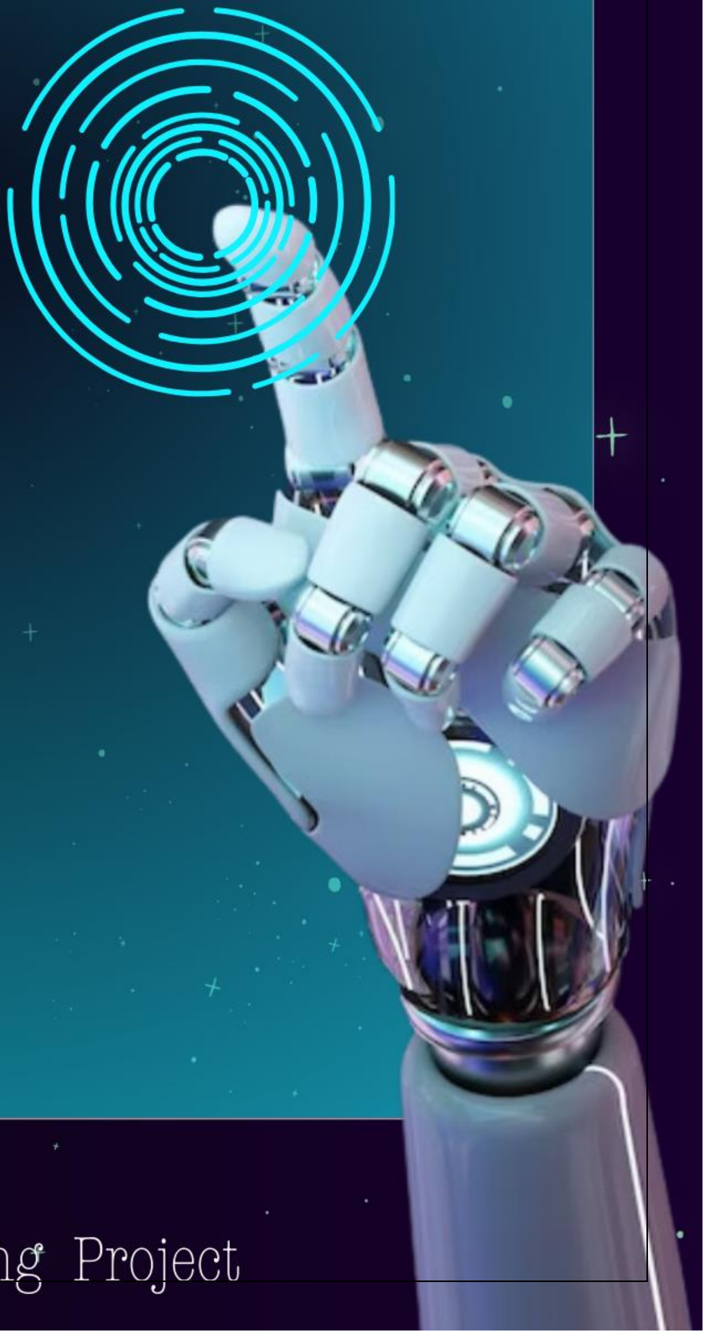
RESUME SCREENING

Our project focuses on Resume Screening as a multi-classification problem using machine learning algorithms. It involves dataset exploration, preprocessing, model training, cross-validation, and evaluation to select the best-performing model.



Authors:
Safeena & Suhana

Assigned By:
Dr.Attaullah

3 MAY - 2025

Machine Learning Project

Contents

Acknowledgement

We would like to express our heartfelt gratitude to our course instructor, **Sir Attaullah**, for his continuous guidance and support throughout this project. His clear explanation of theoretical concepts in **Machine Learning** greatly enhanced our understanding.

We are also sincerely thankful to **Sir Abdul Rafio** for teaching us **Python programming**, which was essential in implementing our project.

This project, being our first hands-on experience in the field of Machine Learning, has significantly boosted our confidence and sparked a deeper interest in the subject. We truly appreciate the dedication, teamwork, and learning environment that made this achievement possible.

Objective

The goal of this project is to create a model that can automatically categorize resumes into different job roles based on the information in them. The objective is to accurately predict which job category a resume belongs to.

The key objectives of the project are:

- To explore and clean the resume data.
- To split the data into training, validation, and test sets.
- To build and train two models: Logistic Regression and K-Nearest Neighbors (KNN).
- To improve the models using k-fold cross-validation to make them more reliable.
- To evaluate the model performance using metrics like accuracy, ROC-AUC, and confusion matrices.
- To compare the models, choose the best one, and suggest improvements for the future.

Abstract

This report explains a project where we used machine learning to solve a multiclass classification problem with a dataset from the UCI Machine Learning Repository. We cleaned and prepared the data, then trained two models: Logistic Regression and K-Nearest Neighbors (KNN). We tested how well each model worked using accuracy, ROC-AUC scores, and confusion matrices. After comparing the results, we chose the best-performing model. The report also highlights current limitations and suggests directions for future improvement.

Introduction

Machine learning helps solve many real-world problems, especially in classification tasks. This project focuses on a multiclass classification problem using data from the UCI repository. We aim to understand how basic models perform and what steps are needed to prepare and analyze such data. The project includes data exploration, model building, and evaluation to find which approach works best.

Methodology

1- Import Libraries

import numpy as np:	For numerical operations
import pandas as pd:	For data manipulation and analysis
import matplotlib.pyplot as plt:	For creating basic plots and visualizations
import seaborn as sns:	For making attractive and advanced statistical plots

2- Load Dataset

```
df = pd.read_csv(r"C:\Users\Hp Laptop\Desktop\ML Project-\UpdatedResumeDataSet.csv")
```

3- Explore Dataset

To understand more about dataset we have used some functions.

df.head()	to view the first few rows of the dataset
df.info()	to check data types and null values
df.describe()	to get statistical summaries (where applicable)
df.columns	to view column names
df.value_counts()	to count unique values
df['column'].unique()	to list unique values in a column
df.isnull().sum()	to check for missing values
df.nunique()	to count unique values per column
df.shape	to find the total number of rows and columns

We have verified that information about our dataset is correct as mentioned in the original resources by using above functions

Number of Columns:	2 (Category, Resume)
Total Rows:	962
Index Range:	0 to 961
Data Types:	Both columns are of type object
Non-null Entries:	All 962 rows are complete (no missing values)
Unique Categories:	25
Unique Resume Texts:	166
Most Frequent Category Count:	84
Most Frequent Resume Text Count:	18

4- Diagrams

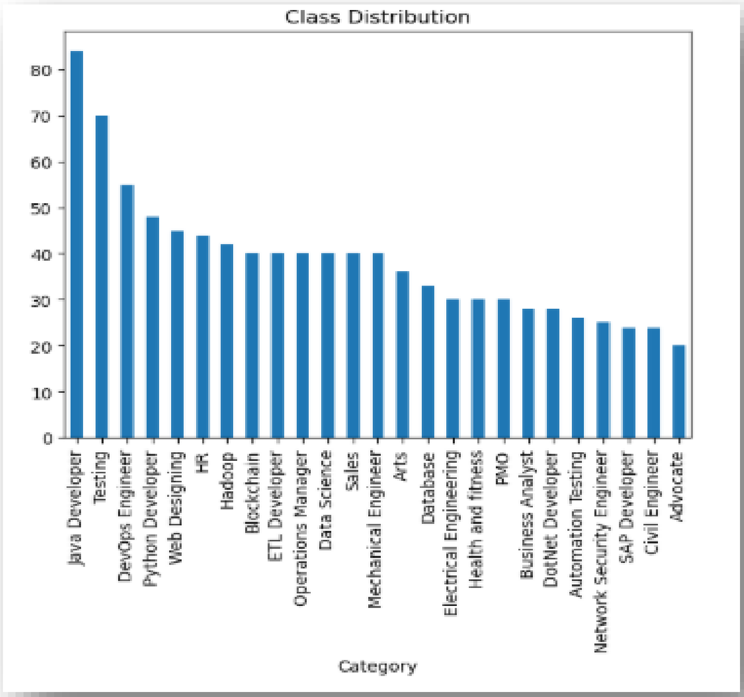
To better understand and present the structure of the dataset, we used various visualizations that highlight key patterns and distributions.

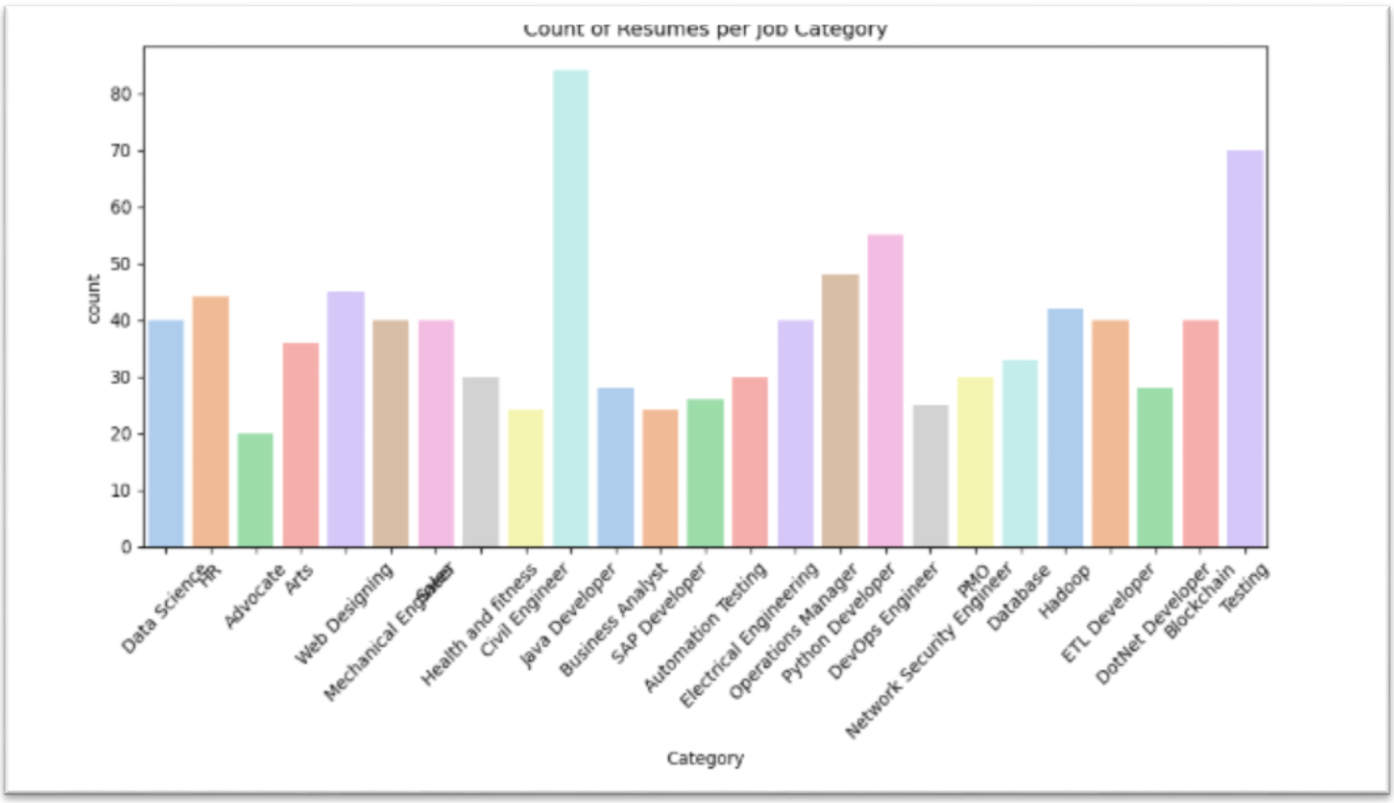
Bar Plot: To show the number of resumes in each job category.

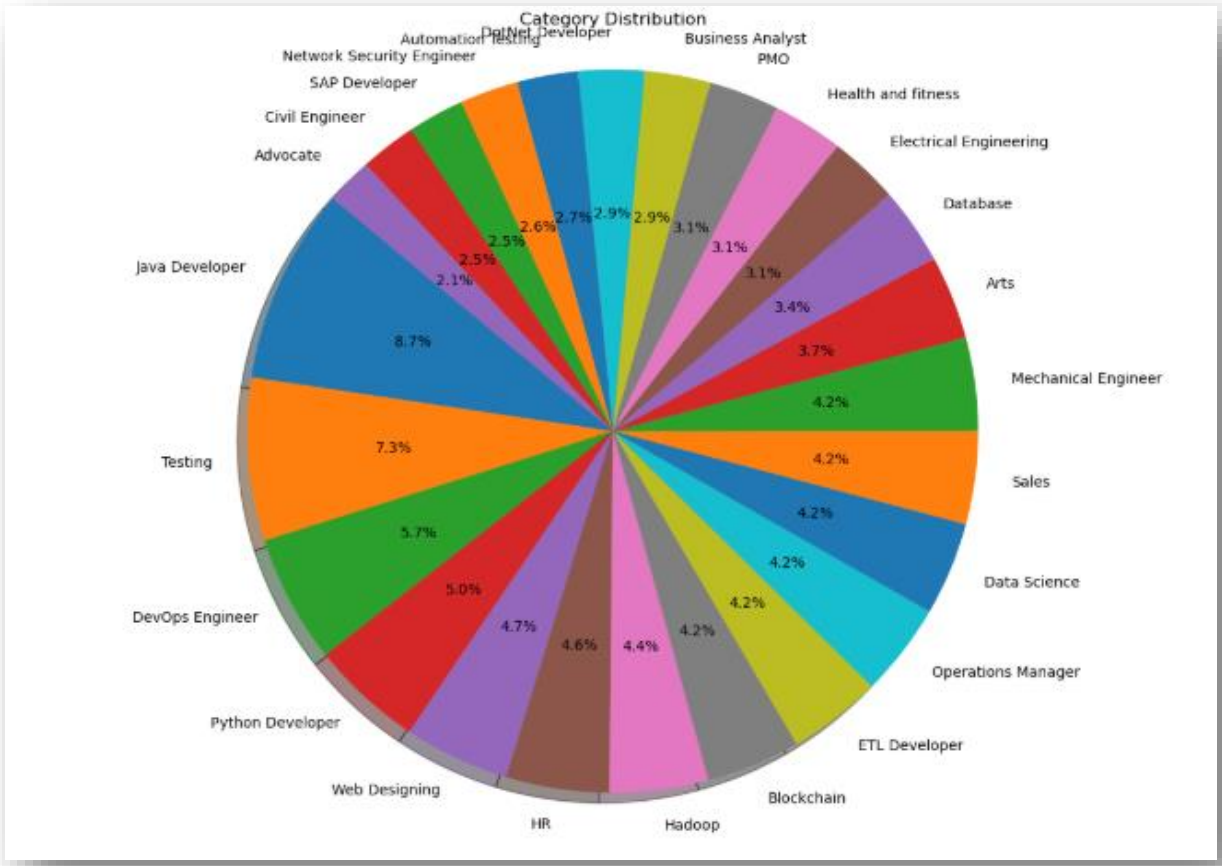
Pie Chart: Visual representation of category distribution.

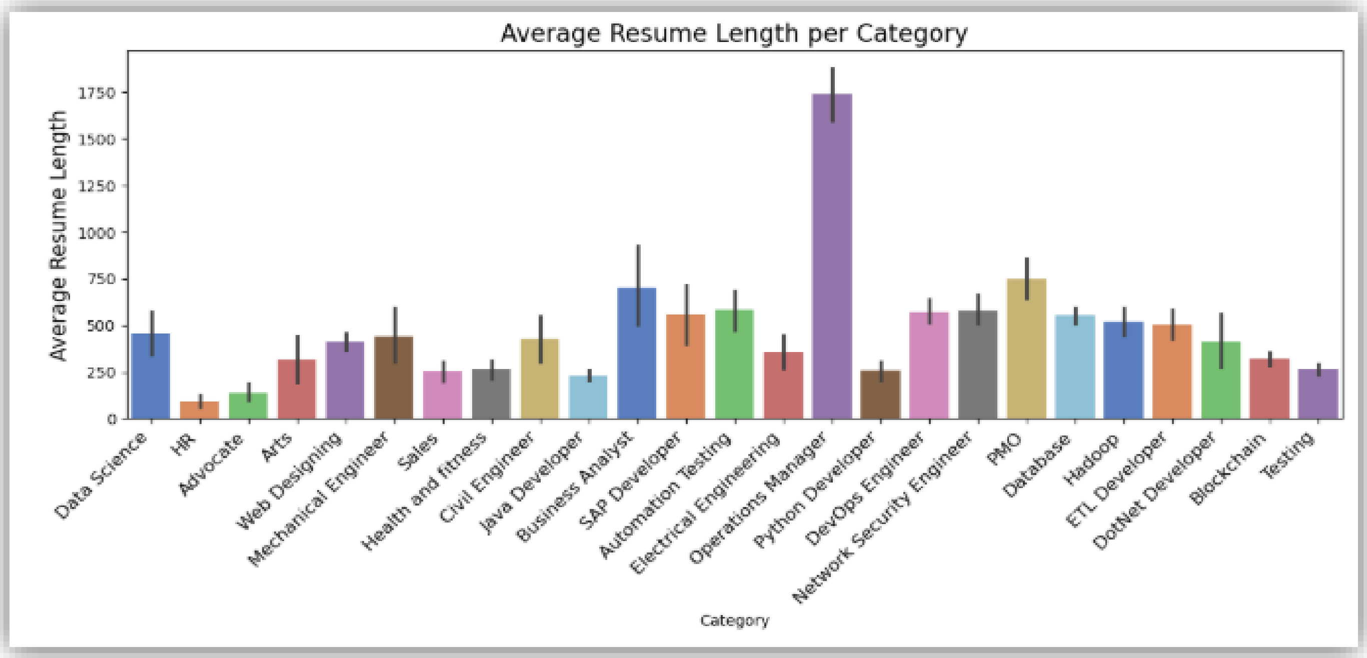
Box Plot: To examine resume length distributions.

Scatter Plot & Count Plot: To visually compare resume lengths and frequencies

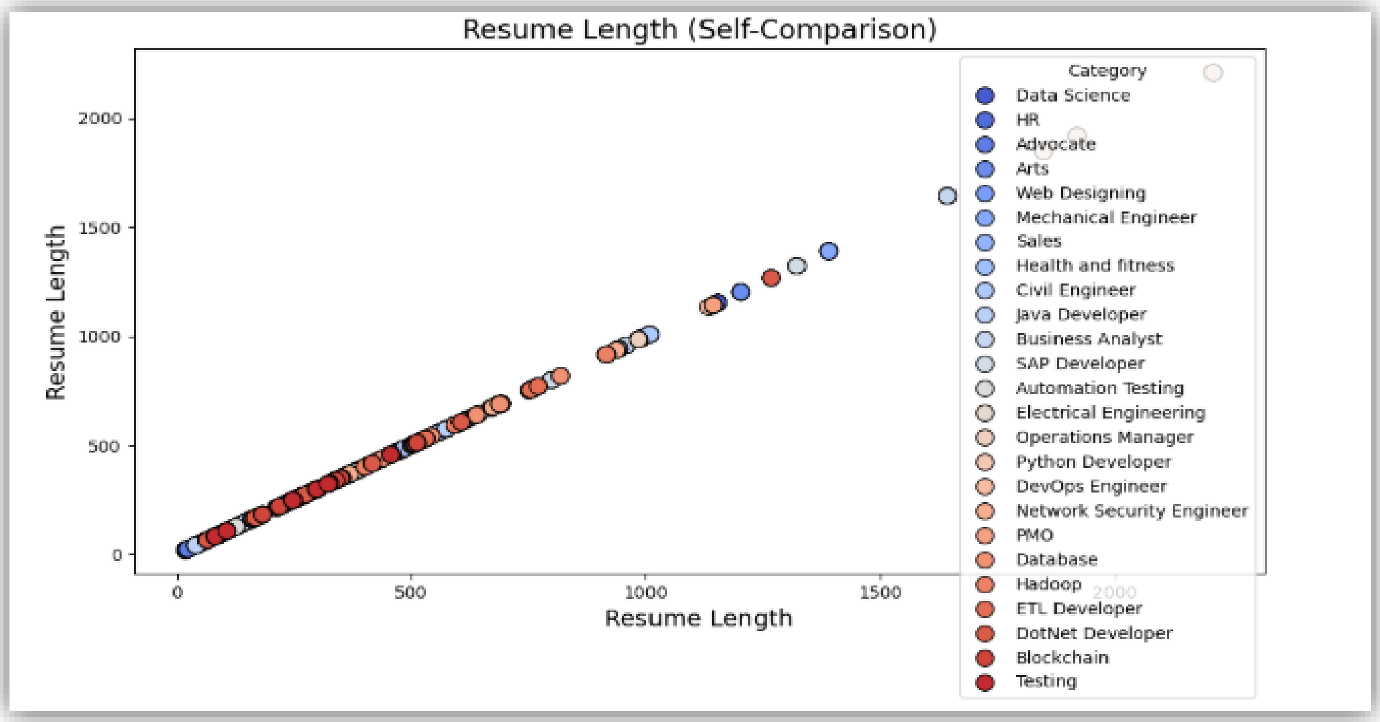



Count of Resumes per Job Category
count
Category
Data Science
HR
Advocate
Arts
Web Designing
Mechanical Engineer
Sales
Health and fitness
Civil Engineer
Java Developer
Business Analyst
SAP Developer
Automation Testing
Electrical Engineering
Operations Manager
Python Developer
DevOps Engineer
Network Security Engineer
PMO
Database
Hadoop
ETL Developer
DotNet Developer
Blockchain
Testing


Category Distribution
Java Developer 8.7%
Testing 7.3%
DevOps Engineer 5.7%
Python Developer 5.0%
Web Designing 4.7%
HR 4.6%
Hadoop 4.4%
Blockchain 4.2%
ETL Developer 4.2%
Operations Manager 4.2%
Data Science 4.2%
Sales 4.2%
Mechanical Engineer 4.2%
Arts 3.7%
Database 3.4%
Electrical Engineering 3.1%
Health and fitness 3.1%
PMO 3.1%
Business Analyst 2.9%
DotNet Developer 2.9%
Automation Testing 2.7%
Network Security Engineer 2.6%
SAP Developer 2.5%
Civil Engineer 2.5%
Advocate 2.1%



Average Resume Length per Category
Average Resume Length
0
250
500
750
1000
1250
1500
1750
Data Science
HR
Advocate
Arts
Web Designing
Mechanical Engineer
Sales
Health and fitness
Civil Engineer
Java Developer
Business Analyst
SAP Developer
Automation Testing
Electrical Engineering
Operations Manager
Python Developer
DevOps Engineer
Network Security Engineer
PMO
Database
Hadoop
ETL Developer
DotNet Developer
Blockchain
Testing
Category



Resume Length (Self-Comparison)
Resume Length
0
500
1000
1500
2000
Resume Length
0
500
1000
1500
2000
Category
Data Science
HR
Advocate
Arts
Web Designing
Mechanical Engineer
Sales
Health and fitness
Civil Engineer
Java Developer
Business Analyst
SAP Developer
Automation Testing
Electrical Engineering
Operations Manager
Python Developer
DevOps Engineer
Network Security Engineer
PMO
Database
Hadoop
ETL Developer
DotNet Developer
Blockchain
Testing

4- Resume Cleaning

Before using the resume data for training models, we cleaned the text to remove unnecessary parts that don’t help with classification.

- Removed URLs – links to websites are not useful for our model
- Removed email addresses – to clean personal information
- Removed special characters – like punctuation and symbols
- Converted text to lowercase – to keep it consistent
- Removed stopwords – common words like "and", "is", "the", which don’t add much meaning
- We saved the cleaned text in a new column called Cleaned_Resume.

5- Label Encoding

The target column in our dataset (Category) contains text labels (like "Data Science", "HR", "Software Engineer", etc.).

Since machine learning models work with numbers—not text—we need to convert these category names into numeric values.

We used Label Encoding to do this. It assigns a unique number to each job category. For example:

Data Science= 0, HR=1, Software Engineer=2 and so on

This way, the model can learn patterns based on numeric labels rather than text.

We used Label Encoder () from scikit -learn to perform this conversion and replaced the original text labels in the Category column with their corresponding numbers.

6- Text Vectorization (TF-IDF)

(Term Frequency–Inverse Document Frequency)

TF-IDF gives importance to the words that are frequent in a resume but not common across all resumes. This helps the model focus on meaningful words.

We used Tfidf Vectorizer() from scikit-learn.

It learned the vocabulary from the cleaned Resume column.

Then it transformed the text into a matrix of numbers.

This numerical data (called resume) was later used to train our classification models.

7- Preprocessing

Splitting the Data

To train and evaluate our models properly, we split the dataset into three parts:

- 70% for Training – used to train the models
- 10% for Validation – used to tune model settings (like hyperparameters)
- 20% for Testing – used to check final model performance

We used train_test_split() from scikit-learn with the stratify option to make sure each split has a similar distribution of job categories.

This helps prevent overfitting and gives us a more accurate idea of how the model will perform on unseen data.

8- Build Two to Three Models

Building the Logistic Regression Model

We used Logistic Regression, a commonly used algorithm for classification tasks. It works well for problems where the output belongs to one of several categories.

We imported LogisticRegression from scikit-learn.

The model was initialized with a max_iter of 1000 to make sure it has enough steps to converge.

This model was later trained on the training data to learn patterns from the resumes and their corresponding job categories.

Building the K-Nearest Neighbors (KNN) Model

We also used the K-Nearest Neighbors (KNN) algorithm for classification. This model predicts the category of a resume based on the categories of the closest resumes in the training data.

We imported KNeighborsClassifier from scikit-learn.

The model was initialized with n_neighbors=5, meaning it looks at the 5 nearest data points to make a prediction.

This model was then trained using the training set.

KNN is simple and works well for smaller datasets, making it a good choice for comparison with Logistic Regression.

9- Train Models on Training Data

The Logistic Regression model was trained using log_reg.fit(X_train, y_train). This step allows the model to learn patterns between resume content and job categories.

Similarly, the K-Nearest Neighbors (KNN) model was trained using knn.fit(X_train, y_train), allowing it to store the training data for future comparison during prediction.

Both models were now ready to be evaluated and tested.

10- Cross-Validation and Hyperparameter Tuning

In this step, we used cross-validation to test how well our Logistic Regression model works with specific settings (called hyperparameters).

We created a Logistic Regression model with C=0.1, penalty='l2', and solver='liblinear'.

We used K-Fold cross-validation with 5 splits.

This means the validation data was split into 5 parts. The model was trained on 4 parts and tested on the remaining part—repeating this 5 times with different parts.

This helps us check how stable and reliable our model is.

Finally, we printed the accuracy for each fold and the average accuracy, which shows how well the model performs on different slices of the data.

11- Predict Test Data

We used both the trained Logistic Regression and KNN models to make predictions on the test set.

The predicted categories for each resume were printed to compare model outputs.

Model Evaluation:

We evaluated both models using accuracy and AUC (Area Under the Curve) scores. Logistic Regression and KNN were compared based on how well they predicted the test data. We also plotted ROC curves to visualize performance.

We tested both models, and **Logistic Regression** had the best **AUC**, while **K-Nearest Neighbors (KNN)** had the highest **accuracy**.

12- Confusion Matrix

We used confusion matrices to compare actual vs. predicted values for both models. These heatmaps help us see how many predictions were correct and where the models made mistakes. Separate matrices were created for Logistic Regression and KNN.

13- Model Training Of Best chosen Model:

Logistic Regression for Resume Classification

We trained a Logistic Regression model to classify resumes into different job categories. First, we split the dataset into training and testing sets. Then, we converted the resume text into numerical form using TF-IDF vectorization. Finally, we trained the model using the transformed training data.

14- Pickle Module

After training the Logistic Regression model, we saved both the model and the TF-IDF vectorizer using the Pickle module. This allows the app to load the pre-trained models, clean the uploaded resume, predict its job category, and save it in the appropriate folder, all without retraining the models each time.

Conclusion

In the Resume Screening project, we used Logistic Regression and KNN models. While KNN achieved the highest accuracy, Logistic Regression delivered the best AUC score. Therefore, we selected Logistic Regression as the best model based on its superior AUC performance.

References

Data Set from Kaggle:
[https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset]