CS6320 - Natural Language Processing
Project Report
Resume Analyzer

https://github.com/Balaguru1601/NLP-Project.git
https://youtu.be/9wfjorL50DA

Balaguru Sethuraman - BXS230069
Somesh Kennedy - SXK240047

1. Objective

The primary objective of this project is to develop an automated system that can classify resumes into predefined job categories based on the textual content extracted from the resume documents. This can help HR professionals and recruiters efficiently filter and sort resumes, thereby streamlining the recruitment process.

In addition to resume classification, the system will offer the following key features:

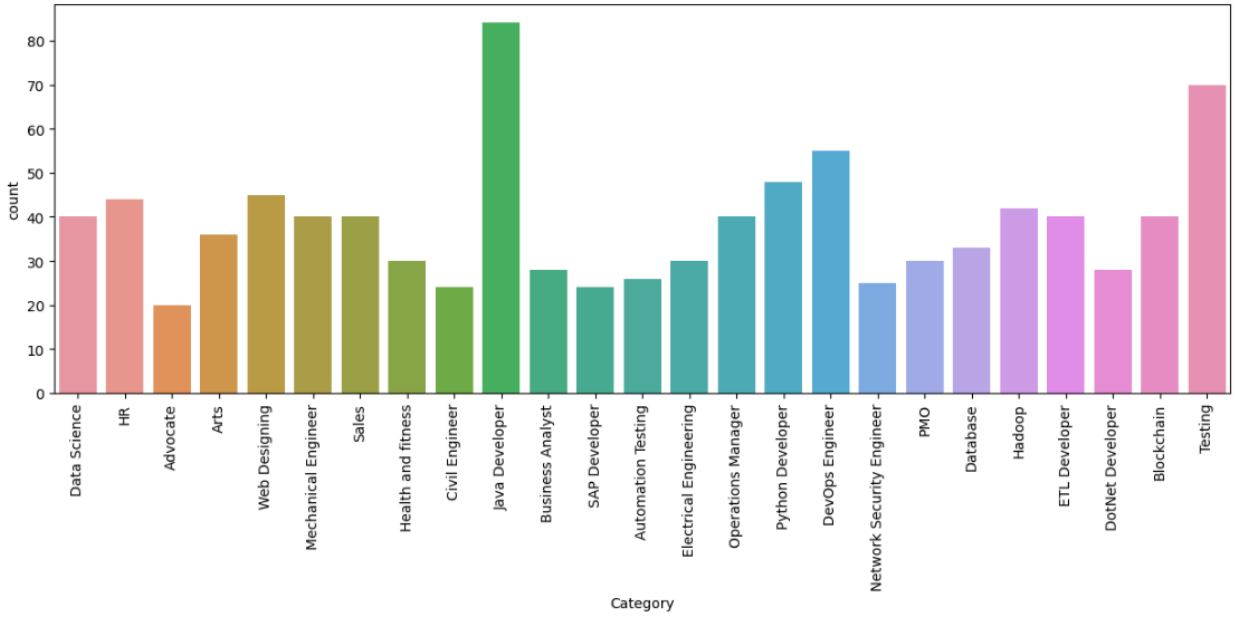
- **Resume Upload & Role Prediction**: Users can upload a resume, and the system will analyze its content to predict the most suitable job role based on the candidate's skills, qualifications, and experience.
- **Resume Creation**: Based on the predicted job role, the system will dynamically generate a professional resume tailored to that role, helping candidates present themselves more effectively.
- **Job Recommendations**: After role prediction, the system will suggest relevant job openings aligned with the identified role, offering value both to job seekers and recruiters.

2. Dataset Description

- **Dataset Name**: Resume Dataset by Gaurav Dutta
- **Format**: CSV
- **File Name**: `UpdatedResumeDataSet.csv`
- **Columns**:

- `Category`: The job role/classification the resume belongs to (e.g., 'Python Developer', 'HR', 'Data Science').
- `Resume`: The full textual content of the candidate's resume.

- **Number of Samples**: 962
- **Number of Classes**: 25 distinct categories



Dataset Distribution across the classes

3. Methodology

The project follows a structured pipeline consisting of the following key stages:

A. Data Exploration and Visualization

- Analyzed the class distribution using `value_counts()` and visualized it using Seaborn's `countplot` to identify any class imbalance in the dataset.

B. Data Cleaning and Preprocessing

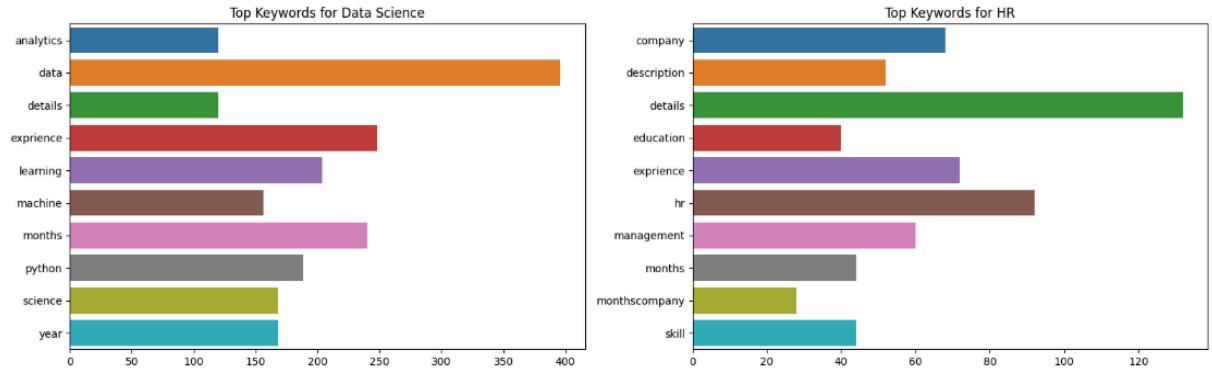
- A custom function `CleanResume()` was created to clean the text:
    - Removed URLs, mentions, hashtags, special characters, digits, and excess whitespace.
    - Converted text to lowercase for uniformity.

C. Label Encoding

- Encoded the `Category` column using `LabelEncoder` from `sklearn`, transforming text labels into numerical format required for classification models.

D. Text Vectorization using TF-IDF

- Used `TfidfVectorizer` with `stop_words='english'` to convert cleaned text into numerical feature vectors.
- This helps quantify the importance of each word relative to the document and corpus.



Result of Vectorization using TF-IDF (Examples Categories: Data Science and HR)

E. Train-Test Split

- Split the data into:

    - `X_train`, `X_test`: Input features
    - `y_train`, `y_test`: Target labels
- Used an 80-20 train-test split with `random_state=42` for reproducibility.

4. Model Development

Chosen Algorithms:

- **K-Nearest Neighbors (KNN)** – selected for simplicity and interpretability.

Model Training:

- Converted sparse TF-IDF matrices to dense arrays using `.toarray()`.
- Trained the `KNeighborsClassifier` using default parameters.

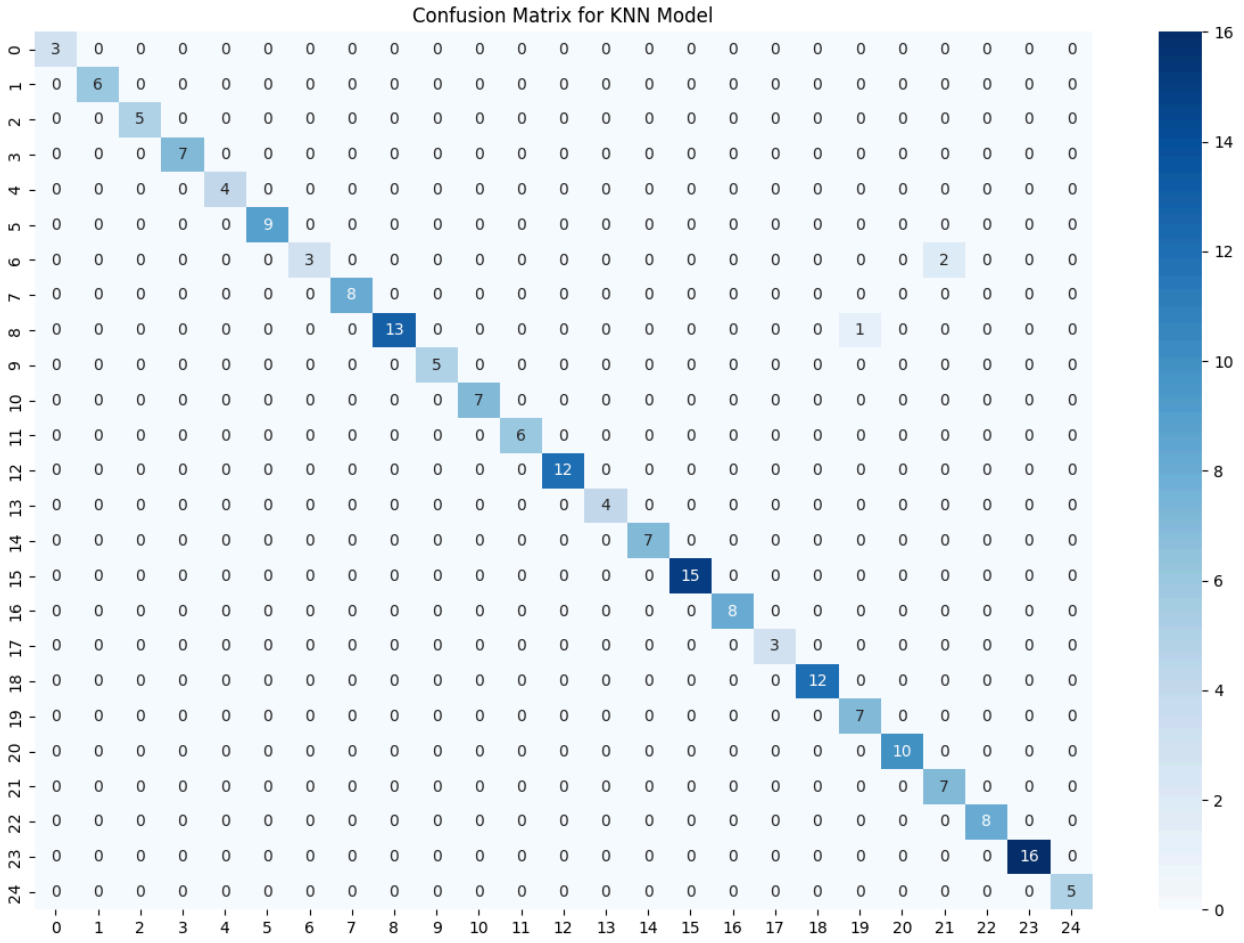
Evaluation Metrics:

- **Accuracy**: Achieved **98.45%** accuracy on the test set.

  Accuracy: 0.9845

  Macro avg F1-score: 0.9809

  Weighted avg F1-score: 0.9839
- **Classification Report**: Precision, Recall, and F1-scores were calculated per class.
- Some minor performance dips were observed in classes with fewer samples.



Confusion Matrix from the Model Evaluation (For all 25 classes compared)

5. Resume Prediction on New PDF Input

PDF Text Extraction:

- Used the `PyMuPDF` (fitz) library to extract text from PDF files.
- Created a function `extract_text_from_pdf()` to extract and concatenate text from all pages.

End-to-End Prediction Pipeline:

1. Extract raw text from uploaded PDF.
2. Clean the text using the `CleanResume()` function.
3. Vectorize using the trained `TfidfVectorizer`.
4. Predict using the trained KNN model.
5. Decode the predicted label using `LabelEncoder`.

```
pdf_path = '/kaggle/input/testing/python-developer2 - Template 18.pdf'
predicted_category = predict_resume_from_pdf(pdf_path)
print(f"Predicted Resume Category: {predicted_category}")
```

```
Predicted Resume Category: Python Developer
```

Model Deployment and Prediction on Individual New Resume

6. Model Deployment and User Interface

This phase focuses on deploying the machine learning model and integrating it into a full-stack application that provides an interactive user experience. The deployment architecture consists of a React-based client and a Flask server that hosts the backend logic and machine learning model.

Client Side (React):

The frontend is built using React, providing a user-friendly interface where users can:

- Upload their resume (in PDF or DOCX format).
- View the predicted job role based on the uploaded content.
- Download a newly generated resume tailored to the predicted role.

- Browse relevant job listings associated with the predicted role.

Server Side (Flask):

The backend server, developed in Flask, manages the processing pipeline and integrates several key components:

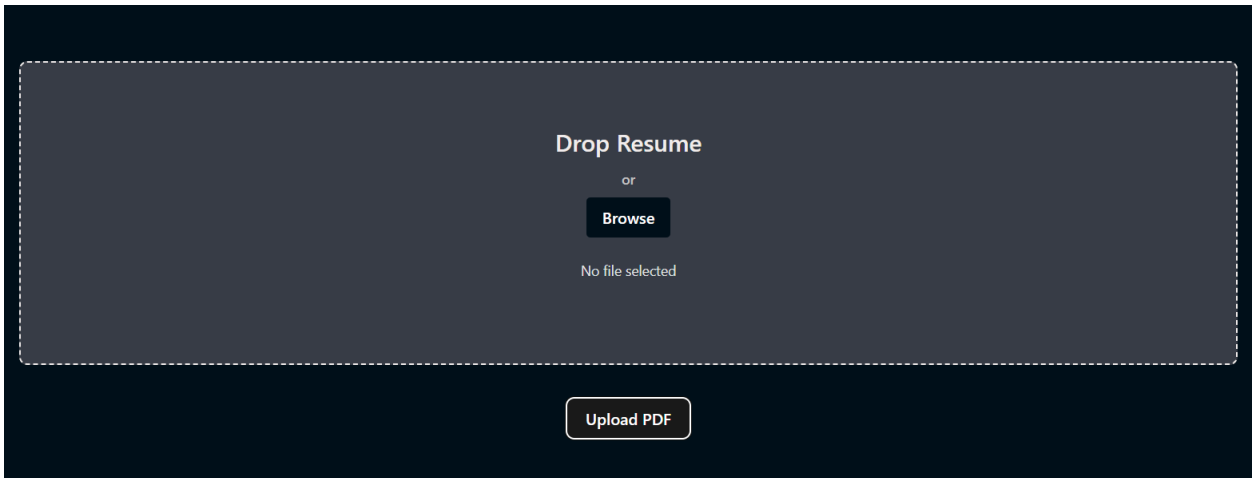
- **Resume Processing and Role Prediction:** Upon resume upload, the text is extracted and processed using natural language processing techniques. A **K-Nearest Neighbors (KNN)** classifier is used to predict the most suitable job role based on the resume's content.

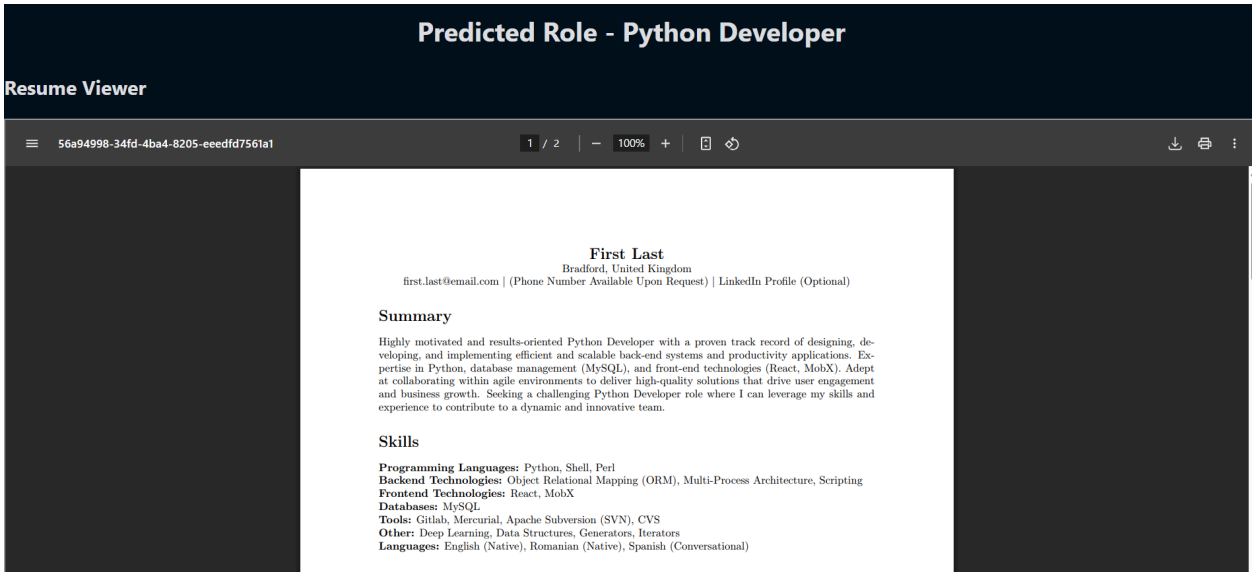
- **Resume Generation with LLM:** Once the role is predicted, the extracted resume content and the predicted role are sent to **Google's Gemini LLM**, which generates a new, professional resume tailored to the target job. The LLM responds in **LaTeX** format to ensure high-quality formatting.

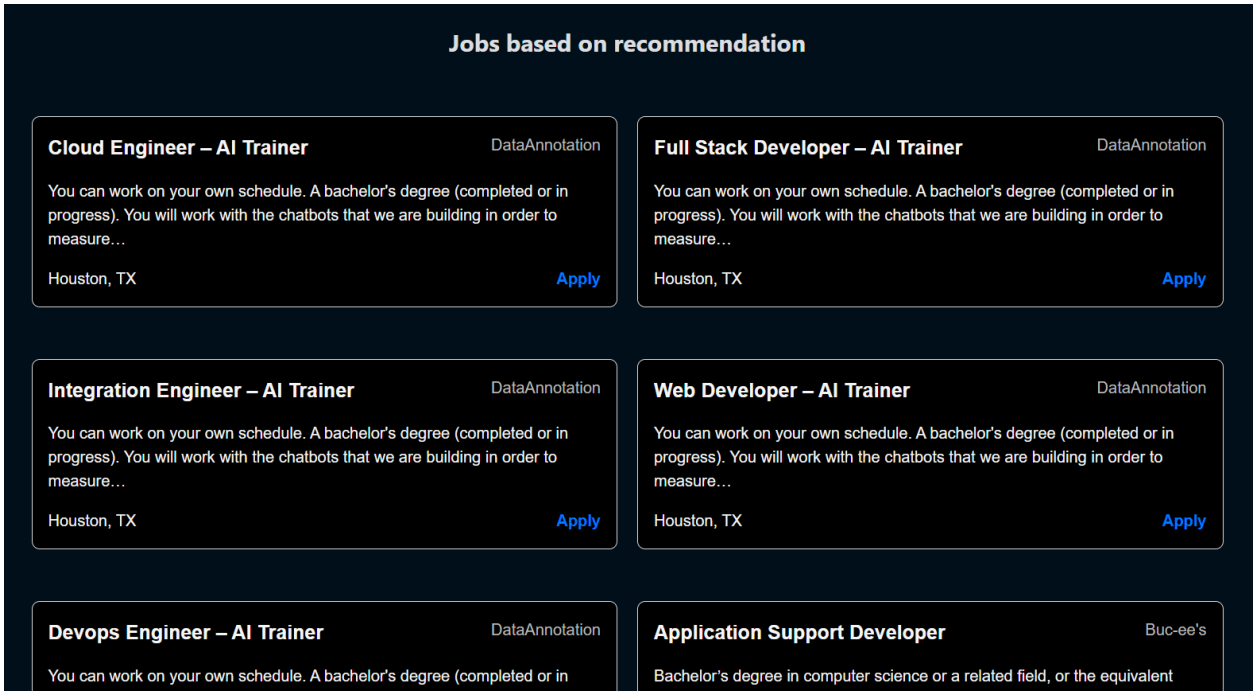
- **PDF Conversion:** The LaTeX output is compiled into a PDF document, which is then returned to the client for download.

- **Job Scraping:** To enrich the experience, the system dynamically scrapes current job listings from online job boards based on the predicted role and includes them in the response to the client.

This integrated architecture enables a seamless pipeline from resume upload to intelligent content generation and job recommendation, making the system valuable for both job seekers and recruiters.







7. Strengths & Limitations

✅ Strengths:

- High classification accuracy.
- Clean and modular pipeline.

- PDF-based resume prediction with dynamic input handling.

⚠️ **Limitations:**

- Slight class imbalance (some job roles have fewer examples).
- KNN requires converting sparse matrices to dense, which could be memory-intensive on larger datasets.

8. Future Improvements

- Use **word embeddings** (Word2Vec, GloVe, or BERT) for semantic understanding.
- Try deep learning models (LSTM, BERT-based classifiers).
- Handle imbalanced data with techniques like SMOTE or class weighting.
- Add **NER (Named Entity Recognition)** for more granular resume parsing (e.g., extract skills, experience).

9. Conclusion

This project demonstrates the power of NLP in automating resume classification. By leveraging traditional machine learning with TF-IDF and KNN, the model achieves high accuracy and generalizes well to unseen PDF resumes as well as display suitable job recommendations. Furthermore, using Gemini LLM, we can generate resumes based on predicted job types. With further enhancements, it can serve as a robust tool in HR automation and recruitment workflows.

10. Tasks by each member

Balaguru Sethuraman - UI creation, LLM integration, Server setup

Somesh Kennedy - Dataset collection and Analysis, Model fine tuning