
            # Extract job title, company name, location, and posting date
            title = job_card.find_element(By.CSS_SELECTOR, 'h2.jobTitle span').text
            company = job_card.find_element(By.CSS_SELECTOR, 'span[data-testid="company-name"]').text
            location = job_card.find_element(By.CSS_SELECTOR, 'div[data-testid="text-location"]').text
            date_posted = job_card.find_element(By.CSS_SELECTOR, 'span[data-testid="myJobsStateDate"]').text
```

Code 4: Using Webdriver

On the job detail page, extract the job description and other relevant information, such as salary range. Use explicit waits to ensure that page elements have been loaded to obtain complete job information.

```
# Click on the job card to view the job description
job_card.find_element(By.CSS_SELECTOR, 'a').click()  # Make sure to click on the link to view the detailed description
time.sleep(3)  # Waiting for job descriptions to load

# Wait for the job description to load
job_description_element = wait.until(
    EC.presence_of_element_located((By.CSS_SELECTOR, '#jobDescriptionText'))
)
job_description = job_description_element.text

# Extract salary information
try:
    salary = driver.find_element(By.CSS_SELECTOR, 'span.css-19j1a75').text
except:
    salary = 'N/A'
```

Code 5: Job card details

Automatically detect and click the "next page" button to crawl multiple pages of data in a loop until the predetermined number of positions is reached or all relevant pages are crawled.

```
# Try to find and click the "Next Page" button
try:
    next_button = driver.find_element(By.CSS_SELECTOR, 'a[aria-label="Next"]')
    next_button.click()
    time.sleep(5)  # Waiting for a new page to load
except Exception as e:
    print("No more pages to load.")
    break
```

Code 6: Go next page

At the same time, after the extraction was completed, I tried to separate the requirements and skills directly from the job description and split them by keywords, but found that the effect was not good, because the keywords would be affected by writing habits and lead to inaccurate extraction of content, so these two parts were not used in the subsequent analysis.

```python
def extract_requirements_and_skills(description):
    # Assume that the job requirements and required skills are contained between keywords
    requirements_keywords = ['Requirements', 'Qualifications', 'Must have']
    skills_keywords = ['Skills', 'Proficient in', 'Experience with']

    requirements = []
    skills = []

    lines = description.split('\n')
    capture_requirements = False
    capture_skills = False

    for line in lines:
        if any(keyword in line for keyword in requirements_keywords):
            capture_requirements = True
            capture_skills = False
        elif any(keyword in line for keyword in skills_keywords):
            capture_skills = True
            capture_requirements = False
        elif line.strip() == '':
            capture_requirements = False
            capture_skills = False

        if capture_requirements:
            requirements.append(line.strip())
        if capture_skills:
            skills.append(line.strip())

    return ' '.join(requirements), ' '.join(skills)
```

Code 7: Separating needs and skills

All the crawled data is only used for academic research and educational purposes, and does not involve any commercial use. Through these methods, we collected job data and laid the foundation for subsequent data cleaning and machine learning model development. In the case of no crawling, we can use the public data sets on Kaggel, which are larger in size and will have better effects on model training.

[4]:	Title	Company	Location	Date Posted	Description	Salary	Requirements	Skills
0	Data Scientist	Microsoft	Redmond, WA 98052 \n(Overlake area)	Posted\nToday	The Azure Core Organization is responsible for...	81, 900—160,200 a year	Qualifications Required Qualifications: Bachel...	NaN
1	Early Career Data Scientist	WSP	Portland, ME 04101 \n(Downtown area)	Posted\nPosted 1 day ago	Our Business\nWe are a global leader in enviro...	NaN	Required Qualifications Bachelor¡¯s Degree in ...	NaN
2	Data Scientist	Booz Allen	Arlington, VA	Posted\nToday	Data Scientist\nThe Opportunity:\nAs a data sc...	75, 600—172,000 a year	NaN	Experience with data exploration, data cleanin...
3	Data Scientist I	Warner Bros. Discovery	Bellevue, WA 98004 \n(Downtown area)	Posted\nPosted 1 day ago	Welcome to Warner Bros. Discovery¡ the stuff ...	91, 000—169,000 a year	NaN	NaN
4	Data Scientist	Microsoft	Atlanta, GA	Posted\nPosted 8 days ago	Are you interested in a start-up like environm...	98, 300—193,200 a year	Qualifications Required Qualifications: Doctor...	NaN
...	...	...	...	...	...	...	...	...
1005	2024-2025 General Resource Teacher (K-12) - Ba...	Clark County School District	Las Vegas, NV 89183 \n(Paradise area)	Posted\nPosted 2 days ago	2024-2025 General Resource Teacher (K-12) - Ba...	NaN	NaN	NaN
1006	(PORTSMOUTH NAVAL HOSPITAL) - BARBER	Navy Exchange Service Command	Portsmouth, VA	Posted\nPosted 30+ days ago	Job Number: 24000137\nPrimary Location: United...	$21.01 an hour	NaN	NaN
1007	Hairstylist/Barber	Detroit Athletic Club	Detroit, MI 48226 \n(Downtown area)	Posted\nPosted 30+ days ago	Job Requirements: \nPrevious work-related skill...	NaN	Job Requirements: Previous work-related skill,...	NaN
1008	BARBER - PORT HUENEME MAIN STORE - FULL-TIME (...	Navy Exchange Service Command	Port Hueneme, CA	Posted\nPosted 30+ days ago	Job Number: 2400013Y\nPrimary Location: United...	$21.62 an hour	NaN	NaN
1009	BARBER	Navy Exchange Service Command	Great Lakes, IL	Posted\nPosted 5 days ago	Job Number: 240001SN\nPrimary Location: United...	$20.95 an hour	NaN	NaN

1010 rows × 8 columns

Figure 1: crawled data

Data wrangling

After successfully crawling the job data from Indeed, we implemented a series of detailed data cleaning and preprocessing steps to ensure data quality and suitability,By applying techniques such as lemmatization, stop word removal, and TF-IDF conversion, we extracted meaningful features from the text data (Dai et al., 2019; Vargas-Calderón & Camargo, 2019), and provide accurate input for machine learning models.

(1) Remove HTML tags and special characters: HTML tags and non-alphabetic characters are removed from job descriptions through regular expressions to ensure the clarity of text content.

(2) Text normalization: All crawled text data is converted to lowercase and tokenized to unify the data format and reduce processing complexity.

(3) Remove stop words and lemmatize: Common English stop words are removed using the NLTK library, and lemmatization is implemented through WordNetLemmatizer to streamline the vocabulary and restore the basic form of the vocabulary.

```python
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Cleanup Function
def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Participle
    words = text.split()
    # Stop word removal and lemmatization
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    return ' '.join(words)
```

Code 8: Data cleaning function

After this, I found a problem. Different companies use different names for the same positions, which leads to them not being classified into one category in model training. So I standardized the position names here to reduce this difference. By mapping the position names to a predefined list of standard positions, the position names in every 10 data records are uniformly replaced with standard names to ensure data consistency.

```python
# 10 for each profession, standardize the profession query list
standardized_titles = []
for job_query in job_queries:
    standardized_title = job_query.replace('+', ' ')
    standardized_titles.extend([standardized_title] * 10)

# Add standardized occupation titles to the dataset
data['standardized_title'] = standardized_titles
```

Code 9: Standardize title

	Before Title \	**After** standardized_title
0	Data Scientist	data scientist
1	Early Career Data Scientist	data scientist
2	Data Scientist	data scientist
3	Data Scientist I	data scientist
4	Data Scientist	data scientist
5	Data Scientist	data scientist
6	Data Scientist	data scientist
7	Data Scientist	data scientist
8	Data Scientist	data scientist
9	Data Scientist21	data scientist
10	Entry Level Software Engineer	software engineer
11	Software Engineer AMTS	software engineer
12	Software Engineer - Early in Career (Backend/F...	software engineer
13	Entry Level Software Engineer	software engineer
14	Software Engineer - Front End (React)	software engineer
15	Software Triage Engineer - tvOS Platform	software engineer
16	Software Engineer-I	software engineer
17	Software Engineer	software engineer
18	Software Engineer	software engineer
19	Legislative Junior Software Engineer	software engineer

Figure 2: Standardized title

After completing the data cleaning and preprocessing, I conducted an in-depth keyword analysis of the text in the job description. By applying the TF-IDF technique, the most critical and unique words in the job description were identified.

```python
# Extract keywords using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=100)
tfidf_matrix = tfidf_vectorizer.fit_transform(data['cleaned_description'])
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
tfidf_sum = tfidf_matrix.sum(axis=0)
tfidf_freq = [(word, tfidf_sum[0, idx]) for word, idx in zip(tfidf_feature_names, range(tfidf_sum.shape[1]))]
tfidf_freq = sorted(tfidf_freq, key=lambda x: x[1], reverse=True)

# Convert keywords and frequencies into DataFrame
tfidf_df = pd.DataFrame(tfidf_freq, columns=['Keyword', 'Frequency'])

# Visualize the top 20 keywords
plt.figure(figsize=(14, 7))
sns.barplot(x='Frequency', y='Keyword', data=tfidf_df.head(20))
plt.title('Top 20 Keywords in Job Descriptions')
plt.xlabel('Frequency')
plt.ylabel('Keyword')
plt.show()
```

Code 10: TF-IDF and plot key words

A bar chart was drawn using the Seaborn library to visually display the frequency of these keywords. As shown in the figure, words such as "work", "experience", and "team" frequently appear in job descriptions. Most of the high-frequency words are related to work experience, teamwork, and responsibilities, which reflects that employers generally value candidates' practical ability and teamwork ability when recruiting.

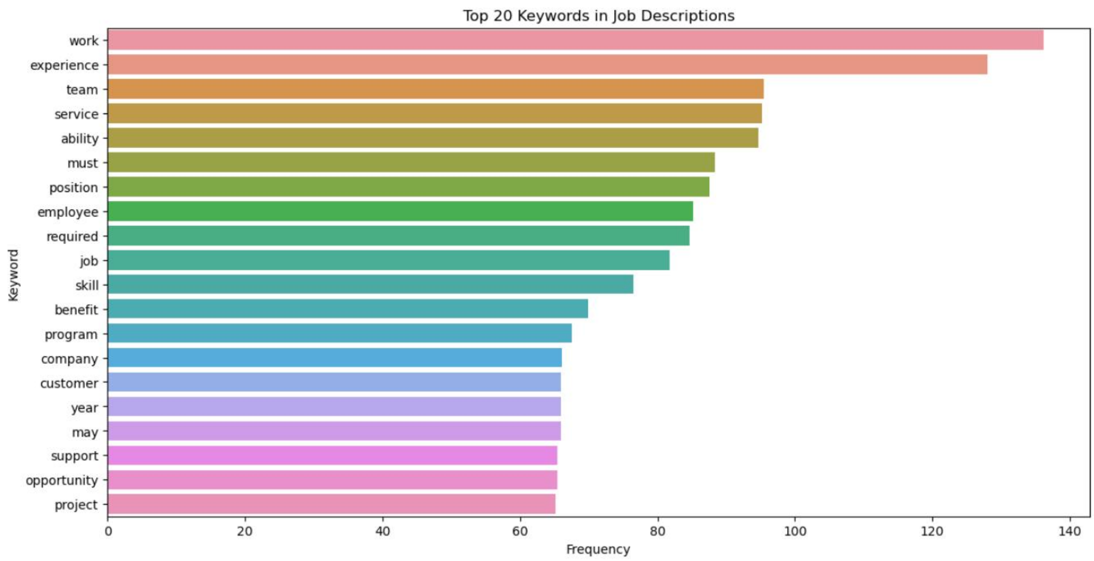


Figure 3: Top 20 key words

The word cloud diagram is further used to visualize the high-frequency words in the job description. The weight or frequency of words is indicated by text of different sizes, making the data analysis results easier to understand.

```python
# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(dict(tfidf_freq))
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Job Descriptions')
plt.show()
```

Code 11: Word clouds

Figure 4: Word Clouds

Later, I compared keywords for two different jobs.

```python
compare_jobs = ['data scientist', 'barber']

# Calculate and visualize job keywords
def analyze_job_keywords(job_title):
    job_data = data[data['standardized_title'] == job_title]
    tfidf_vectorizer = TfidfVectorizer(max_features=100)
    job_tfidf_matrix = tfidf_vectorizer.fit_transform(job_data['cleaned_description'])
    job_tfidf_sum = job_tfidf_matrix.sum(axis=0)
    job_tfidf_freq = [(word, job_tfidf_sum[0, idx]) for word, idx in zip(tfidf_vectorizer.get_feature_names_out(), range(job_tfidf_sum.shape[1]))]
    job_tfidf_freq = sorted(job_tfidf_freq, key=lambda x: x[1], reverse=True)
    job_tfidf_df = pd.DataFrame(job_tfidf_freq, columns=['Keyword', 'Frequency'])
    return job_tfidf_df

# Calculate keywords for two positions separately
data_scientist_keywords = analyze_job_keywords('data scientist')
barber_keywords = analyze_job_keywords('barber')
```

Code 12: Analyze two different job's key words

For the data scientist position, the keywords are mainly focused on technical and professional skills, such as "data", "ml", and "model". This reflects the statistical analysis and machine learning skills that data scientist candidates need to have.

In the barber job description, the keywords are more focused on service and operational skills, such as "client", "hair", and "cosmetology". These keywords emphasize the importance of customer service and specific manual skills.

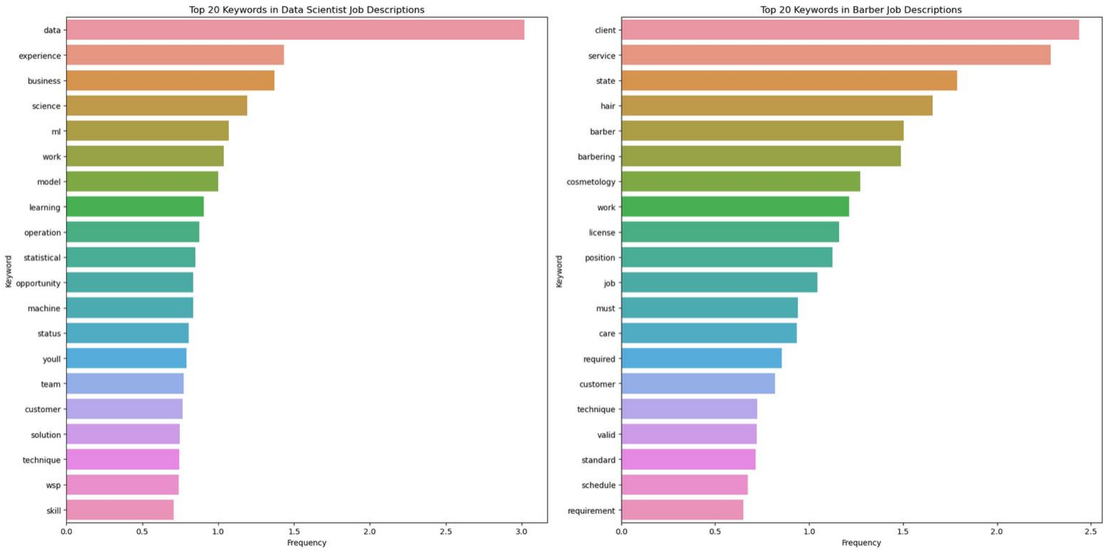


Figure 5: Key words of two different jobs

The above analysis demonstrates the use of TF-IDF technology to extract information from job descriptions, and visualizes the keywords in job descriptions and the differences in keywords between different positions.

Machine Learning

After completing keyword analysis and data preprocessing, I built and trained a variety of different machine learning models to evaluate their performance on job classification and recommendation tasks, various machine learning models including random forests, support vector machines, and neural networks were trained to predict job matches based on user skills (Dave et al., 2018; Paparrizos et al., 2011). Advanced NLP techniques such as topic analysis and word embedding were also employed to enhance the recommendation accuracy (Yang & Rim, 2014; Lima et al., 2015).

I chose the following models to try and evaluate to cover different machine learning strategies:

Random Forest

(1) TF-IDF vectorization: Use TfidfVectorizer to vectorize the cleaned job descriptions and select up to 5,000 most important word features.
(2) Label encoding: Use LabelEncoder to convert the job title into a numeric value to provide a suitable output format for the classification model.
(3) Data partitioning: Split the dataset into a training set and a test set at a ratio of 80% and 20%, ensuring that there is enough data for training without compromising the generalization ability of the model.
(4) Initialize and train random forest: Initialize RandomForestClassifier and set 100 trees.
(5) Output classification report: Generate and print a classification report, providing precision, recall, and F1 scores for each job category, and analyzing the performance of the model in detail.
(6) Plot confusion matrix: Display the confusion matrix through a heat map to intuitively show the prediction accuracy of the model on each category.

```
# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['cleaned_description'])

# Creating a label mapping
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize the Random Forest Model
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the model
clf.fit(X_train, y_train)

# predict
y_pred = clf.predict(X_test)

# Evaluating the Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_, zero_division=0))

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()
```

Code 13: Data partitioning and building random forest models

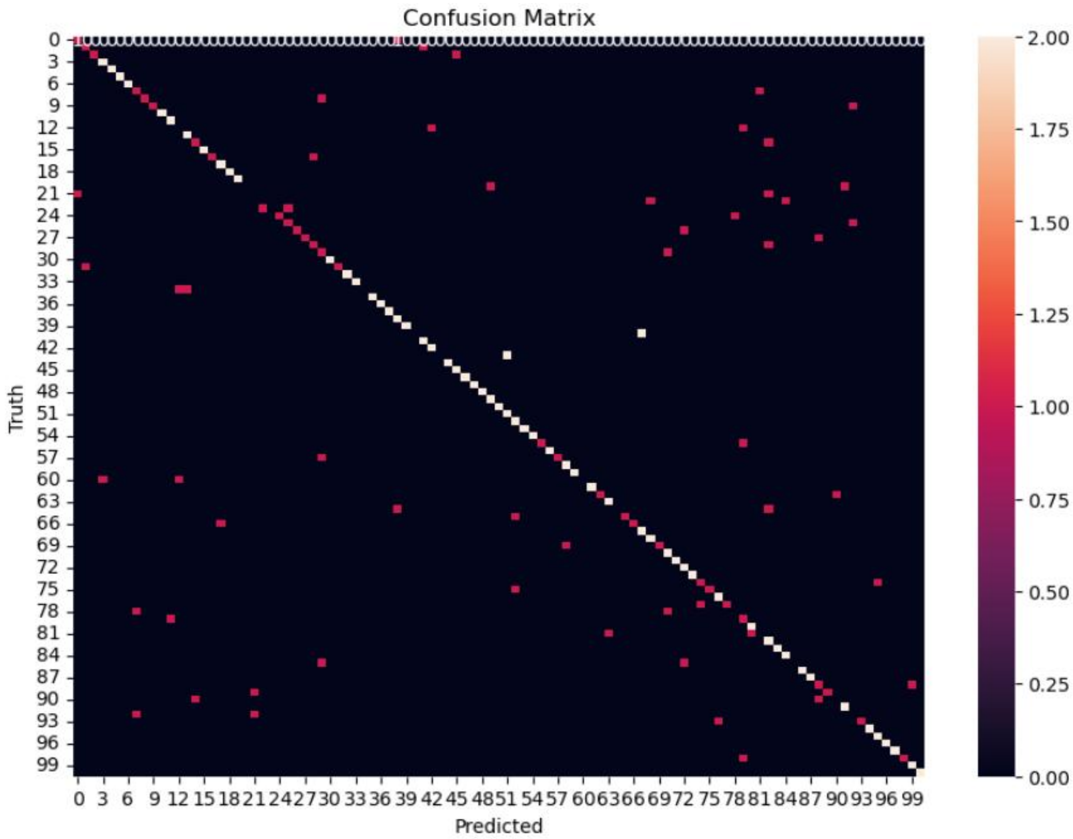
Model Performance Overview Accuracy: The overall accuracy of the model on the test set reached 70.79%, which shows that the model is able to predict the job category with high accuracy.

	precision	recall	f1-score	support
accountant	0.50	0.50	0.50	2
aerospace engineer	0.50	0.50	0.50	2
aircraft mechanic	1.00	0.50	0.67	2
architect	0.67	1.00	0.80	2
automotive technician	1.00	1.00	1.00	2
barber	1.00	1.00	1.00	2
bartender	1.00	1.00	1.00	2
biomedical engineer	0.33	0.50	0.40	2
bus driver	1.00	0.50	0.67	2
business analyst	1.00	0.50	0.67	2
carpenter	1.00	1.00	1.00	2
chef	0.67	1.00	0.80	2
chemical engineer	0.00	0.00	0.00	2
civil engineer	0.67	1.00	0.80	2

Figure 6: Partial model performance (full version in reference)

Category Performance: Some job titles such as "barber", "biomedical engineer", and "data scientist" show high precision and recall, while jobs such as "data analyst", "systems administrator" show lower performance indicators. This may be due to the large language overlap between job descriptions.

Confusion Matrix: Through the visualization of the confusion matrix, we can see that the predictions for most job categories are concentrated on the diagonal, indicating that the model's predictions for most jobs are accurate. However, there are also some cases of misclassification, especially for some job categories that are similar to each other.




```
The most suitable job for the skill set 'machine learning python data analysis' is: restaurant manager
```

Figure 7: Confusion matrix and prediction results

When using the skill set "machine learning python data analysis" for prediction, the model recommends the job "restaurant manager". This significant error may be caused by data imbalance.

The random forest model provides us with a reliable framework to build a job recommendation system. Its accuracy and interpretability make it a powerful tool for solving such tasks. Although the performance in the test did not meet expectations, future improvements and optimizations will further improve the performance of the system and make it more effective in practical applications.

Support Vector Machine

I also tried the support vector machine model in my job recommendation system

(1) TF-IDF vectorization, model prediction and evaluation, data partitioning, and label encoding are the same as before.

(2) Model training: The model was trained on the split training data, using all default parameter configurations, and no additional parameter optimization was performed.

```
# Initialize the SVM model
svm_model = SVC(kernel='linear', probability=True)

# Training the model
svm_model.fit(X_train, y_train)

# predict
y_pred = svm_model.predict(X_test)

# Evaluating the Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Code 14: Building SVM models

The SVM model achieved an accuracy of 65.35% on the test set, showing that the model has a certain ability to classify the job data.

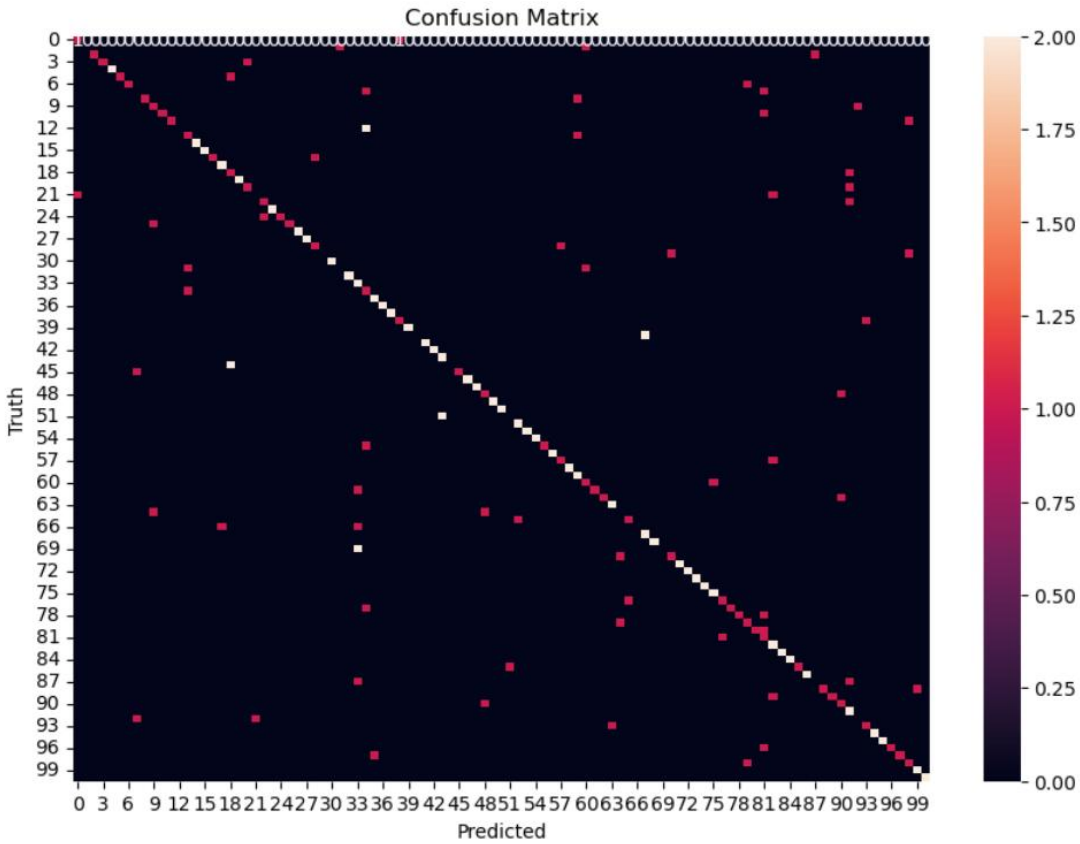
Classification Performance: We noticed that some jobs such as "data engineer" and "machine learning engineer" performed well, while categories such as "data analyst" and "systems administrator" performed poorly.

Accuracy: 65.35%

	precision	recall	f1-score	support
accountant	0.50	0.50	0.50	2
aerospace engineer	0.00	0.00	0.00	2
aircraft mechanic	1.00	0.50	0.67	2
architect	1.00	0.50	0.67	2
automotive technician	1.00	1.00	1.00	2
barber	1.00	0.50	0.67	2
bartender	1.00	0.50	0.67	2
biomedical engineer	0.00	0.00	0.00	2
bus driver	1.00	0.50	0.67	2
business analyst	0.33	0.50	0.40	2
carpenter	1.00	0.50	0.67	2

Figure 8: Partial model performance (full version in reference)

Confusion Matrix: From the confusion matrix, we can see that although many jobs are correctly classified, many jobs are misclassified. This shows that the model still has challenges in distinguishing jobs.




```
The most suitable job for the skill set 'machine learning python data analysis' is: machine learning engineer
```

Figure 9: Confusion matrix and prediction results

Job Prediction: For the skill set "machine learning python data analysis", the model successfully predicted "machine learning engineer" as the most suitable job.

In summary, the SVM model proved to be an effective method in our job recommendation system, although it still has room for improvement in some job classifications. By continuing to optimize and adjust, we hope to further improve the accuracy and reliability of the model.

XGBoost

In this project, I also used the XGBoost model to handle the job recommendation task.

(1) TF-IDF vectorization, model prediction and evaluation, data partitioning, and label encoding are the same as before.

(2) Initialize the XGBoost model: Use XGBClassifier to configure the target and related parameters for multi-class classification.

(3) Model training: Train the XGBoost model on the training set and monitor the performance in real time on the test set through eval_set.

```
# Initializing the XGBoost Model
clf = xgb.XGBClassifier(objective='multi:softprob', num_class=len(label_encoder.classes_), random_state=42)

# Training the model
clf.fit(X_train, y_train, eval_metric=["mlogloss", "merror"], eval_set=[(X_test, y_test)], verbose=True)

# predict
y_pred = clf.predict(X_test)
```

Code 15: Building XGBoost models

The overall accuracy of the model on the test set is 65.84%, which shows the ability of XGBoost to handle the data, although slightly lower than the random forest model.

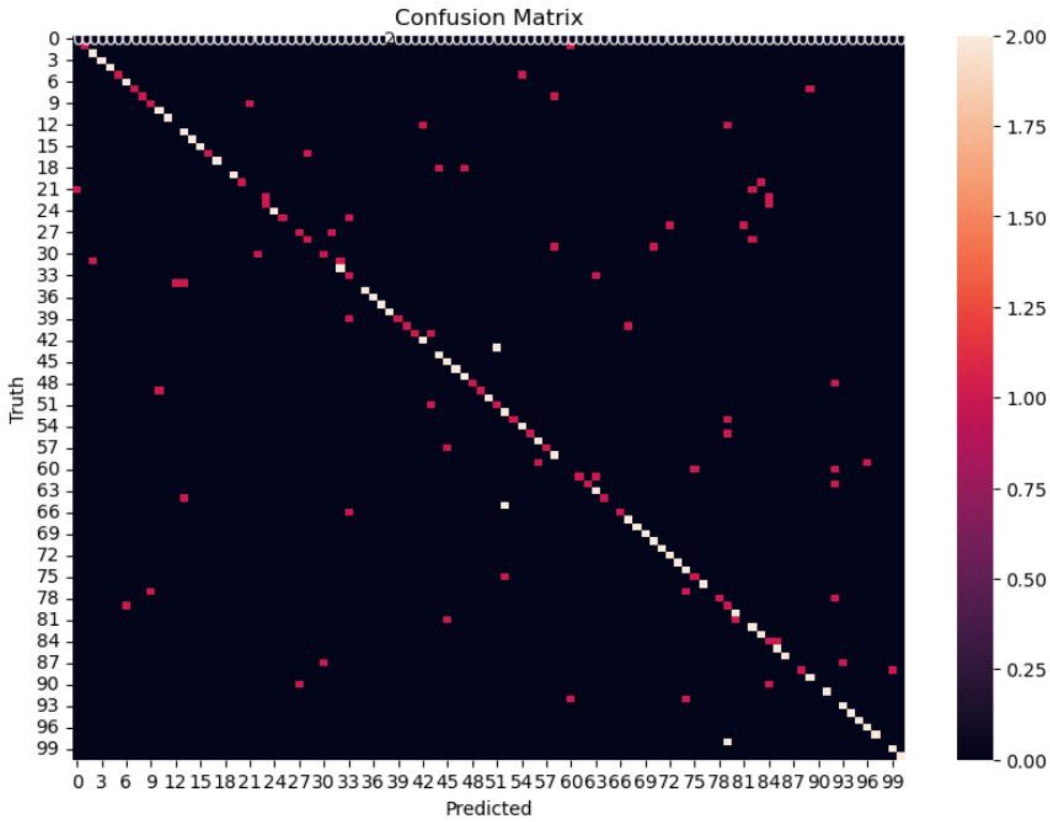
```
Accuracy: 65.84%
                         precision    recall  f1-score   support

            accountant       0.00      0.00      0.00         2
    aerospace engineer       1.00      0.50      0.67         2
     aircraft mechanic       0.67      1.00      0.80         2
             architect       1.00      1.00      1.00         2
 automotive technician       1.00      1.00      1.00         2
                barber       1.00      0.50      0.67         2
             bartender       0.67      1.00      0.80         2
   biomedical engineer       1.00      0.50      0.67         2
            bus driver       1.00      0.50      0.67         2
      business analyst       0.50      0.50      0.50         2
             carpenter       0.67      1.00      0.80         2
```

Figure 10: Partial model performance (full version in reference)

Category Performance: Some job categories such as "architect" and "data scientist" show high precision and recall, while categories such as "data analyst" and "systems administrator" perform poorly. This may be due to insufficient coverage of keywords in these job descriptions or the model's failure to fully capture their characteristics.

Confusion Matrix Analysis: The points on the diagonal represent instances correctly classified by the model, and the darker the color, the greater the number. It can be seen that for many job titles, the model can accurately predict. Some jobs may be misclassified because of similar descriptions.




```
The most suitable job for the skill set 'machine learning python data analysis' is: data engineer
```

Figure 11: Confusion matrix and prediction results

Based on the provided skill set "machine learning python data analysis", the most suitable job predicted by the model is "data engineer", which is a reasonable prediction that is highly correlated with the provided skills. This shows the potential of the model in understanding the relationship between skills and job titles.

In general, the introduction of the XGBoost model has proven to be effective, and its high configurability and powerful performance make it an ideal choice for solving complex classification problems.

Neural Network

Finally, I tried using a neural network model, built on the TensorFlow framework, using a multi-layer perceptron (MLP) architecture to provide job matching for a given skill set.

(1) TF-IDF vectorization, model prediction and evaluation, data partitioning are the same as before.

(2) Model architecture: The model consists of three hidden layers, each using a ReLU activation function, and each layer is followed by a Dropout layer to reduce overfitting.

(3) Label processing: The job labels are processed by the Label Encoder and converted to one-hot encoding, which is applied to the output layer of the neural network.

```python
# Convert labels to one-hot encoding
y = to_categorical(y)
```

```python
# Initialize neural network model
ann = tf.keras.models.Sequential()

# Add input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units=512, activation='relu', input_shape=(X_train.shape[1],)))
ann.add(tf.keras.layers.Dropout(0.5))

# Add second hidden layer
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add third hidden layer
ann.add(tf.keras.layers.Dense(units=128, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add output layer
ann.add(tf.keras.layers.Dense(units=y_train.shape[1], activation='softmax'))

# Compile model
ann.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Code 16: Building neural network structure

On the test set, the neural network model achieved an accuracy of 39.11%. This result is relatively low, indicating that the model may need further optimization or tuning.

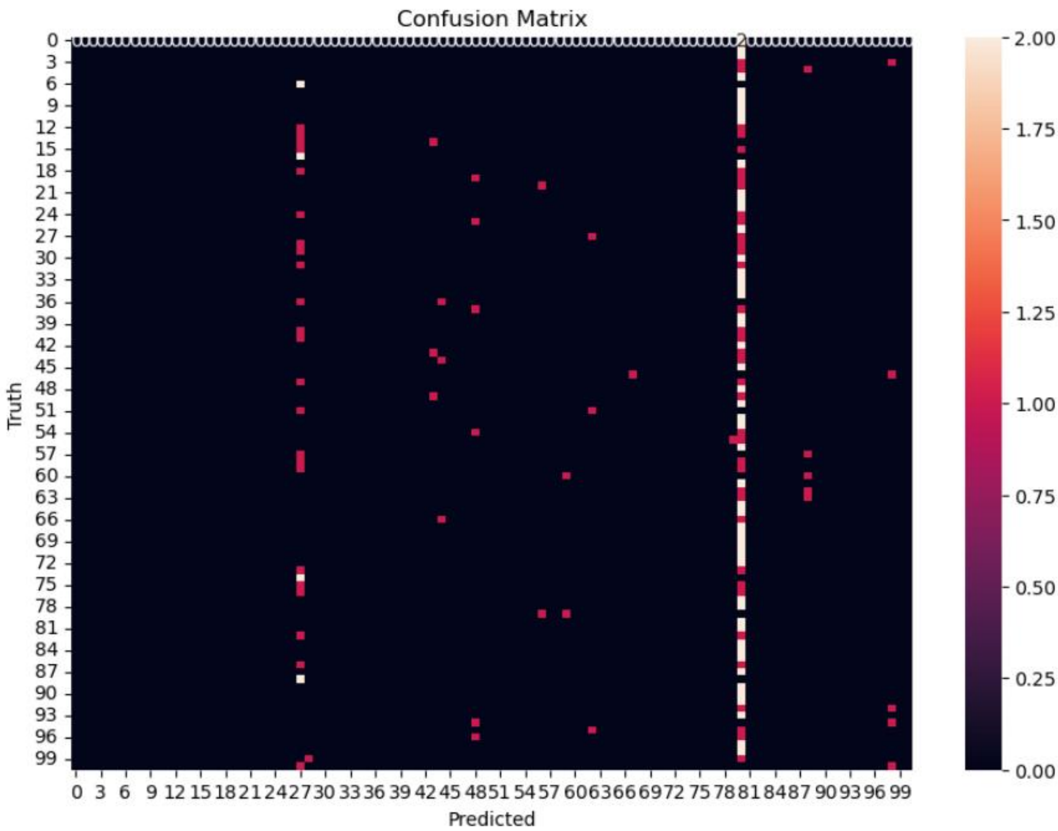
Classification performance: Many jobs have low recall and precision, which may be due to class imbalance or the model is not sensitive to certain types of data.

Accuracy: 39.11%

	precision	recall	f1-score	support
accountant	1.00	0.50	0.67	2
aerospace engineer	0.00	0.00	0.00	2
aircraft mechanic	0.00	0.00	0.00	2
architect	0.00	0.00	0.00	2
automotive technician	1.00	1.00	1.00	2
barber	0.67	1.00	0.80	2
bartender	0.33	0.50	0.40	2
biomedical engineer	0.17	0.50	0.25	2
bus driver	1.00	1.00	1.00	2
business analyst	0.00	0.00	0.00	2

Figure 12: Partial model performance (full version in reference)

Confusion matrix: The confusion matrix shows the performance of the model on each job classification. It can be seen that while some jobs are correctly classified, most jobs have a high misclassification rate.




```
1/1 [==============================] - 0s 17ms/step
The most suitable job for the skill set 'machine learning python data analysis' is: mechanic
```

Figure 13: Confusion matrix and prediction results

Job prediction: Although the overall accuracy is not high, the neural network successfully predicts jobs for specific skill sets, such as "machine learning engineer" for the "machine learning python data analysis" skill set.

After seeing the poor results, I thought it might be a TF-IDF problem, so I tried using Word2Vec technology, but got even worse results.

```
# Train the Word2Vec model
sentences = data['cleaned_description'].tolist()
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Convert each job description to a vector
def vectorize_text(text, model, vector_size):
    words = [word for word in text if word in model.wv.index_to_key]
    if len(words) == 0:
        return np.zeros(vector_size)
    word_vectors = model.wv[words]
    return np.mean(word_vectors, axis=0)

data['vectorized_description'] = data['cleaned_description'].apply(lambda x: vectorize_text(x, word2vec_model, 100))

# Build the feature matrix
X = np.stack(data['vectorized_description'].values)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])
```

Code 17: Using Word2Vec

The model' s accuracy is 1.98%, which is almost completely wrong.

Accuracy: 1.98%

	precision	recall	f1-score	support
accountant	0.00	0.00	0.00	2
aerospace engineer	0.00	0.00	0.00	2
aircraft mechanic	0.00	0.00	0.00	2
architect	0.00	0.00	0.00	2
automotive technician	0.00	0.00	0.00	2
barber	0.00	0.00	0.00	2
bartender	0.00	0.00	0.00	2
biomedical engineer	0.00	0.00	0.00	2
bus driver	0.00	0.00	0.00	2

Figure 14: Partial model performance

This may be because Word2Vec relies on a large-scale corpus to learn word vectors. If the corpus is not large enough or of low quality, the generated word vectors are not sufficient to support accurate job classification.

To sum up, neural networks may be a good solution, but due to the small amount of data and the low quality of data, it is impossible to achieve a high accuracy.

Conclusion

In this project, we developed a job recommendation system based on natural language processing technology. By crawling job advertisements on the Indeed website, we used a variety of machine learning models to try to accurately match user skills with job requirements in the market.

Through practice, we found that although machine learning-based models showed varying degrees of success in predicting jobs, each model also showed its specific advantages and limitations. The performance of each model was evaluated based on precision, recall, and F1 scores, demonstrating varying degrees of success in job classification tasks (Nigam et al., 2019). For example, the random forest model showed high accuracy in the job classification task, while the support vector machine (SVM) and XGBoost, although slightly less accurate, still showed good classification ability during testing. In addition, although the performance of neural networks was not optimal in the experiment, its potential should not be underestimated with the support of better quality and large-scale data.

From the experimental results, the models differed in precision and recall, which may be related to differences in data preprocessing, feature selection, and model parameter adjustment. The selection and optimization of each method needs to be adjusted according to the specific application scenario and data characteristics to achieve the best results.

In addition, some challenges in the project also suggest the direction of improvement in future work. For example, improving the text preprocessing process, optimizing more advanced word vector models such as Word2Vec, or exploring more complex model structures may improve the performance of the model.

In summary, this project provides a practical job recommendation tool that demonstrates the potential of using NLP and machine learning techniques to process complex text data. Through continuous optimization and expansion, this technology can help more job seekers find matching positions in the highly competitive job market, so as to better plan and develop their careers.

Reference

Article

[1]Abel, F., Benczúr, A., Kohlsdorf, D., Larson, M., & Pálovics, R. (2016). RecSys Challenge 2016: Job Recommendations. Proceedings of the 10th ACM Conference on Recommender Systems, 425–426. https://doi.org/10.1145/2959100.2959207

[2]Dai, T., Zhu, L., Wang, Y., Zhang, H., Cai, X., & Zheng, Y. (2019). Joint Model Feature Regression and Topic Learning for Global Citation Recommendation. IEEE Access, 7, 1706–1720. https://doi.org/10.1109/ACCESS.2018.2884981

[3]Dave, V. S., Zhang, B., Al Hasan, M., AlJadda, K., & Korayem, M. (2018). A Combined Representation Learning Approach for Better Job and Skill Recommendation. Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 1997–2005. https://doi.org/10.1145/3269206.3272023

[4]Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. Applied Mathematics and Computation, 270, 756–767. https://doi.org/10.1016/j.amc.2015.08.059

[5]Nigam, A., Roy, A., Singh, H., & Waila, H. (2019). Job Recommendation through Progression of Job Selection. 2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS), 212–216. https://doi.org/10.1109/CCIS48116.2019.9073723

[6]Paparrizos, I., Cambazoglu, B. B., & Gionis, A. (2011). Machine learned job recommendation. Proceedings of the Fifth ACM Conference on Recommender Systems, 325–328. https://doi.org/10.1145/2043932.2043994

[7]Vargas-Calderón, V., & Camargo, J. E. (2019). Characterization of citizens using word2vec and latent topic analysis in a large set of tweets. Cities, 92, 187–196. https://doi.org/10.1016/j.cities.2019.03.019

[8]Yang, M.-C., & Rim, H.-C. (2014). Identifying interesting Twitter contents using topical analysis. Expert Systems with Applications, 41(9), 4330–4336. https://doi.org/10.1016/j.eswa.2013.12.051

Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.firefox_binary import FirefoxBinary
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import pandas as pd
#%%
# Set the Firefox driver path and Firefox executable file path
driver_path = './geckodriver'
firefox_binary_path = 'C:/Program Files/Mozilla Firefox/firefox.exe'
#%%
def scrape_indeed(driver_path, firefox_binary_path, job_query, num_jobs_to_scrape):
    # Setting up Firefox
    options = webdriver.FirefoxOptions()
    binary = FirefoxBinary(firefox_binary_path)
    options.binary = binary

    # Setting the crawl URL
    url = f'https://www.indeed.com/jobs?q={job_query}&l='
    driver.get(url)
    time.sleep(5)   # Waiting for the page to load

    # Initialize the job information list
    job_data = []
    job_count = 0

    # Creating a WebDriver Instance
```

```
        driver = webdriver.Firefox(executable_path=driver_path, options=options)
        wait = WebDriverWait(driver, 10)

        while job_count < num_jobs_to_scrape:
            # Capture job cards
            job_cards = driver.find_elements(By.CSS_SELECTOR,
'.job_seen_beacon')

            for job_card in job_cards:
                if job_count >= num_jobs_to_scrape:
                    break
                try:
                    # Extract job title, company name, location, and posting date
                    title = job_card.find_element(By.CSS_SELECTOR,
'h2.jobTitle span').text
                    company = job_card.find_element(By.CSS_SELECTOR,
'span[data-testid="company-name"]').text
                    location = job_card.find_element(By.CSS_SELECTOR,
'div[data-testid="text-location"]').text
                    date_posted = job_card.find_element(By.CSS_SELECTOR,
'span[data-testid="myJobsStateDate"]').text

                    # Click on the job card to view the job description
                    job_card.find_element(By.CSS_SELECTOR, 'a').click()  #
Make sure to click on the link to view the detailed description
                    time.sleep(3)  # Waiting for job descriptions to load

                    # Wait for the job description to load
                    job_description_element = wait.until(
```

```
            EC.presence_of_element_located((By.CSS_SELECTOR,
'#jobDescriptionText'))
        )
        job_description = job_description_element.text

        # Extract salary information
        try:
            salary = driver.find_element(By.CSS_SELECTOR,
'span.css-19j1a75').text
        except:
            salary = 'N/A'

        # Extract job requirements and required skills
        requirements, skills =
extract_requirements_and_skills(job_description)

        # Add the captured information to the list
        job_data.append({
            'Title': title,
            'Company': company,
            'Location': location,
            'Date Posted': date_posted,
            'Description': job_description,
            'Salary': salary,
            'Requirements': requirements,
            'Skills': skills
        })

        job_count += 1
```

```
                    print(f"Collected job data: {title} at {company} in {location} ({job_count}/{num_jobs_to_scrape})")

                    driver.back()
                    time.sleep(3)  # Waiting to return to job list
                except Exception as e:
                    print(f"Error: {e}")
                    continue

            # Try to find and click the "Next Page" button
            try:
                next_button = driver.find_element(By.CSS_SELECTOR, 'a[aria-label="Next"]')
                next_button.click()
                time.sleep(5)  # Waiting for a new page to load
            except Exception as e:
                print("No more pages to load.")
                break

        # Close the browser
        driver.quit()

        return job_data
    #%%
    def extract_requirements_and_skills(description):
        # Assume that the job requirements and required skills are contained between keywords
        requirements_keywords = ['Requirements', 'Qualifications', 'Must have']
        skills_keywords = ['Skills', 'Proficient in', 'Experience with']
```

```
    requirements = []
    skills = []

    lines = description.split('\n')
    capture_requirements = False
    capture_skills = False

    for line in lines:
        if any(keyword in line for keyword in requirements_keywords):
            capture_requirements = True
            capture_skills = False
        elif any(keyword in line for keyword in skills_keywords):
            capture_skills = True
            capture_requirements = False
        elif line.strip() == '':
            capture_requirements = False
            capture_skills = False

        if capture_requirements:
            requirements.append(line.strip())
        if capture_skills:
            skills.append(line.strip())

    return ' '.join(requirements), ' '.join(skills)

#%%
def main():
    num_jobs_to_scrape = 10   # Set the number of positions to crawl for each
position type
```

```
# Define a list of job types to crawl
job_queries = [
    'data+scientist', 'software+engineer', 'project+manager',
'product+manager',
    'business+analyst', 'web+developer', 'graphic+designer',
'network+engineer',
    'systems+administrator', 'database+administrator', 'quality+assurance',
    'technical+support', 'data+analyst', 'security+analyst', 'data+engineer',
    'machine+learning+engineer', 'devops+engineer', 'cloud+architect',
'it+manager',
    'seo+specialist', 'digital+marketer', 'content+writer', 'sales+manager',
    'financial+analyst', 'accountant', 'hr+manager', 'marketing+manager',
    'customer+service+representative', 'operations+manager',
'logistics+coordinator',
    'procurement+manager', 'supply+chain+analyst', 'legal+assistant',
'paralegal',
    'nurse', 'medical+assistant', 'pharmacist', 'physical+therapist',
    'research+scientist', 'lab+technician', 'teacher', 'school+principal',
    'instructional+designer', 'education+consultant', 'library+assistant',
    'counselor', 'psychologist', 'social+worker', 'therapist',
    'engineer', 'mechanical+engineer', 'civil+engineer', 'electrical+engineer',
    'chemical+engineer', 'environmental+engineer', 'aerospace+engineer',
    'biomedical+engineer', 'architect', 'urban+planner',
    'construction+manager', 'electrician', 'plumber', 'carpenter',
    'welder', 'machinist', 'automotive+technician', 'hvac+technician',
    'chef', 'restaurant+manager', 'bartender', 'waiter',
    'event+planner', 'travel+agent', 'tour+guide', 'flight+attendant',
    'pilot', 'aircraft+mechanic', 'bus+driver', 'truck+driver',
    'mechanic', 'dispatcher', 'safety+manager', 'safety+specialist',
    'janitor', 'custodian', 'groundskeeper', 'landscaper',
```

```
            'security+guard', 'firefighter', 'police+officer', 'detective',
            'correctional+officer', 'emergency+medical+technician', 'paramedic',
            'fitness+trainer', 'personal+trainer', 'massage+therapist',
            'esthetician', 'hair+stylist', 'cosmetologist', 'barber'
        ]

        all_job_data = []

        # Traverse the list of job types and crawl the job information of each job type
        for job_query in job_queries:
            print(f"Scraping jobs for: {job_query.replace('+', ' ')}")
            job_data = scrape_indeed(driver_path, firefox_binary_path, job_query, num_jobs_to_scrape)
            all_job_data.extend(job_data)

        # Save all job data to a CSV file
        df = pd.DataFrame(all_job_data)
        df.to_csv('indeed_all_job_data.csv', index=False)
    #%%
    if __name__ == "__main__":
        main()
    #%%
    import pandas as pd
    import re
    import nltk
    from nltk.corpus import stopwords
    from nltk.stem import WordNetLemmatizer
    import seaborn as sns
    import matplotlib.pyplot as plt
```

```
# Read the CSV file and specify the encoding as 'ISO-8859-1'
data = pd.read_csv('indeed_all_job_data.csv', encoding='ISO-8859-1')

# Make sure have downloaded the necessary NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Cleanup Function
def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Convert to lowercase
    text = text.lower()
    # Participle
    words = text.split()
    # Stop word removal and lemmatization
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    return ' '.join(words)

# Convert all columns to string type and handle missing values
data['Description'] = data['Description'].astype(str).fillna('')
```

```
# Applying cleanup functions
data['cleaned_description'] = data['Description'].apply(clean_text)

# View the cleaned data
print(data[['Title', 'cleaned_description']].head(11))


#%%
# Standardized occupational titles
job_queries = [
    'data+scientist', 'software+engineer', 'project+manager', 'product+manager',
    'business+analyst', 'web+developer', 'graphic+designer', 'network+engineer',
    'systems+administrator', 'database+administrator', 'quality+assurance',
    'technical+support', 'data+analyst', 'security+analyst', 'data+engineer',
    'machine+learning+engineer', 'devops+engineer', 'cloud+architect',
'it+manager',
    'seo+specialist', 'digital+marketer', 'content+writer', 'sales+manager',
    'financial+analyst', 'accountant', 'hr+manager', 'marketing+manager',
    'customer+service+representative', 'operations+manager',
'logistics+coordinator',
    'procurement+manager', 'supply+chain+analyst', 'legal+assistant', 'paralegal',
    'nurse', 'medical+assistant', 'pharmacist', 'physical+therapist',
    'research+scientist', 'lab+technician', 'teacher', 'school+principal',
    'instructional+designer', 'education+consultant', 'library+assistant',
    'counselor', 'psychologist', 'social+worker', 'therapist',
    'engineer', 'mechanical+engineer', 'civil+engineer', 'electrical+engineer',
    'chemical+engineer', 'environmental+engineer', 'aerospace+engineer',
    'biomedical+engineer', 'architect', 'urban+planner',
    'construction+manager', 'electrician', 'plumber', 'carpenter',
    'welder', 'machinist', 'automotive+technician', 'hvac+technician',
```

```
    'chef', 'restaurant+manager', 'bartender', 'waiter',
    'event+planner', 'travel+agent', 'tour+guide', 'flight+attendant',
    'pilot', 'aircraft+mechanic', 'bus+driver', 'truck+driver',
    'mechanic', 'dispatcher', 'safety+manager', 'safety+specialist',
    'janitor', 'custodian', 'groundskeeper', 'landscaper',
    'security+guard', 'firefighter', 'police+officer', 'detective',
    'correctional+officer', 'emergency+medical+technician', 'paramedic',
    'fitness+trainer', 'personal+trainer', 'massage+therapist',
    'esthetician', 'hair+stylist', 'cosmetologist', 'barber'
]

# 10 for each profession, standardize the profession query list
standardized_titles = []
for job_query in job_queries:
    standardized_title = job_query.replace('+', ' ')
    standardized_titles.extend([standardized_title] * 10)

# Add standardized occupation titles to the dataset
data['standardized_title'] = standardized_titles

# View the cleaned data
print(data[['Title', 'cleaned_description', 'standardized_title']].head(20))

#%%
from sklearn.feature_extraction.text import TfidfVectorizer

# Extract keywords using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=100)
tfidf_matrix = tfidf_vectorizer.fit_transform(data['cleaned_description'])
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
```

```
tfidf_sum = tfidf_matrix.sum(axis=0)
tfidf_freq = [(word, tfidf_sum[0, idx]) for word, idx in zip(tfidf_feature_names, range(tfidf_sum.shape[1]))]
tfidf_freq = sorted(tfidf_freq, key=lambda x: x[1], reverse=True)

# Convert keywords and frequencies into DataFrame
tfidf_df = pd.DataFrame(tfidf_freq, columns=['Keyword', 'Frequency'])

# Visualize the top 20 keywords
plt.figure(figsize=(14, 7))
sns.barplot(x='Frequency', y='Keyword', data=tfidf_df.head(20))
plt.title('Top 20 Keywords in Job Descriptions')
plt.xlabel('Frequency')
plt.ylabel('Keyword')
plt.show()

#%%
from wordcloud import WordCloud

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(dict(tfidf_freq))
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Job Descriptions')
plt.show()

#%%
compare_jobs = ['data scientist', 'barber']
```

```
# Calculate and visualize job keywords
def analyze_job_keywords(job_title):
    job_data = data[data['standardized_title'] == job_title]
    tfidf_vectorizer = TfidfVectorizer(max_features=100)
    job_tfidf_matrix = tfidf_vectorizer.fit_transform(job_data['cleaned_description'])
    job_tfidf_sum = job_tfidf_matrix.sum(axis=0)
    job_tfidf_freq = [(word, job_tfidf_sum[0, idx]) for word, idx in zip(tfidf_vectorizer.get_feature_names_out(), range(job_tfidf_sum.shape[1]))]
    job_tfidf_freq = sorted(job_tfidf_freq, key=lambda x: x[1], reverse=True)
    job_tfidf_df = pd.DataFrame(job_tfidf_freq, columns=['Keyword', 'Frequency'])
    return job_tfidf_df

# Calculate keywords for two positions separately
data_scientist_keywords = analyze_job_keywords('data scientist')
barber_keywords = analyze_job_keywords('barber')

# Visualize the top 20 keywords comparison
fig, axes = plt.subplots(1, 2, figsize=(20, 10))
sns.barplot(x='Frequency', y='Keyword', data=data_scientist_keywords.head(20), ax=axes[0])
axes[0].set_title('Top 20 Keywords in Data Scientist Job Descriptions')
axes[0].set_xlabel('Frequency')
axes[0].set_ylabel('Keyword')

sns.barplot(x='Frequency', y='Keyword', data=barber_keywords.head(20), ax=axes[1])
axes[1].set_title('Top 20 Keywords in Barber Job Descriptions')
```

```
axes[1].set_xlabel('Frequency')
axes[1].set_ylabel('Keyword')

plt.tight_layout()
plt.show()
#%%
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['cleaned_description'])

# Creating a label mapping
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initialize the Random Forest Model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Training the model
clf.fit(X_train, y_train)

# predict
y_pred = clf.predict(X_test)

# Evaluating the Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_,
zero_division=0))

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()

# Prediction Function
def predict_job(skill_set):
    cleaned_skill_set = clean_text(skill_set)
    skill_vector = tfidf_vectorizer.transform([cleaned_skill_set])
    prediction = clf.predict(skill_vector)
    predicted_index = prediction[0]
```

```python
        return label_encoder.inverse_transform([predicted_index])[0]

# Sample Skill Sets
skill_set = "machine learning python data analysis"
predicted_job = predict_job(skill_set)
print(f"The most suitable job for the skill set '{skill_set}' is: {predicted_job}")
#%%
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['cleaned_description'])

# Creating a label mapping
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Initializing the XGBoost Model
clf = xgb.XGBClassifier(objective='multi:softprob', num_class=len(label_encoder.classes_), random_state=42)

# Training the model
```

```
clf.fit(X_train, y_train, eval_metric=["mlogloss", "merror"], eval_set=[(X_test, y_test)], verbose=True)

# predict
y_pred = clf.predict(X_test)

# Evaluating the Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_, zero_division=0))

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()

# Prediction Function
def predict_job(skill_set):
    cleaned_skill_set = clean_text(skill_set)
    skill_vector = tfidf_vectorizer.transform([cleaned_skill_set])
    prediction = clf.predict(skill_vector)
    predicted_index = prediction[0]
    return label_encoder.inverse_transform([predicted_index])[0]
```

```
# Sample Skill Sets
skill_set = "machine learning python data analysis"
predicted_job = predict_job(skill_set)
print(f"The most suitable job for the skill set '{skill_set}' is: {predicted_job}")

#%%
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer



# Initialize the SVM model
svm_model = SVC(kernel='linear', probability=True)

# Training the model
svm_model.fit(X_train, y_train)

# predict
```

```
y_pred = svm_model.predict(X_test)

# Evaluating the Model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_,
zero_division=0))

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()

# Prediction Function
def predict_job(skill_set):
    cleaned_skill_set = clean_text(skill_set)
    skill_vector = tfidf_vectorizer.transform([cleaned_skill_set])
    prediction = svm_model.predict(skill_vector)
    predicted_index = prediction[0]
    return label_encoder.inverse_transform([predicted_index])[0]

# Sample Skill Sets
skill_set = "machine learning python data analysis"
predicted_job = predict_job(skill_set)
```

```
print(f"The most suitable job for the skill set '{skill_set}' is: {predicted_job}")

#%%
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.utils import to_categorical

# Feature extraction using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X = tfidf_vectorizer.fit_transform(data['cleaned_description'])

# Create label mapping
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])

# Convert labels to one-hot encoding
y = to_categorical(y)
```

```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X.toarray(), y, test_size=0.2, random_state=42, stratify=y)

# Feature scaling
sc = StandardScaler(with_mean=False)
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Initialize neural network model
ann = tf.keras.models.Sequential()

# Add input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units=512, activation='relu', input_shape=(X_train.shape[1],)))
ann.add(tf.keras.layers.Dropout(0.5))

# Add second hidden layer
ann.add(tf.keras.layers.Dense(units=256, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add third hidden layer
ann.add(tf.keras.layers.Dense(units=128, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add output layer
ann.add(tf.keras.layers.Dense(units=y_train.shape[1], activation='softmax'))

# Compile model
```

```
ann.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
history = ann.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

# Predict
y_pred = ann.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Evaluate model
accuracy = accuracy_score(y_test_classes, y_pred_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test_classes, y_pred_classes, target_names=label_encoder.classes_, zero_division=0))

# Compute confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()

# Prediction function
```

```
def predict_job(skill_set):
    cleaned_skill_set = clean_text(skill_set)
    skill_vector = tfidf_vectorizer.transform([cleaned_skill_set]).toarray()
    skill_vector = sc.transform(skill_vector)
    prediction = ann.predict(skill_vector)
    predicted_index = np.argmax(prediction, axis=1)[0]
    return label_encoder.inverse_transform([predicted_index])[0]

# Sample skill set
skill_set = "machine learning python data analysis"
predicted_job = predict_job(skill_set)
print(f"The most suitable job for the skill set '{skill_set}' is: {predicted_job}")

#%%
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from gensim.models import Word2Vec
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
# Train the Word2Vec model
sentences = data['cleaned_description'].tolist()
word2vec_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# Convert each job description to a vector
def vectorize_text(text, model, vector_size):
    words = [word for word in text if word in model.wv.index_to_key]
    if len(words) == 0:
        return np.zeros(vector_size)
    word_vectors = model.wv[words]
    return np.mean(word_vectors, axis=0)

data['vectorized_description'] = data['cleaned_description'].apply(lambda x: vectorize_text(x, word2vec_model, 100))

# Build the feature matrix
X = np.stack(data['vectorized_description'].values)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(data['standardized_title'])

# Convert labels to one-hot encoding
y = tf.keras.utils.to_categorical(y)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Feature scaling
```

```
sc = StandardScaler(with_mean=False)    # Set with_mean to False to avoid issues with sparse matrices
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Initialize the neural network model
ann = tf.keras.models.Sequential()

# Add input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units=128, activation='relu', input_shape=(X_train.shape[1],)))
ann.add(tf.keras.layers.Dropout(0.5))

# Add second hidden layer
ann.add(tf.keras.layers.Dense(units=64, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add third hidden layer
ann.add(tf.keras.layers.Dense(units=32, activation='relu'))
ann.add(tf.keras.layers.Dropout(0.5))

# Add output layer
ann.add(tf.keras.layers.Dense(units=y_train.shape[1], activation='softmax'))

# Compile the model
ann.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
```

```
history = ann.fit(X_train, y_train, epochs=200, batch_size=32, validation_split=0.2, verbose=1)

# Predict
y_pred = ann.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Evaluate the model
accuracy = accuracy_score(y_test_classes, y_pred_classes)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification
print(classification_report(y_test_classes, y_pred_classes, target_names=label_encoder.classes_, zero_division=0))

# Compute confusion matrix
cm = confusion_matrix(y_test_classes, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix')
plt.show()

# Prediction function
def predict_job(skill_set, model, scaler, label_enc):
    cleaned_skill_set = clean_text(skill_set)
    skill_vector = vectorize_text(cleaned_skill_set, word2vec_model, 100).reshape(1, -1)
```

```
    skill_vector = scaler.transform(skill_vector)
    prediction = model.predict(skill_vector)
    predicted_index = np.argmax(prediction, axis=1)[0]
    return label_enc.inverse_transform([predicted_index])[0]

# Sample skill set
skill_set = "machine learning python data analysis"
predicted_job = predict_job(skill_set, ann, sc, label_encoder)
print(f"The most suitable job for the skill set '{skill_set}' is: {predicted_job}")
```

Model result

Random Forest

Accuracy: 70.79%

	precision	recall	f1-score	support
accountant	0.50	0.50	0.50	2
aerospace engineer	0.50	0.50	0.50	2
aircraft mechanic	1.00	0.50	0.67	2
architect	0.67	1.00	0.80	2
automotive technician	1.00	1.00	1.00	2
barber	1.00	1.00	1.00	2
bartender	1.00	1.00	1.00	2
biomedical engineer	0.33	0.50	0.40	2
bus driver	1.00	0.50	0.67	2
business analyst	1.00	0.50	0.67	2
carpenter	1.00	1.00	1.00	2
chef	0.67	1.00	0.80	2
chemical engineer	0.00	0.00	0.00	2
civil engineer	0.67	1.00	0.80	2
cloud architect	0.50	0.50	0.50	2
construction manager	1.00	1.00	1.00	2
content writer	1.00	0.50	0.67	2
correctional officer	0.67	1.00	0.80	2
cosmetologist	1.00	1.00	1.00	2
counselor	1.00	1.00	1.00	2
custodian	0.00	0.00	0.00	2
customer service representative	0.00	0.00	0.00	2
data analyst	0.00	0.00	0.00	2
data engineer	0.00	0.00	0.00	2
data scientist	1.00	0.50	0.67	2
database administrator	0.50	0.50	0.50	2
detective	1.00	0.50	0.67	2
devops engineer	1.00	0.50	0.67	2
digital marketer	0.50	0.50	0.50	2
dispatcher	0.25	0.50	0.33	2
education consultant	1.00	1.00	1.00	2
electrical engineer	1.00	0.50	0.67	2
electrician	1.00	1.00	1.00	2
emergency medical technician	1.00	1.00	1.00	2
engineer	0.00	0.00	0.00	2
environmental engineer	1.00	1.00	1.00	2
esthetician	1.00	1.00	1.00	2

event planner	1.00	1.00	1.00	2
financial analyst	0.50	1.00	0.67	2
firefighter	1.00	1.00	1.00	2
fitness trainer	0.00	0.00	0.00	2
flight attendant	0.67	1.00	0.80	2
graphic designer	0.67	1.00	0.80	2
groundskeeper	0.00	0.00	0.00	2
hair stylist	1.00	1.00	1.00	2
hr manager	0.67	1.00	0.80	2
hvac technician	1.00	1.00	1.00	2
instructional designer	1.00	1.00	1.00	2
it manager	1.00	1.00	1.00	2
janitor	0.67	1.00	0.80	2
lab technician	1.00	1.00	1.00	2
landscaper	0.50	1.00	0.67	2
legal assistant	0.50	1.00	0.67	2
library assistant	1.00	1.00	1.00	2
logistics coordinator	1.00	1.00	1.00	2
machine learning engineer	1.00	0.50	0.67	2
machinist	1.00	1.00	1.00	2
marketing manager	1.00	0.50	0.67	2
massage therapist	0.67	1.00	0.80	2
mechanic	1.00	1.00	1.00	2
mechanical engineer	0.00	0.00	0.00	2
medical assistant	1.00	1.00	1.00	2
network engineer	1.00	0.50	0.67	2
nurse	0.67	1.00	0.80	2
operations manager	0.00	0.00	0.00	2
paralegal	1.00	0.50	0.67	2
paramedic	1.00	0.50	0.67	2
personal trainer	0.50	1.00	0.67	2
pharmacist	0.67	1.00	0.80	2
physical therapist	1.00	0.50	0.67	2
pilot	0.50	1.00	0.67	2
plumber	1.00	1.00	1.00	2
police officer	0.50	1.00	0.67	2
procurement manager	1.00	1.00	1.00	2
product manager	0.50	0.50	0.50	2
project manager	1.00	0.50	0.67	2
psychologist	0.67	1.00	0.80	2
quality assurance	1.00	0.50	0.67	2
research scientist	0.00	0.00	0.00	2
restaurant manager	0.25	0.50	0.33	2
safety manager	0.67	1.00	0.80	2

safety specialist	0.00	0.00	0.00	2
sales manager	0.33	1.00	0.50	2
school principal	1.00	1.00	1.00	2
security analyst	0.67	1.00	0.80	2
security guard	0.00	0.00	0.00	2
seo specialist	1.00	1.00	1.00	2
social worker	1.00	1.00	1.00	2
software engineer	0.33	0.50	0.40	2
supply chain analyst	1.00	0.50	0.67	2
systems administrator	0.00	0.00	0.00	2
teacher	0.67	1.00	0.80	2
technical support	0.00	0.00	0.00	2
therapist	1.00	0.50	0.67	2
tour guide	1.00	1.00	1.00	2
travel agent	0.67	1.00	0.80	2
truck driver	1.00	1.00	1.00	2
urban planner	1.00	1.00	1.00	2
waiter	1.00	0.50	0.67	2
web developer	0.67	1.00	0.80	2
welder	1.00	1.00	1.00	2
accuracy			0.71	202
macro avg	0.70	0.71	0.68	202
weighted avg	0.70	0.71	0.68	202

Support Vector Machine

Accuracy: 65.35%

	precision	recall	f1-score	support
accountant	0.50	0.50	0.50	2
aerospace engineer	0.00	0.00	0.00	2
aircraft mechanic	1.00	0.50	0.67	2
architect	1.00	0.50	0.67	2
automotive technician	1.00	1.00	1.00	2
barber	1.00	0.50	0.67	2
bartender	1.00	0.50	0.67	2
biomedical engineer	0.00	0.00	0.00	2
bus driver	1.00	0.50	0.67	2
business analyst	0.33	0.50	0.40	2
carpenter	1.00	0.50	0.67	2
chef	1.00	0.50	0.67	2
chemical engineer	0.00	0.00	0.00	2
civil engineer	0.33	0.50	0.40	2
cloud architect	1.00	1.00	1.00	2
construction manager	1.00	1.00	1.00	2
content writer	1.00	0.50	0.67	2
correctional officer	0.67	1.00	0.80	2
cosmetologist	0.25	0.50	0.33	2
counselor	1.00	1.00	1.00	2
custodian	0.50	0.50	0.50	2
customer service representative	0.00	0.00	0.00	2
data analyst	0.50	0.50	0.50	2
data engineer	1.00	1.00	1.00	2
data scientist	1.00	0.50	0.67	2
database administrator	1.00	0.50	0.67	2
detective	1.00	1.00	1.00	2
devops engineer	1.00	1.00	1.00	2
digital marketer	0.50	0.50	0.50	2
dispatcher	0.00	0.00	0.00	2
education consultant	1.00	1.00	1.00	2
electrical engineer	0.00	0.00	0.00	2
electrician	1.00	1.00	1.00	2
emergency medical technician	0.29	1.00	0.44	2
engineer	0.17	0.50	0.25	2
environmental engineer	0.67	1.00	0.80	2
esthetician	1.00	1.00	1.00	2
event planner	1.00	1.00	1.00	2
financial analyst	0.50	0.50	0.50	2

firefighter	1.00	1.00	1.00	2
fitness trainer	0.00	0.00	0.00	2
flight attendant	1.00	1.00	1.00	2
graphic designer	1.00	1.00	1.00	2
groundskeeper	0.50	1.00	0.67	2
hair stylist	0.00	0.00	0.00	2
hr manager	1.00	0.50	0.67	2
hvac technician	1.00	1.00	1.00	2
instructional designer	1.00	1.00	1.00	2
it manager	0.33	0.50	0.40	2
janitor	1.00	1.00	1.00	2
lab technician	1.00	1.00	1.00	2
landscaper	0.00	0.00	0.00	2
legal assistant	0.67	1.00	0.80	2
library assistant	1.00	1.00	1.00	2
logistics coordinator	1.00	1.00	1.00	2
machine learning engineer	1.00	0.50	0.67	2
machinist	1.00	1.00	1.00	2
marketing manager	0.50	0.50	0.50	2
massage therapist	1.00	1.00	1.00	2
mechanic	0.50	1.00	0.67	2
mechanical engineer	0.33	0.50	0.40	2
medical assistant	1.00	0.50	0.67	2
network engineer	1.00	0.50	0.67	2
nurse	0.67	1.00	0.80	2
operations manager	0.00	0.00	0.00	2
paralegal	0.50	0.50	0.50	2
paramedic	0.00	0.00	0.00	2
personal trainer	0.50	1.00	0.67	2
pharmacist	1.00	1.00	1.00	2
physical therapist	0.00	0.00	0.00	2
pilot	0.50	0.50	0.50	2
plumber	1.00	1.00	1.00	2
police officer	1.00	1.00	1.00	2
procurement manager	1.00	1.00	1.00	2
product manager	1.00	1.00	1.00	2
project manager	0.67	1.00	0.80	2
psychologist	0.50	0.50	0.50	2
quality assurance	1.00	0.50	0.67	2
research scientist	1.00	0.50	0.67	2
restaurant manager	0.33	0.50	0.40	2
safety manager	1.00	0.50	0.67	2
safety specialist	0.17	0.50	0.25	2
sales manager	0.40	1.00	0.57	2

school principal	1.00	1.00	1.00	2
security analyst	1.00	1.00	1.00	2
security guard	1.00	0.50	0.67	2
seo specialist	1.00	1.00	1.00	2
social worker	0.00	0.00	0.00	2
software engineer	1.00	0.50	0.67	2
supply chain analyst	1.00	0.50	0.67	2
systems administrator	0.33	0.50	0.40	2
teacher	0.33	1.00	0.50	2
technical support	0.00	0.00	0.00	2
therapist	0.50	0.50	0.50	2
tour guide	1.00	1.00	1.00	2
travel agent	1.00	1.00	1.00	2
truck driver	1.00	0.50	0.67	2
urban planner	1.00	0.50	0.67	2
waiter	0.33	0.50	0.40	2
web developer	0.67	1.00	0.80	2
welder	1.00	1.00	1.00	2
accuracy			0.65	202
macro avg	0.69	0.65	0.64	202
weighted avg	0.69	0.65	0.64	202

XGBoost

Accuracy: 65.84%

	precision	recall	f1-score	support
accountant	0.00	0.00	0.00	2
aerospace engineer	1.00	0.50	0.67	2
aircraft mechanic	0.67	1.00	0.80	2
architect	1.00	1.00	1.00	2
automotive technician	1.00	1.00	1.00	2
barber	1.00	0.50	0.67	2
bartender	0.67	1.00	0.80	2
biomedical engineer	1.00	0.50	0.67	2
bus driver	1.00	0.50	0.67	2
business analyst	0.50	0.50	0.50	2
carpenter	0.67	1.00	0.80	2
chef	1.00	1.00	1.00	2
chemical engineer	0.00	0.00	0.00	2
civil engineer	0.50	1.00	0.67	2
cloud architect	1.00	1.00	1.00	2
construction manager	1.00	1.00	1.00	2
content writer	1.00	0.50	0.67	2
correctional officer	1.00	1.00	1.00	2
cosmetologist	0.00	0.00	0.00	2
counselor	1.00	1.00	1.00	2
custodian	1.00	0.50	0.67	2
customer service representative	0.00	0.00	0.00	2
data analyst	0.00	0.00	0.00	2
data engineer	0.50	0.50	0.50	2
data scientist	1.00	1.00	1.00	2
database administrator	1.00	0.50	0.67	2
detective	0.00	0.00	0.00	2
devops engineer	0.50	0.50	0.50	2
digital marketer	0.50	0.50	0.50	2
dispatcher	0.00	0.00	0.00	2
education consultant	0.50	0.50	0.50	2
electrical engineer	0.00	0.00	0.00	2
electrician	0.67	1.00	0.80	2
emergency medical technician	0.25	0.50	0.33	2
engineer	0.00	0.00	0.00	2
environmental engineer	1.00	1.00	1.00	2
esthetician	1.00	1.00	1.00	2
event planner	1.00	1.00	1.00	2
financial analyst	0.50	1.00	0.67	2

firefighter	1.00	0.50	0.67	2
fitness trainer	1.00	0.50	0.67	2
flight attendant	1.00	0.50	0.67	2
graphic designer	0.67	1.00	0.80	2
groundskeeper	0.00	0.00	0.00	2
hair stylist	0.67	1.00	0.80	2
hr manager	0.50	1.00	0.67	2
hvac technician	1.00	1.00	1.00	2
instructional designer	0.67	1.00	0.80	2
it manager	1.00	0.50	0.67	2
janitor	1.00	0.50	0.67	2
lab technician	1.00	1.00	1.00	2
landscaper	0.33	0.50	0.40	2
legal assistant	0.40	1.00	0.57	2
library assistant	1.00	0.50	0.67	2
logistics coordinator	0.67	1.00	0.80	2
machine learning engineer	1.00	0.50	0.67	2
machinist	0.67	1.00	0.80	2
marketing manager	1.00	0.50	0.67	2
massage therapist	0.50	1.00	0.67	2
mechanic	0.00	0.00	0.00	2
mechanical engineer	0.00	0.00	0.00	2
medical assistant	1.00	0.50	0.67	2
network engineer	1.00	0.50	0.67	2
nurse	0.50	1.00	0.67	2
operations manager	1.00	0.50	0.67	2
paralegal	0.00	0.00	0.00	2
paramedic	1.00	0.50	0.67	2
personal trainer	0.67	1.00	0.80	2
pharmacist	1.00	1.00	1.00	2
physical therapist	1.00	1.00	1.00	2
pilot	0.67	1.00	0.80	2
plumber	1.00	1.00	1.00	2
police officer	0.67	1.00	0.80	2
procurement manager	1.00	1.00	1.00	2
product manager	0.50	1.00	0.67	2
project manager	0.50	0.50	0.50	2
psychologist	1.00	1.00	1.00	2
quality assurance	0.00	0.00	0.00	2
research scientist	1.00	0.50	0.67	2
restaurant manager	0.17	0.50	0.25	2
safety manager	0.67	1.00	0.80	2
safety specialist	0.00	0.00	0.00	2
sales manager	0.50	1.00	0.67	2

school principal	0.67	1.00	0.80	2
security analyst	0.25	0.50	0.33	2
security guard	0.67	1.00	0.80	2
seo specialist	1.00	1.00	1.00	2
social worker	0.00	0.00	0.00	2
software engineer	1.00	0.50	0.67	2
supply chain analyst	0.67	1.00	0.80	2
systems administrator	0.00	0.00	0.00	2
teacher	1.00	1.00	1.00	2
technical support	0.00	0.00	0.00	2
therapist	0.67	1.00	0.80	2
tour guide	1.00	1.00	1.00	2
travel agent	1.00	1.00	1.00	2
truck driver	0.67	1.00	0.80	2
urban planner	1.00	1.00	1.00	2
waiter	0.00	0.00	0.00	2
web developer	0.67	1.00	0.80	2
welder	1.00	1.00	1.00	2
accuracy			0.66	202
macro avg	0.65	0.66	0.62	202
weighted avg	0.65	0.66	0.62	202

Neural Network

Accuracy: 39.11%

	precision	recall	f1-score	support
accountant	1.00	0.50	0.67	2
aerospace engineer	0.00	0.00	0.00	2
aircraft mechanic	0.00	0.00	0.00	2
architect	0.00	0.00	0.00	2
automotive technician	1.00	1.00	1.00	2
barber	0.67	1.00	0.80	2
bartender	0.33	0.50	0.40	2
biomedical engineer	0.17	0.50	0.25	2
bus driver	1.00	1.00	1.00	2
business analyst	0.00	0.00	0.00	2
carpenter	1.00	0.50	0.67	2
chef	0.67	1.00	0.80	2
chemical engineer	0.00	0.00	0.00	2
civil engineer	1.00	0.50	0.67	2
cloud architect	0.33	0.50	0.40	2
construction manager	0.33	0.50	0.40	2
content writer	1.00	0.50	0.67	2
correctional officer	0.50	0.50	0.50	2
cosmetologist	0.00	0.00	0.00	2
counselor	0.50	0.50	0.50	2
custodian	0.50	0.50	0.50	2
customer service representative	0.00	0.00	0.00	2
data analyst	0.00	0.00	0.00	2
data engineer	0.00	0.00	0.00	2
data scientist	0.25	0.50	0.33	2
database administrator	0.67	1.00	0.80	2
detective	0.50	0.50	0.50	2
devops engineer	0.00	0.00	0.00	2
digital marketer	0.00	0.00	0.00	2
dispatcher	0.00	0.00	0.00	2
education consultant	1.00	0.50	0.67	2
electrical engineer	0.00	0.00	0.00	2
electrician	1.00	0.50	0.67	2
emergency medical technician	0.50	0.50	0.50	2
engineer	0.00	0.00	0.00	2
environmental engineer	0.00	0.00	0.00	2
esthetician	1.00	1.00	1.00	2
event planner	0.67	1.00	0.80	2
financial analyst	0.00	0.00	0.00	2

firefighter	1.00	1.00	1.00	2
fitness trainer	1.00	0.50	0.67	2
flight attendant	0.50	1.00	0.67	2
graphic designer	0.00	0.00	0.00	2
groundskeeper	0.00	0.00	0.00	2
hair stylist	0.00	0.00	0.00	2
hr manager	1.00	0.50	0.67	2
hvac technician	1.00	0.50	0.67	2
instructional designer	0.67	1.00	0.80	2
it manager	0.40	1.00	0.57	2
janitor	0.50	0.50	0.50	2
lab technician	0.67	1.00	0.80	2
landscaper	0.33	0.50	0.40	2
legal assistant	0.00	0.00	0.00	2
library assistant	0.67	1.00	0.80	2
logistics coordinator	0.25	0.50	0.33	2
machine learning engineer	0.00	0.00	0.00	2
machinist	1.00	1.00	1.00	2
marketing manager	0.25	0.50	0.33	2
massage therapist	0.50	1.00	0.67	2
mechanic	1.00	0.50	0.67	2
mechanical engineer	0.33	0.50	0.40	2
medical assistant	0.00	0.00	0.00	2
network engineer	1.00	0.50	0.67	2
nurse	0.25	0.50	0.33	2
operations manager	0.00	0.00	0.00	2
paralegal	0.00	0.00	0.00	2
paramedic	0.67	1.00	0.80	2
personal trainer	0.67	1.00	0.80	2
pharmacist	0.00	0.00	0.00	2
physical therapist	0.14	0.50	0.22	2
pilot	1.00	1.00	1.00	2
plumber	0.00	0.00	0.00	2
police officer	0.00	0.00	0.00	2
procurement manager	0.50	0.50	0.50	2
product manager	0.00	0.00	0.00	2
project manager	0.50	0.50	0.50	2
psychologist	0.33	0.50	0.40	2
quality assurance	0.00	0.00	0.00	2
research scientist	0.00	0.00	0.00	2
restaurant manager	0.00	0.00	0.00	2
safety manager	0.50	0.50	0.50	2
safety specialist	0.33	0.50	0.40	2
sales manager	0.50	0.50	0.50	2

school principal	0.50	0.50	0.50	2
security analyst	0.25	0.50	0.33	2
security guard	0.33	0.50	0.40	2
seo specialist	1.00	0.50	0.67	2
social worker	0.00	0.00	0.00	2
software engineer	0.00	0.00	0.00	2
supply chain analyst	0.00	0.00	0.00	2
systems administrator	0.00	0.00	0.00	2
teacher	0.50	0.50	0.50	2
technical support	0.00	0.00	0.00	2
therapist	0.00	0.00	0.00	2
tour guide	1.00	1.00	1.00	2
travel agent	0.00	0.00	0.00	2
truck driver	1.00	1.00	1.00	2
urban planner	0.00	0.00	0.00	2
waiter	0.00	0.00	0.00	2
web developer	0.00	0.00	0.00	2
welder	0.00	0.00	0.00	2
accuracy			0.39	202
macro avg	0.37	0.39	0.36	202
weighted avg	0.37	0.39	0.36	202