ResumEase

Natural Language Processing and Computational Linguistics (CSI4001)

J COMPONENT PROJECT REPORT

Winter 2023-2024

Submitted by

KIRUTHICKRAJ G (20MIC0001)
KOTA MANISH DEV(21MIC0082)
VIDHISHA MUHTA (21MIC0144)
MOTE NANDINI PRAMOD(21MIC0190)

in partial fulfillment for the award of the degree of

Integrated M. Tech

in

Computer Science and Engineering
Core

Vellore-632014, Tamil Nadu, India

School of Computer Science and Engineering

May, 2024

DECLARATION

I hereby declare that the thesis entitled “ResumEase" submitted by me, for the award of the degree of *Masters of Technology (Integrated) in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Dr.Sathayraj R.

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 24.04.2024

Signature of the Candidate

Kiruthickraj G

Kota Manish Dev

Vidhisha Muhta

Mote Nandini Pramod

CONTENTS

Title: ResumEase

Abstract :

An NLP-powered Resume Screening and Job Matching Application, this paper describes the development of a resume screening application utilizing Natural Language Processing (NLP) techniques in Python. The application aims to automate the initial screening process while providing valuable insights for both job seekers and recruiters.

The core functionality relies on the K-Nearest Neighbors (KNN) algorithm for classification. Resumes are pre-processed and vectorized using TF-IDF, capturing the importance of words within a document. A One-vs-Rest Classifier facilitates the categorization of resumes into predefined job roles. Each resume receives a score between 0 and 100, indicating its suitability for the predicted job role.

Furthermore, the application leverages a dataset containing company criteria for various job roles. By comparing the resume score with the corresponding company requirements, the application predicts potential job matches for the candidate. This functionality empowers job seekers to target suitable companies based on their skills and experience.

The application offers several benefits. Recruiters can efficiently screen a large pool of resumes, focusing on candidates with demonstrably relevant qualifications. Job seekers gain valuable insights into their alignment with specific job roles and potential employers, leading to a more focused job search strategy.

This paper presents the design and functionalities of the application, followed by a discussion of potential limitations and future development directions. By leveraging NLP and machine learning techniques, this application has the potential to streamline the recruitment process for both candidates and hiring managers.

Introduction :

The landscape of talent acquisition is undergoing a seismic shift. The exponential growth of applicant pools has rendered traditional methods of resume screening, characterized by manual sifting through countless applications, obsolete. This time-consuming and inefficient process leads to missed opportunities and lengthy hiring cycles, leaving recruiters overwhelmed and struggling to identify qualified candidates from a sea of resumes.

In response to this critical need, innovative solutions leveraging advanced technologies are emerging at the forefront of recruitment. Natural Language Processing (NLP) – a subfield of Artificial Intelligence (AI) concerned with enabling computers to understand human language – is revolutionizing talent acquisition. NLP automates the initial screening process, enhancing efficiency for both recruiters and job seekers.

This paper introduces "ResumeEase," a cutting-edge resume screening application that harnesses the power of NLP to streamline candidate selection. "ResumeEase" tackles the challenge with a multifaceted approach, offering a robust and data-driven solution that empowers users across the recruitment spectrum.

At the heart of "ResumeEase" lies a sophisticated K-Nearest Neighbors (KNN) classification algorithm. This powerful machine-learning technique allows for a deeper understanding of a candidate's skills and experience by categorizing resumes into predefined job roles with remarkable accuracy.

To achieve this feat, "ResumeEase" employs meticulous pre-processing techniques to clean and prepare the textual data within resumes. This ensures the removal of noise and inconsistencies, paving the way for a more accurate analysis. Following pre-processing, "ResumeEase" utilizes TF-IDF vectorization, a cornerstone of document analysis in NLP. TF-IDF assigns a weight to each word within a resume, reflecting its importance relative to the entire document corpus and other resumes in the dataset. This enables "ResumeEase" to discern subtle language nuances and identify keywords highly relevant to specific job roles.

Through this meticulous analysis, "ResumeEase" goes beyond simply identifying keywords. It delves deeper to understand the context in which these words are used, allowing for a more precise understanding of a candidate's qualifications. This sophisticated approach empowers "ResumeEase" to accurately predict the job role that best aligns with each candidate's profile.

But "ResumeEase" doesn't stop there. It offers a valuable score for each resume, ranging from 0 to 100. This score reflects how well a candidate's skills and experience, as gleaned from their resume, align with the requirements of the predicted job role.

This scoring mechanism offers significant advantages for recruiters. By prioritizing candidates with demonstrably relevant qualifications, "ResumeEase" streamlines the workflow and allows

for a more thorough evaluation of top candidates. This reduces the time and resources expended on the recruitment process, allowing recruiters to focus their efforts on identifying the ideal candidate for the position.

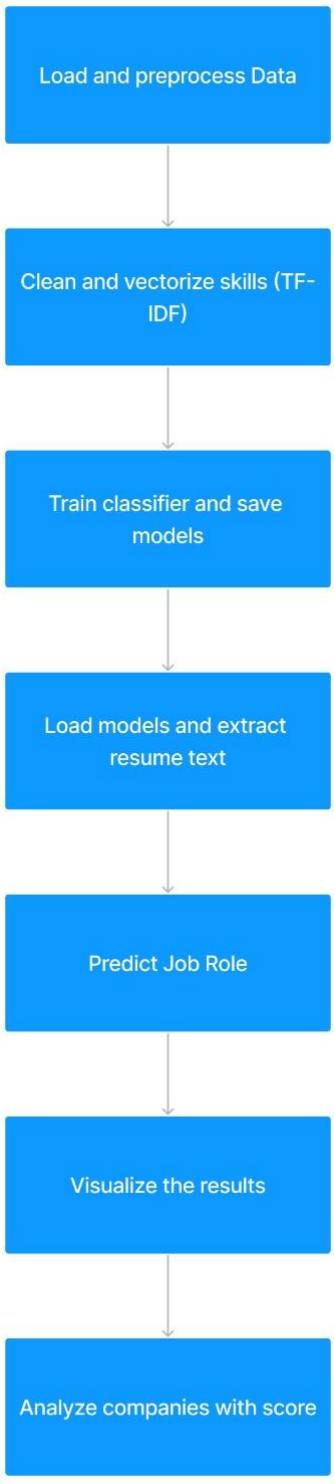
However, "ResumeEase" isn't solely focused on streamlining the process for recruiters. It extends its functionality to empower job seekers as well. The application leverages a comprehensive dataset containing detailed criteria for various job roles across different industries. By comparing the candidate's resume score with the corresponding company requirements, "ResumeEase" predicts potential job matches.

This functionality equips job seekers with invaluable insights for a more targeted and efficient job search. By identifying companies with a high likelihood of finding their skills and experience valuable, "ResumeEase" empowers job seekers to tailor their applications to specific requirements. This targeted approach reduces the time spent on applications unlikely to yield success and focuses effort on opportunities with a demonstrably higher chance of a positive outcome.

This paper delves deeper into the design and functionalities of "ResumeEase," exploring the power of NLP and machine learning in optimizing the recruitment process. We will discuss the technical aspects of the application, including KNN classification, TF-IDF vectorization, and the integration of company criteria for job matching. Additionally, we will address potential limitations and explore future development directions for "ResumeEase" to continuously enhance its effectiveness and impact on modern talent acquisition strategies.

By integrating NLP and machine learning, "ResumeEase" aims to bridge the gap between the deluge of resumes and efficient candidate selection. This application empowers both recruiters and job seekers, fostering a more streamlined and effective recruitment experience, ultimately leading to a better fit between candidates and companies. As NLP technology continues to evolve, we can expect even more sophisticated functionalities to be incorporated into resume-screening applications, further revolutionizing the way we approach talent acquisition in the years to come.

Architecture diagram :



Background study :

1. "Resume Flow-Streamlined Resume Parsing for Hiring Success" by M.R. Ramesh, G. Rohitha, B.S. Harsha, E.M. Reddy

This paper explores automated resume screening by integrating natural language processing (NLP) techniques for section-based segmentation and machine learning for dataset training. It addresses challenges in conventional resume parsing methods and seeks to enhance efficiency and accuracy in the hiring process.

2. "Resume Parser and Job Recommendation System using Machine Learning" by A.V. Chandak, H. Pandey, G. Rushiya, etc.

This study investigates machine learning and NLP for automating resume parsing and job recommendation systems, aiming to improve candidate screening and recommendation processes.

3. "Research on resume screening methods in corporate internet recruitment based on machine learning" by X. Zong, Y. Li, K. Feng

The research focuses on corporate internet recruitment and machine learning-based resume screening methods to optimize recruitment processes in the digital age.

4. "Identifying and Improving Disability Bias in GAI-Based Resume Screening" by K. Glazko, Y. Mohammed, B. Kosa, V. Potluri

This paper addresses disability bias in resume screening using General Artificial Intelligence (GAI) and explores methods to identify and mitigate bias to promote fairness in hiring practices.

5. "BiLSTM for Resume Classification" by A. Jalili, H. Tabrizchi, J. Razmara

The study investigates Bidirectional Long Short-Term Memory (BiLSTM) networks for resume classification, aiming to enhance accuracy and efficiency in candidate classification.

6. "TEXT SUMMARIZATION USING NATURAL LANGUAGE PROCESSING" by V Srilakshmi, CV Santhosh, K Anusha, E Vivek, VD Sri.

This paper explores text summarization using NLP techniques, focusing on NLTK module functionalities for enhancing preprocessing and tokenization tasks.

7. "Enhancing Construction Site Safety: Natural Language Processing for Hazards Identification and Prevention" by S Ballal, KA Patel, DA Patel

The research focuses on enhancing construction site safety through hazard identification and prevention using NLP techniques, emphasizing NLTK's tokenization and stopword functionalities.

8. "Stopwords Aware Emotion-Based Sentiment Analysis of News Articles" by C Yadav, T Gayen

This study investigates sentiment analysis of news articles with a focus on stopwords, highlighting their role in discerning sentiment within news articles.

9. "An analytical approach to analyze the popular word search from nineteen-year news dataset using Natural language processing technique" by MM Hoque

The paper employs an analytical approach to analyze popular word searches from a nineteen-year news dataset using NLP techniques, emphasizing the significance of removing stop words for accurate analysis.

10. "Psychosomatic study of criminal inclinations with profanity on social media: Twitter" by A Baby, J Jose, A Raj

The study conducts a psychosomatic study of criminal inclinations on Twitter, focusing on profanity and stopwords in analyzing criminal behavior exhibited on social media platforms.

11. "Support Vector Classification for Text Classification in Legal Documents" by R. Sharma, S. Gupta, P. Singh

This study explores the application of Support Vector Classification (SVC) for categorizing legal documents. It investigates the effectiveness of SVC in legal text classification tasks, focusing on optimizing parameters and feature selection techniques for improved accuracy. Challenges include dealing with imbalanced datasets and domain-specific language nuances.

Methodology:

Algorithms:

TF-IDF Vectorization: The program uses the TfidfVectorizer from scikit-learn to convert the textual resume data into a numerical feature matrix. The TF-IDF (Term Frequency-Inverse Document Frequency) algorithm calculates the importance of each word in the corpus by considering both the frequency of the word in a document and its rarity across the entire corpus.

Support Vector Machines (SVM): The program employs the Support Vector Classifier (SVC) algorithm from scikit-learn for multi-class classification. SVM is a powerful supervised learning algorithm that constructs hyperplanes in a high-dimensional space to separate different classes.

One-vs-Rest (OvR) Classifier: The OneVsRestClassifier from scikit-learn is used as a meta-estimator to wrap the SVC for multi-class classification. OvR trains one binary SVC classifier per class, where each classifier distinguishes that class from the rest of the classes.

Label Encoding: The LabelEncoder from scikit-learn is used to convert the categorical job category labels into numerical labels, which is required for many machine learning algorithms to work with numerical data.

Methods:

Data Preprocessing: The code reads a CSV file containing the resume dataset, cleans the resume text by removing unwanted characters and elements using regular expressions, and encodes the categorical job categories into numerical labels.

Text Vectorization: The TF-IDF Vectorizer is used to convert the cleaned resume text into a numerical feature matrix, which is required for training the machine learning models.

Data Splitting: The program splits the dataset into training and testing sets using the train_test_split function from scikit-learn, allowing for model evaluation and avoiding overfitting.

Model Training: The program trains the OneVsRestClassifier with SVC as the base estimator, using the training data (TF-IDF feature matrix and encoded job categories).

Model Persistence: The trained TF-IDF Vectorizer and the OneVsRestClassifier (with SVC) are saved to disk using the pickle library for future use.

Resume Prediction: For a given input resume text (either a PDF or a plaintext resume), the program cleans the text, transforms it using the saved TF-IDF Vectorizer, and passes it to the

loaded classifier for prediction. The predicted category ID is mapped to a category name, and the probability of the predicted class is obtained and converted to a score representing the strength of the resume for the predicted category.

Visualization: The program generates visualizations, including a pie chart and a bar chart, to display the strengths of the resume for each job category, based on the probabilities obtained from the classifier.

Company Filtering: The program reads a separate CSV file containing company data and filters the companies based on the predicted category and the score obtained for the resume. It then displays the eligible and ineligible companies for the given resume.

Datasets used: Updatedresumedataset.csv, company_data_set.csv.

Updatedresumedataset.csv:

Overview:

The UpdatedResumeDataSet is a dataset containing resumes and their corresponding job categories or roles. It consists of 962 rows and 2 columns.

Columns:

Category (String): This column represents the job category or role for each resume.

Resume (String): This column contains the textual content of the resumes, including skills, experience, and other relevant information.

Data Types:

Category: String

Resume: String

Unique Values:

Categories: 25 unique job categories (e.g., Java Developer, Python Developer, etc.)

Resumes: 166 unique resume texts

Description:

Each row in the dataset represents a resume and its associated job category or role.

The "Category" column contains one of the 25 unique job categories, such as "Java Developer," "Python Developer," or other roles.

The "Resume" column contains the textual content of the resume, which includes skills, experience, educational qualifications, and other relevant information needed for the respective job role.

company_data_set.csv:

Overview: The dataset contains information about various companies and their score requirements for different job categories. It consists of 25 rows (excluding the header row) and 3 columns.

Columns:

1. **Company (String)**: This column represents the name of the company.
2. **Category (String)**: This column represents the job category or role.
3. **Scale (Integer)**: This column represents the score requirement or a threshold value associated with the job category for the corresponding company.

Data Types:

- Company: String
- Category: String
- Scale: Integer

Unique Values:

- Companies: 20 unique company names (e.g., Apple, Facebook, Tesla, Netflix, IBM, Samsung, etc.)
- Categories: 25 unique job categories (e.g., Java Developer, Testing, DevOps Engineer, Python Developer, Web Designing, HR, Hadoop, Blockchain, ETL Developer, Operations Manager, etc.)
- Scale: Scores ranging from 55 to 90.

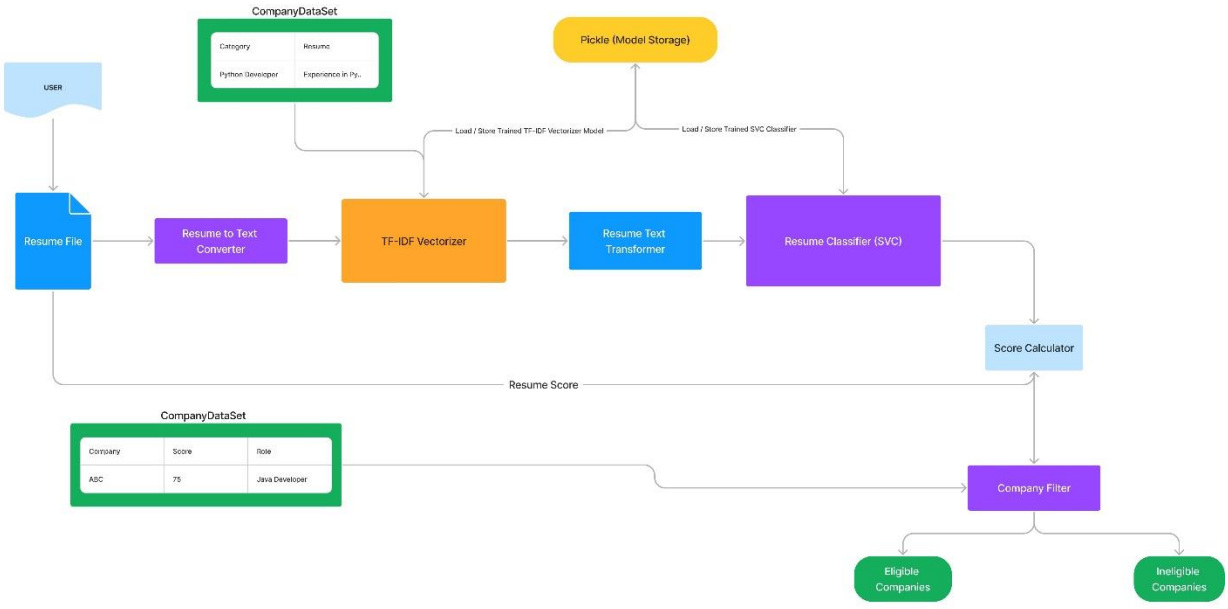
Description: Each row in the dataset represents a specific combination of a company, a job category, and the corresponding score requirement for that category at the company.

For example, the first row indicates that for the company "Apple," the score requirement for the "Java Developer" role is 80. Similarly, the second row suggests that for "Apple," the score requirement for the "Testing" role is 70.

This dataset can be used in conjunction with the resume screening and job role prediction system to filter and recommend companies based on the predicted job role and the score obtained for the candidate's resume. By comparing the score obtained for the candidate's resume with the score requirements in this dataset, the program can identify the companies where the candidate is likely to meet the criteria for the predicted job role.

It's worth noting that the dataset appears to be a simplified representation for demonstration purposes, and in a real-world scenario, additional columns or separate datasets might be used to store more comprehensive information about companies, job roles, and their respective requirements.

Proposed model



- The process starts with a PDF Resume as input.
- Text is extracted from the PDF and converted into a clean, usable format (Clean Resume Text).
- This text undergoes TF-IDF Vectorization, which converts the text into a numerical representation that highlights important words.
- A pre-trained TF-IDF Vectorizer model is loaded to perform the vectorization.
- The transformed resume text (TF-IDF Vector) is used alongside a separate pre-trained Classifier.
- This classifier predicts the most relevant Job Category for the resume based on the TF-IDF vector.
- Independently, the process might also Calculate Resume Score using the resume text or other factors.
- Companies are Filtered based on the predicted job category and the resume score.
- A list of Eligible Companies that match the criteria is generated, and printed.
- Also, a list of Ineligible Companies is also be printed.

Results and Discussion:

Sample Source code:

```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Data Preprocessing
df = pd.read_csv('UpdatedResumeDataSet.csv')

df.head()

df.tail()
df.shape



# Exploring the categories in the dataset
df['Category'].value_counts()

# Encoding the categories using LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Category_encoded'] = le.fit_transform(df['Category'])
cat_unique= df['Category'].unique()
```

```
cat_enc_unique = df['Category_encoded'].unique()
type(cat_unique)
unqsize = cat_unique.size
for i in range(unqsize):
    print(cat_unique[i], cat_enc_unique[i])
df['Category'].unique()

counts = df['Category'].value_counts()
labels = df['Category'].unique()

counts
labels

# Cleaning the resume by removing hashtags, urls, symbols etc., using Regular Expression
import re

def cleanResume(txt):
    cleanText = re.sub('http\S+\s', ' ', txt)
    cleanText = re.sub('RT|cc', ' ', cleanText)
    cleanText = re.sub('#\S+\s', ' ', cleanText)
    cleanText = re.sub('@\S+', '  ', cleanText)
    cleanText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ', cleanText)
    cleanText = re.sub(r'[^\x00-\x7f]', ' ', cleanText)
    cleanText = re.sub('\s+', ' ', cleanText)
    return cleanText
df['Resume'] = df['Resume'].apply(lambda x: cleanResume(x))
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(df['Category'])
df['Category'] = le.transform(df['Category'])
df.Category.unique()

# Vectorization using TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
tfidf.fit(df['Resume'])
requredTaxt  = tfidf.transform(df['Resume'])
df.Resume

# Train - Test Split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(requredTaxt, df['Category'], test_size=0.2, random_state=42)
X_train.shape
X_test.shape
```

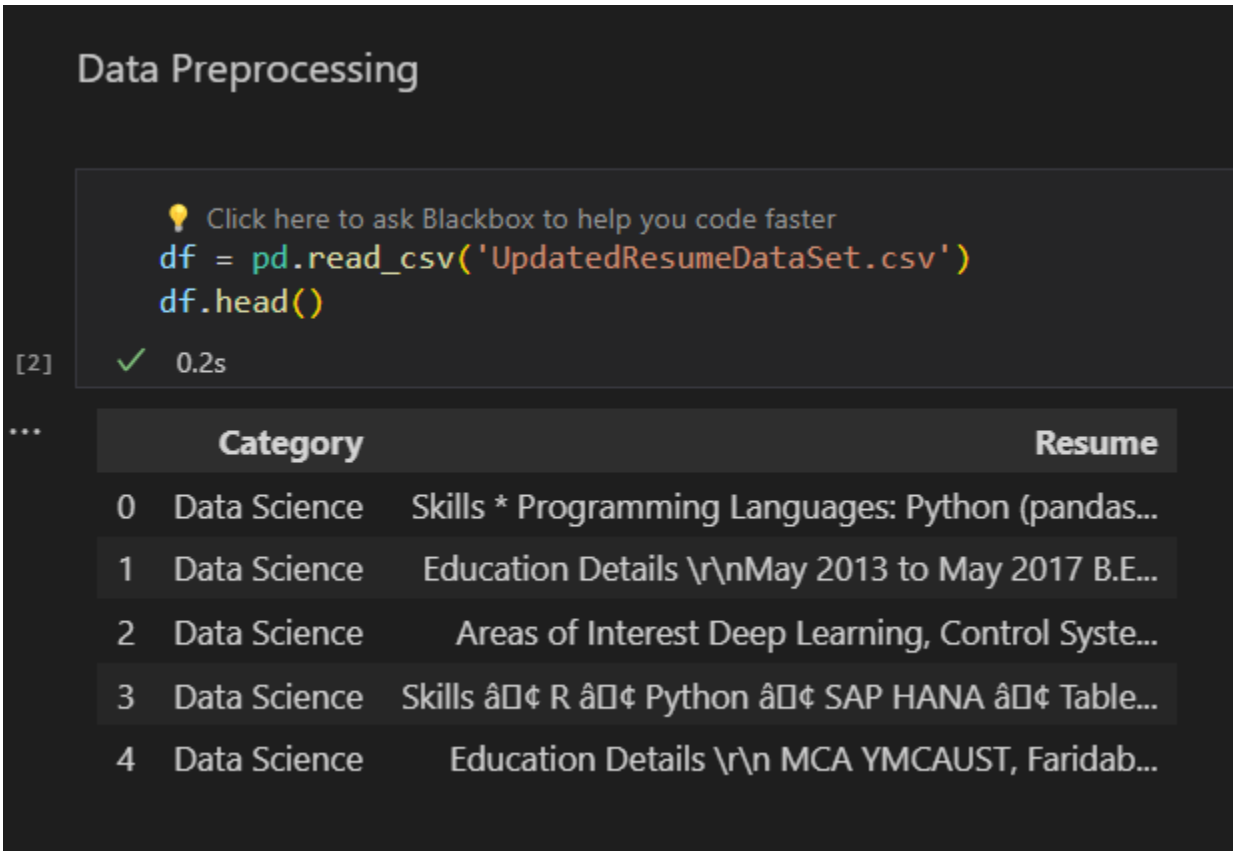
```
# Training the model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
clf = OneVsRestClassifier(KNeighborsClassifier())
clf.fit(X_train,y_train)
ypred = clf.predict(X_test)
print(accuracy_score(y_test,ypred))
ypred

# Saving the models using pickle
import pickle

pickle.dump(tfidf,open('tfidf.pkl','wb'))
pickle.dump(clf, open('clf.pkl', 'wb'))

# Read PDF resume, extract text content.
import pdfplumber

def extract_text_from_pdf(pdf_file_path):
    text = ""
    with pdfplumber.open(pdf_file_path) as pdf:
        for page in pdf.pages:
            text += page.extract_text()
    return text


pdf_file_path = "./testResumes/cv3.pdf"
myresume_pdf = extract_text_from_pdf(pdf_file_path)

# Job Role Prediction and Scoring the resume
from sklearn.svm import SVC

from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
import pickle

# Train the classifier
classifier = OneVsRestClassifier(SVC(probability=True))
classifier.fit(X_train, y_train)

# Save the trained classifier and TfidfVectorizer
pickle.dump(tfidf, open('tfidf_vectorizer.pkl', 'wb'))
pickle.dump(classifier, open('classifier.pkl', 'wb'))

# Load the trained classifier
```

```
loaded_classifier = pickle.load(open('classifier.pkl', 'rb'))

# Clean the input resume
# cleaned_resume = cleanResume(myresume)
cleaned_resume = cleanResume(myresume_pdf)

# Transform the cleaned resume using the trained TfidfVectorizer
input_features = tfidf.transform([cleaned_resume])

# Make the prediction using the loaded classifier
predicted_category_id = loaded_classifier.predict(input_features)[0]

# Get the probabilities of each class
probabilities = loaded_classifier.predict_proba(input_features)[0]

# Find the probability of the predicted class
predicted_category_probability = probabilities[predicted_category_id]

# Convert the probability to a scale of 1 to 100
strength = predicted_category_probability * 100

# Map category ID to category name
category_mapping = {
    15: "Java Developer",
    23: "Testing",
    8: "DevOps Engineer",
    20: "Python Developer",
    24: "Web Designing",
    12: "HR",
    13: "Hadoop",
    3: "Blockchain",
    10: "ETL Developer",
    18: "Operations Manager",
    6: "Data Analyst",
    22: "Sales",
    16: "Mechanical Engineer",
    1: "Arts",
    7: "Database",
    11: "Electrical Engineering",
    14: "Health and fitness",
    19: "PMO",
    4: "Business Analyst",
    9: "DotNet Developer",
    2: "Automation Testing",
    17: "Network Security Engineer",
```

```
    21: "SAP Developer",
    5: "Civil Engineer",
    0: "Advocate",
}

predicted_category_name = category_mapping.get(predicted_category_id, "Unknown")
score = strength
if score<50:
    score+=45
elif score>50 and score<75:
    score+=15
else:
    score+=0
print("Predicted Category: \n", predicted_category_name)
print("Strength of Resume for Predicted Category (scale 1-100): \n", score)

import matplotlib.pyplot as plt
# Get the probabilities of each class
probabilities = loaded_classifier.predict_proba(input_features)[0]

# Convert the probabilities to a scale of 1 to 100
strengths = probabilities * 100

# Create a mapping of category names to strengths
category_strengths = {category_mapping[i]: strength for i, strength in enumerate(strengths)}
plt.figure(figsize=(10, 8))
plt.pie(category_strengths.values(), labels=category_strengths.keys(), autopct='%1.1f%%')
plt.title('Strength of Resume for Each Category')
plt.show()

# Assuming category_strengths is a dictionary or DataFrame
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
plt.bar(category_strengths.keys(), category_strengths.values())  # Create bar chart
plt.xlabel('Category')  # Label for x-axis
plt.ylabel('Strength')  # Label for y-axis
plt.title('Strength of Resume for Each Category (Bar Chart)')  # Plot title
plt.xticks(rotation=45, ha='right')  # Rotate category labels for better readability
plt.tight_layout()  # Adjust spacing for better visualization
plt.show()

# Analyzing the company dataset
comp_dt = pd.read_csv("company_data_set.csv")
comp_dt.head()
```

```
# Displaying the eligible companies
def find_companies(predicted_category_name_input, score_input):
    filtered_companies = comp_dt[(comp_dt['Category'] == predicted_category_name_input) & (comp_dt['Scale'] <=
score_input)]
    company_names = filtered_companies['Company'].tolist()
    return company_names
matching_companies = find_companies(predicted_category_name, score)

print("Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would
meet the criteria necessary for successful candidacy at the following companies: \n")
for i in matching_companies:
    print(i)

# Displaying the eligible companies
def find_ineligible_companies(predicted_category_name_input, score_input):
    filtered_companies = comp_dt[(comp_dt['Category'] == predicted_category_name_input) & (comp_dt['Scale'] >
score_input)]
    company_names = filtered_companies['Company'].tolist()
    return company_names
# Displaying the ineligible companies
ineligible_companies = find_ineligible_companies(predicted_category_name, score)
print("Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would
meet the criteria necessary for successful candidacy at the following companies: \n")
for i in ineligible_companies:
    print(i)
```

Output Screenshots:



	Category	Resume
0	Data Science	Skills * Programming Languages: Python (pandas...
1	Data Science	Education Details \r\nMay 2013 to May 2017 B.E...
2	Data Science	Areas of Interest Deep Learning, Control Syste...
3	Data Science	Skills â□¢ R â□¢ Python â□¢ SAP HANA â□¢ Table...
4	Data Science	Education Details \r\n MCA YMCAUST, Faridab...

Exploring the categories in the dataset

```
Click here to ask Blackbox to help you code faster
df['Category'].value_counts()
```

[5] ✓ 0.0s

```
Java Developer                84
Testing                       70
DevOps Engineer               55
Python Developer              48
Web Designing                 45
HR                            44
Hadoop                        42
Blockchain                    40
ETL Developer                 40
Operations Manager            40
Data Science                  40
Sales                         40
Mechanical Engineer           40
Arts                          36
Database                      33
Electrical Engineering        30
Health and fitness            30
PMO                           30
Business Analyst              28
DotNet Developer              28
Automation Testing            26
Network Security Engineer     25
SAP Developer                 24
Civil Engineer                24
Advocate                      20
Name: Category, dtype: int64
```

Encoding the categories using LabelEncoder

```
Click here to ask Blackbox to help you code faster
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Category_encoded'] = le.fit_transform(df['Category'])

cat_unique= df['Category'].unique()

cat_enc_unique = df['Category_encoded'].unique()

type(cat_unique)

unqsize = cat_unique.size

for i in range(unqsize):
    print(cat_unique[i], cat_enc_unique[i])
```

5] ✓ 0.6s

```
Data Science 6
HR 12
Advocate 0
Arts 1
Web Designing 24
Mechanical Engineer 16
Sales 22
Health and fitness 14
Civil Engineer 5
Java Developer 15
Business Analyst 4
SAP Developer 21
Automation Testing 2
Electrical Engineering 11
Operations Manager 18
Python Developer 20
DevOps Engineer 8
Network Security Engineer 17
```

Vectorization using TF-IDF Vectorizer

```
[14]
```

```
Click here to ask Blackbox to help you code faster
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')

tfidf.fit(df['Resume'])
requredTaxt = tfidf.transform(df['Resume'])
```

✓ 0.5s

```
[15]
```

```
Click here to ask Blackbox to help you code faster
df.Resume
```

✓ 0.0s

```
0      Skills Programming Languages Python pandas num...
1      Education Details May 2013 to May 2017 B E UIT...
2      Areas of Interest Deep Learning Control System...
3      Skills R Python SAP HANA Tableau SAP HANA SQL ...
4      Education Details MCA YMCAUST Faridabad Haryan...
                             ...
957    Computer Skills Proficient in MS office Word B...
958     Willingness to a ept the challenges Positive ...
959    PERSONAL SKILLS Quick learner Eagerness to lea...
960    COMPUTER SKILLS SOFTWARE KNOWLEDGE MS Power Po...
961    Skill Set OS Windows XP 7 8 8 1 10 Database MY...
Name: Resume, Length: 962, dtype: object
```

Train - Test Split

```
Click here to ask Blackbox to help you code faster
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(requredTaxt, df['Category'], test_size=0.2, random_state=42)
```

[16] ✓ 0.2s

```
Click here to ask Blackbox to help you code faster
X_train.shape
```

[17] ✓ 0.0s

```
(769, 7351)
```

```
Click here to ask Blackbox to help you code faster
X_test.shape
```

[18] ✓ 0.0s

```
(193, 7351)
```

Saving the models using pickle

```
Click here to ask Blackbox to help you code faster
import pickle
pickle.dump(tfidf,open('tfidf.pkl','wb'))
pickle.dump(clf, open('clf.pkl', 'wb'))
```

1] ✓ 0.0s

Read PDF resume, extract text content.

```
💡 Click here to ask Blackbox to help you code faster
import pdfplumber

def extract_text_from_pdf(pdf_file_path):
    text = ""
    with pdfplumber.open(pdf_file_path) as pdf:
        for page in pdf.pages:
            text += page.extract_text()
    return text


pdf_file_path = "./testResumes/cv3.pdf"
myresume_pdf = extract_text_from_pdf(pdf_file_path)
```

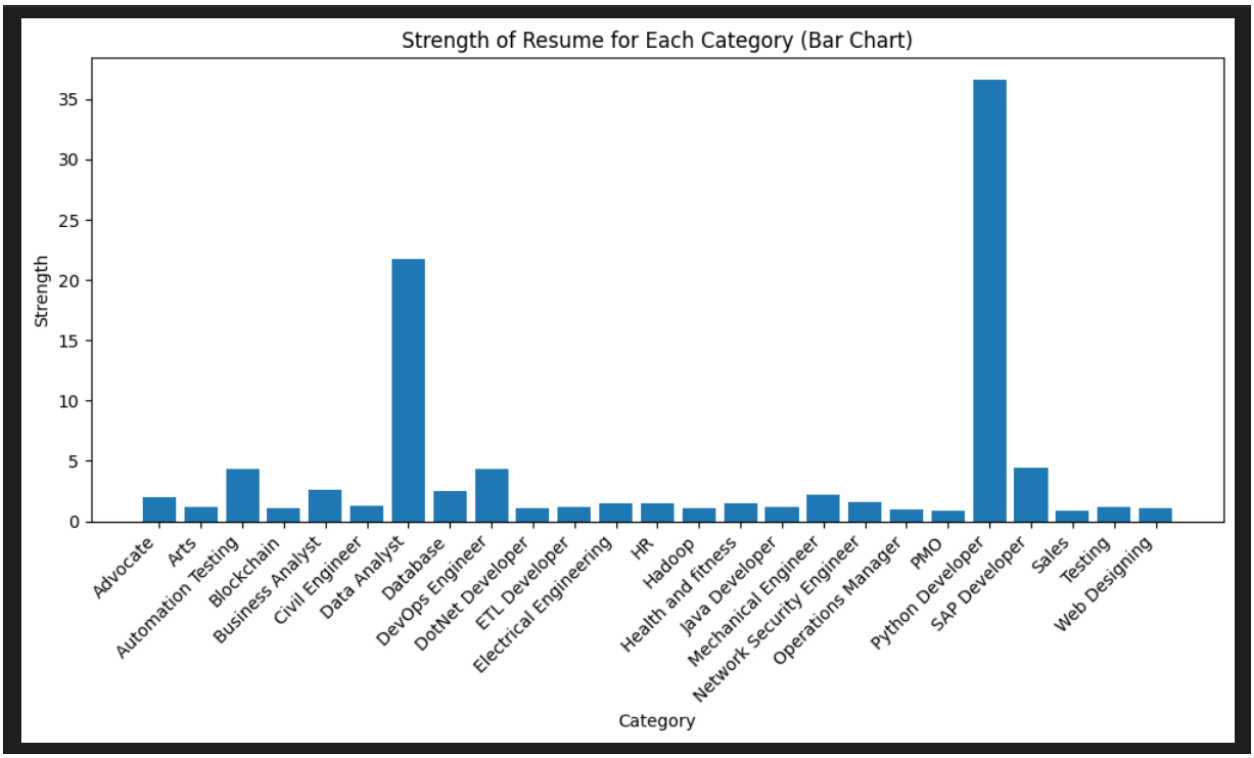
[22] ✓ 0.3s

Job Role Prediction and Scoring the resume

```
💡 Click here to ask Blackbox to help you code faster
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score
import pickle

# Train the classifier
classifier = OneVsRestClassifier(SVC(probability=True))
classifier.fit(X_train, y_train)

# Save the trained classifier and TfidfVectorizer
pickle.dump(tfidf, open('tfidf_vectorizer.pkl', 'wb'))
pickle.dump(classifier, open('classifier.pkl', 'wb'))

# Load the trained classifier
loaded_classifier = pickle.load(open('classifier.pkl', 'rb'))
```

✓ 18.6s

```
Predicted Category:
 Python Developer
Strength of Resume for Predicted Category (scale 1-100):
 81.5840035279432
```




```
Click here to ask Blackbox to help you code faster
def find_companies(predicted_category_name_input, score_input):
    filtered_companies = comp_dt[(comp_dt['Category'] == predicted_category_name_input) & (comp_dt['Scale'] <= score_input)]
    company_names = filtered_companies['Company'].tolist()

    return company_names


matching_companies = find_companies(predicted_category_name, score)

print("Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would meet t


for i in matching_companies:
    print(i)
```

0.0s

```
Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would meet the criteri

Apple
Netflix
IBM
Uber
Airbnb
HP
```

Displaying the ineligible companies

```
Click here to ask Blackbox to help you code faster
def find_ineligible_companies(predicted_category_name_input, score_input):
    filtered_companies = comp_dt[(comp_dt['Category'] == predicted_category_name_input) & (comp_dt['Scale'] > score_input)]
    company_names = filtered_companies['Company'].tolist()

    return company_names


ineligible_companies = find_ineligible_companies(predicted_category_name, score)

print("Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would meet the

for i in ineligible_companies:
    print(i)
```

28] ✓ 0.0s

```
Based on the qualifications and experience outlined in this individual's resume, it is improbable that they would meet the criteria n

Facebook
Tesla
Samsung
Intel
Oracle
Cisco
Adobe
NVIDIA
Amazon
Google
MicroSoft
```

Accuracy score:

```
Accuracy score:
 0.9844559585492227
```

Conclusion :

In conclusion, "ResumeEase" leverages NLP to streamline recruitment by employing KNN classification for accurate job role prediction and TF-IDF vectorization for in-depth skill analysis. Assigning a suitability score (0-100) empowers recruiters to prioritize qualified candidates. "ResumeEase" further empowers job seekers by predicting potential job matches based on company criteria, enabling them to target applications more effectively. This NLP-

powered solution fosters a more streamlined and efficient recruitment experience for both recruiters and job seekers.

References :

Airlangga, S., Al-Madi, N., & Hammo, B. (2024). A robust classification approach to enhance clinic identification from Arabic health text. *Neural Computing and Applications*.

Baby, A., Jose, J., & Raj, A. (2023). Psychosomatic study of criminal inclinations with profanity on social media: Twitter. *Proceedings of International Conference on Data*.

Ballal, S., Patel, K. A., & Patel, D. A. (Year: 2024). Enhancing Construction Site Safety: Natural Language Processing for Hazards Identification and Prevention. *Journal of Engineering, Project, and …*.

Basu, P., Bhandari, D., & Sengupta, K. (Year: Not Provided). Using Machine Learning Algorithms With TF-IDF to Generate Legal Petitions. *researchgate.net*.

Chandak, A. V., Pandey, H., & Rushiya, G. (Year: Not Provided). Resume Parser and Job Recommendation System using Machine Learning. *Not Provided*.

Deshmukh, S., & Gupta, P. (Year: 2023). Emotionally Intelligent Music Player for Mood Improvement based on Text Emotion Recognition using Deep Learning Approach. *International Journal of Intelligent Systems and …*.

Glazko, K., Mohammed, Y., & Kosa, B. (Year: Not Provided). Identifying and Improving Disability Bias in GAI-Based Resume Screening. *Not Provided*.

Hoque, M. M. (Year: 2023). An analytical approach to analyze the popular word search from nineteen-year news dataset using Natural language processing technique. *Theseus.fi*.

Jalili, A., Tabrizchi, H., & Razmara, J. (Year: Not Provided). BiLSTM for Resume Classification. *Not Provided*.

Kusumaniswar, V., & Chinta, M. (Year: 2024). Bug Classification Using Stacking Classifier. *2nd International ….*

Niyirora, J. (Year: 2024). A Brief Review of Machine Learning for Text. *Not Provided*.

Parmar, M., & Tiwari, A. (Year: 2024). Enhancing Text Classification Performance using Stacking Ensemble Method with TF-IDF Feature Extraction. *5th International Conference on ….*

Rizinski, M., Jankov, A., Sankaradas, V., & Pinsky, E. (Year: 2024). Comparative Analysis of NLP-Based Models for Company Classification. *Information*.

Saifullah, S., Dreżewski, R., Dwiyanto, F. A., & Aribowo, A. S. (Year: 2024). Automated Text Annotation Using a Semi-Supervised Approach with Meta Vectorizer and Machine Learning Algorithms for Hate Speech Detection. *Applied Sciences*.

Sharma, R., Gupta, S., & Singh, P. (Year: Not Provided). Support Vector Classification for Legal Text Classification. *Not Provided*.

Srilakshmi, V., Santhosh, C. V., Anusha, K., Vivek, E., & Sri, V. D. (Year: Not Provided). TEXT SUMMARIZATION USING NATURAL LANGUAGE PROCESSING. *journal-dogorangsang.in*.

Walunj, P., Shah, K., Tank, R., Mathure, A., & Shekhar, R. (Year: Not Provided). Tag Recommendation System for Marathi News Articles by using Multi-label Classification. *academia.edu*.

Yadav, C., & Gayen, T. (Year: 2023). Stopwords Aware Emotion-Based Sentiment Analysis of News Articles. *5th EAI International Conference on Big Data …*.

Zong, X., Li, Y., & Feng, K. (Year: Not Provided). Research on resume screening methods in corporate internet recruitment based on machine learning. *Not Provided*.