Resume Rating System

Mentors :Gaurav Rampuria, Siddhant Shekhar, Shruti Sekhar

January 18, 2025

Contents

1 Introduction

The document explains a Python-based system designed to evaluate resumes, classify skills by job category, and rank candidates based on their relevance to specific roles. Key components include:

- Data preprocessing and cleaning.
- Classification of skills using Generative AI (Google Gemini).
- Scoring resumes based on skill relevance.
- Visualizing the results.

2 Code Explanation

The Python script is divided into the following steps:

2.1 Mounting Google Drive and Importing Libraries

The script starts by mounting Google Drive to access files and importing necessary libraries:

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

2.2 Loading and Preprocessing Data

This section focuses on cleaning the data within the Category column and then identifying the unique categories present:

```
data = pd.read_csv('/content/drive/MyDrive/resumes.csv')
data = data.iloc[:, 0:7]
data.head()

# Remove invalid categories
```

```
for index, row in data.iterrows():
  if row['Category'] == np.nan or row['Category'] == '
     Tableau</li> <li> Data warehouses' or row['Category
     '] == "nan":
    data.drop(index, inplace=True)
```

```
['AVIATION',
 'ENGINEERING',
 'PUBLIC-RELATIONS',
 'FITNESS',
 'CHEF',
 'BANKING',
 'AGRICULTURE',
 'ADVOCATE',
 'INFORMATION-TECHNOLOGY',
 'CONSTRUCTION',
 'ACCOUNTANT',
 'ARTS',
 'CONSULTANT',
 'APPAREL',
 'DESIGNER',
 'SALES',
 'HR',
 'TEACHER',
 nan,
 'BUSINESS-DEVELOPMENT',
 'FINANCE',
 'DIGITAL-MEDIA',
 'HEALTHCARE',
 'AUTOMOBILE',
 'BPO']
```

List of unique categories found in the data .

2.3 Extracting and Organizing Skills

Skills are parsed and organized by job category.This dictionary is designed to store unique skills associated with different job categories:

```
categories = data['Category'].unique().tolist()
skill_dictionary = {}
for category in categories:
  cat_skill = []
```

```
  for index, row in data.iterrows():
    if row['Category'] == category:
      # Clean and parse skills
      \ldots
  skill_dictionary[category] = cat_skill
```

2.4 Skill Classification Using Generative AI

Here we have made the use of Generative AI to classify skills into four categories:

- Very Relevant: Implies these are very relevant skills used by industry professionals extensively
- Relevant: these skills are relevant to the category role directly
- Somewhat relevant: Not directly relevant but can be put to use in the role
- Irrelevant: Skills not related to this job category.
  The program takes as input a dictionary of job categories and a list of skills associated with each category. It sends these skills to Google's Gemini AI model (through LangChain), asking the model to classify each skill based on its relevance to the job role. The AI generates a response that classifies each skill into one of the four categories. The results are formatted in JSON for easy interpretation and processing.

The function `classify_skills_by_category` integrates Google Gemini for classification:

```
!pip install langchain_google_genai
from langchain_google_genai import GoogleGenerativeAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
import os
import json
from typing import Dict, List

def classify_skills_by_category(skills_dictionary: Dict[
   str, List[str]]) -> Dict[str, Dict[str, List[str]]]:
    os.environ["GOOGLE_API_KEY"] = "AIzaSyBQapvjvdx-
       HtN37sPFB7OgTbEXAEcFxdE"
    llm = GoogleGenerativeAI(
```

```
        model="gemini-pro",
        google_api_key=os.environ["GOOGLE_API_KEY"],
        temperature=0.1
    )



    # Create chain
    chain = LLMChain(
        llm=llm,
        prompt=PromptTemplate(
            input_variables=["category", "skills"],
            template=prompt_template
        )
    )
    results = {}
    for category, skills in skills_dictionary.items():
        print(f"\nProcessing category: {category}")
        skills_str = ", ".join(skills)

        try:
            response = chain.run(category=category,
               skills=skills_str)
            response = response.strip()
            if response.startswith("```json"):
                response = response.replace("```json", "
                   ").replace("```", "")
            parsed_result = json.loads(response)
            required_categories = ["very_relevant", "
               relevant", "somewhat_relevant", "
               irrelevant"]
            for cat in required_categories:
                if cat not in parsed_result:
                    parsed_result[cat] = []
            all_classified_skills = []
            for skills_list in parsed_result.values():
                all_classified_skills.extend(skills_list
                   )

            results[category] = parsed_result

        except Exception as e:
            results[category] = {
```

```
                "very_relevant": [],
                "relevant": [],
                "somewhat_relevant": [],
                "irrelevant": []
            }

    return results
classified_results = classify_skills_by_category(
   skill_dictionary)
classified_results['BPO']
```

```
classified_results['BPO']
```

```
{'very_relevant': ['software', 'database'],
 'relevant': ['timeconstraint'],
 'somewhat_relevant': ['marketing'],
 'irrelevant': ['medium']}
```

2.5 Scoring and Ranking Resumes

This code snippet aims to calculate a weighted percentage score for each candidate's skills based on how relevant those skills are to the candidate's job category.
Basic match scores and weighted percentages are calculated:

```
def calculate_weighted_percentage(candidate_skills,
   classified_skills):
    weighted_score = 0
    maximum_possible_weighted_score = 0
    for skill_type, skill_list in classified_skills.
       items():
            weight = 0
            if skill_type == "very_relevant":
                weight = 5
            elif skill_type == "relevant":
                weight = 2
            elif skill_type == "somewhat_relevant":
                weight = 1

            for skill in skill_list:
```

```
                maximum_possible_weighted_score += weight
                if skill in candidate_skills:
                    weighted_score += weight

    if maximum_possible_weighted_score == 0:
        return 0

    return (weighted_score /
       maximum_possible_weighted_score) * 100

for index, row in data.iterrows():
    candidate_skills = row['skills']
    category = row['Category']

    if category in classified_results:
        weighted_percentage =
           calculate_weighted_percentage(
           candidate_skills, classified_results[category
           ])
        data.loc[index, 'weighted_percentage'] =
           weighted_percentage
    else:
      data.loc[index, 'weighted_percentage'] = 0

data
```

In essence, this code assesses the relevance of a candidate's skills to their target job category and quantifies it using a weighted percentage score. This score can be used for ranking and filtering candidates based on their skill match.

```
avg_very_relevant_skill_ratio = data['
   Relevant_Skill_Ratio'].mean()
avg_basic_match_score = data['basic_match_score'].mean()
avg_weighted_percentage = data['weighted_percentage'].
   mean()
avg_resume_efficiency = data['Resume_Efficiency'].mean()


data['scaled_very_relevant_skill_ratio'] = data['
   Relevant_Skill_Ratio'] * (5 /
   avg_very_relevant_skill_ratio)
data['scaled_basic_match_score'] = data['
   basic_match_score'] * (10 / avg_basic_match_score)
```

```
data['scaled_weighted_percentage'] = data['
    weighted_percentage'] * (25 / avg_weighted_percentage
    )
data['scaled_resume_efficiency'] = data['
    Resume_Efficiency'] * (10 / avg_resume_efficiency)

data['Total_Score'] = data['
    scaled_very_relevant_skill_ratio'] + data['
    scaled_basic_match_score'] + data['
    scaled_weighted_percentage'] + data['
    scaled_resume_efficiency']

data.head()
```

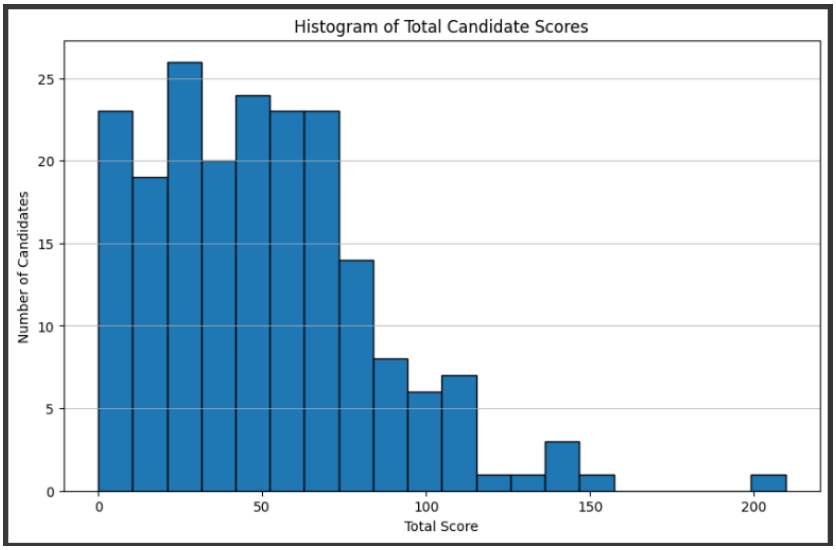
The code aims calculation for scaling and summing scores across several columns in a DataFrame (data) to generate a Total Score for each row, which likely represents an evaluation of resumes or job applications based on multiple criteria
For each of the columns listed , the code scales the values relative to their average using a scaling factor. The scaling factor is calculated as the ratio of a constant value (e.g., 5, 10, 25) to the average value of the respective column. The code scales four columns (Relevant Skill Ratio, basic match score, weighted percentage, and Resume Efficiency) in the DataFrame by comparing each value to the average of its respective column. This makes the values relative to their averages. After scaling, it computes a Total Score for each row by summing the scaled values. This method normalizes the scores, allowing for direct comparison and combination, even when the original columns have different ranges or distributions.

2.6 Visualizing the Results

Histograms and scatter plots provide a visual summary of the analysis:

```

import matplotlib.pyplot as plt


plt.figure(figsize=(10, 6))
plt.hist(data['Total_Score'], bins=20, edgecolor='black'
    )
plt.xlabel('Total Score')
plt.ylabel('Number of Candidates')
plt.title('Histogram of Total Candidate Scores')
```

```
plt.grid(axis='y', alpha=0.75)
plt.show()
```



3 Results and Insights

- **Basic Match Score:** Measures the percentage of matching skills.
- **Weighted Percentage:** Assigns scores based on skill relevance.
- **Resume Efficiency:** Evaluates overall effectiveness considering all metrics.

Grading is applied to categorize candidates:

```
def get_grade(percentile):
    if percentile >= 75:
        return 'A'
    elif percentile >= 40:
        return 'B'
    else:
        return 'C'
data['Percentile_cat'] = data.groupby('Category')['Total_Score'].transform(
        lambda x: x.rank(pct=True) * 100)
data['Grade_cat'] = data['Percentile_cat'].apply(get_grade)
```

```
data['Percentile'] = data['Total_Score'].transform(
        lambda x: x.rank(pct=True) * 100)
data['Grade'] = data['Percentile'].apply(get_grade)
data.head()
```

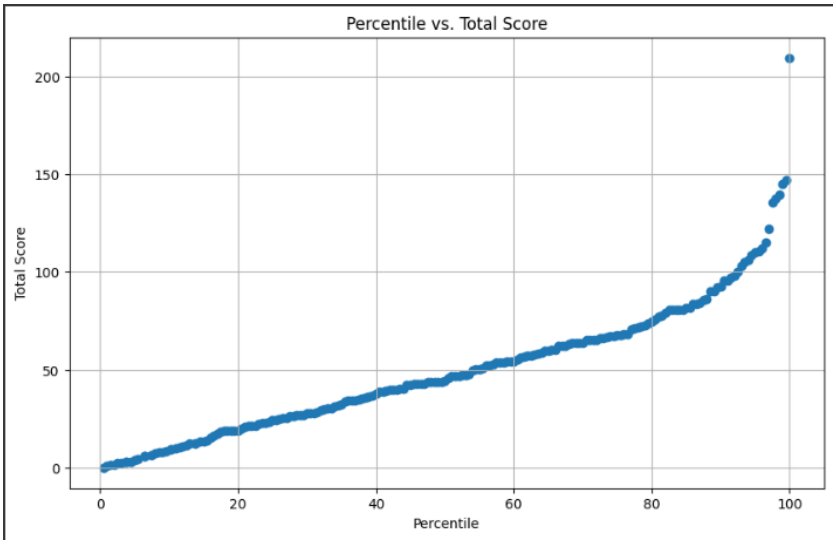
The code computes the percentile ranks for scores both within each category and across the entire dataset. It then assigns grades ('A', 'B', or 'C') to each entry based on these percentile ranks, for both category-specific and overall percentiles. Finally, it displays the first few rows of the updated dataset, showing the newly added percentile and grade columns. This method effectively categorizes scores into percentiles and assigns grades accordingly, providing a clearer understanding of how each score compares within its category and on a global scale

```
import ipywidgets as widgets
from IPython.display import display

Job_Category = None

dropdown = widgets.Dropdown(
    options = categories,
    value='ENGINEERING',
    description='Choose:',
    disabled=False,
)

submit_button = widgets.Button(description="Submit")

def on_button_click(change):
    global Job_Category
    Job_Category = dropdown.value
    print(f"Job category stored: {Job_Category}")
    category_data = data[data['Category'] ==
       Job_Category].sort_values(by='Total_Score',
       ascending=False)
    print(category_data)

submit_button.on_click(on_button_click)

display(dropdown, submit_button)
```

Here we have created a user interface within the notebook. Users can select a job category from the dropdown and click "Submit". The code then filters

and displays data for the chosen category, sorted by 'Total Score'. This allows for interactive exploration of the data based on job categories.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(data['Percentile'], data['Total_Score'])
plt.xlabel('Percentile')
plt.ylabel('Total Score')
plt.title('Percentile vs. Total Score')
plt.grid(True)
plt.show()
```



4 Conclusion

This Python script offers an automated system for resume analysis and skill evaluation. By leveraging Generative AI, it provides nuanced classification and scoring, enabling better candidate selection.