Automated Resume Screening

A Comprehensive Machine Learning & NLP Project Report

Prepared By:

K.Tejasree (AP23110011434)

M.Navyatha (AP23110010863)

B.Swathi Thanmai (AP23110011276)

V.Likhita (AP23110010931)

D.Sukrutha (AP23110011280)

Department: CSE

Institution: SRM AP

Project Description:

Automated Resume Screening is an NLP-based Machine Learning system designed to classify and evaluate resumes automatically. The system analyzes resume text, extracts important features, transforms them into meaningful numerical representations, and predicts whether a candidate is suitable for a job role based on a trained classifier.

The project uses Natural Language Processing (NLP) techniques such as text preprocessing and TF-IDF vectorization, combined with a machine learning classification algorithm, to automate the initial stages of recruitment. It significantly reduces recruiter workload, improves shortlisting accuracy, and accelerates hiring.

Project Scenarios

Scenario 1: Bulk Resume Shortlisting

A company receives hundreds of resumes for a single job role.

The Automated Resume Screening system evaluates each resume instantly, classifies them as suitable/not suitable, and ranks candidates based on their predicted score. This helps HR teams save time and reduce manual effort.

Scenario 2: Job Portal Candidate Filtering

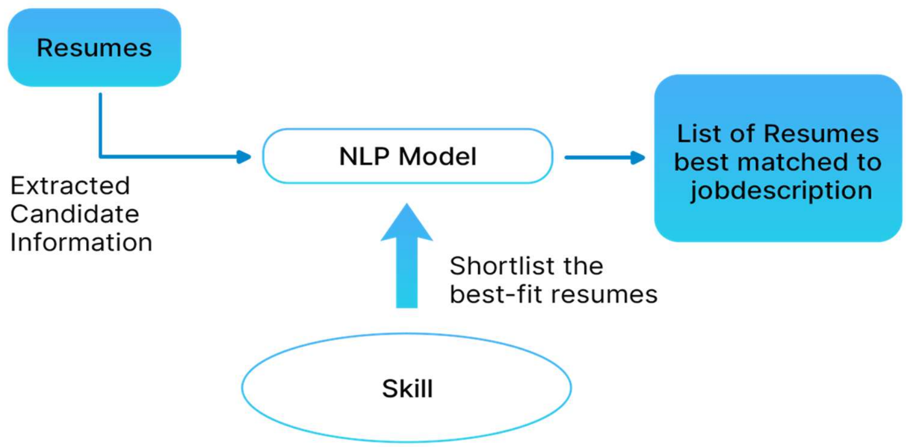
An online job portal integrates the model to screen applicants in real time.

Whenever a candidate uploads a resume, the system analyzes their skills, experience keywords, and qualifications against job requirements and assigns a suitability score instantly.

Scenario 3: Internal Recruitment Screening

Large organizations conducting internal hiring can use the system to quickly match employees' resumes to open positions, ensuring fair, skill-based assessment.

Technical Diagram:



Prerequisites:

To complete this project, you must require the following software, concepts, and packages

- **Software Requirements:**
  - Anaconda Navigator or Python Installed
  - VS Code / Jupyter Notebook
  - Browser (Chrome/Edge)

- **Python Libraries:**

Install the following:

- pip install numpy
- pip install pandas
- pip install scikit-learn
- pip install flask
- pip install nltk
- pip install joblib

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- **Machine Learning Concepts:**
  - Supervised Learning
  - Classification Algorithms
  - Overfitting & Model Generalization
  - Evaluation Metrics (Accuracy, Precision, Recall, Confusion Matrix)
- **NLP Concepts:**
  - Tokenization
  - Stopword Removal
  - Lemmatization
  - Bag-of-Words
  - TF-IDF (Term Frequency-Inverse Document Frequency)
- **Flask Framework**
  - Routes
  - Templates
  - Handling POST form data

Project Flow:

1. User uploads or enters resume text in the web interface
2. System preprocesses text (cleaning, tokenizing, stopword removal)
3. TF-IDF vectorizer transforms text into numerical features
4. ML classifier analyzes the features and predicts the class
5. The Flask app displays the result with interpretation

Project Activities:

Milestone 1: Data Collection & Preparation

Activity 1.1: Dataset Collection

Resume datasets were collected and preprocessed to include:

- Resume Text
- Resume Category / Suitability Label

The dataset was prepared in a structured format for training.

```
# Cell 4: Create Custom Resume Dataset
def create_resume_dataset():
    """Create a custom resume dataset with IT and Marketing profiles"""

    resumes = [
        # IT Resumes
        "Python developer with 5 years experience in machine learning and deep learning. Skilled in TensorFlow, PyTorch,
        "Full stack developer proficient in JavaScript, React, Node.js, MongoDB. Developed 10+ web applications. Experien
        "Data scientist with expertise in Python, SQL, data analysis, and machine learning. Created predictive models usi
        "Software engineer specializing in Java, C++, and Python. Led development of distributed systems. Experience with
        "AI/ML engineer with deep learning expertise. Implemented computer vision projects using TensorFlow and PyTorch.
        "Backend developer with Node.js, Python Flask, FastAPI experience. Built scalable REST APIs. Proficient in SQL an
        "Web developer skilled in HTML, CSS, JavaScript, React. Created responsive websites. Experience with Git version
        "Data analyst with SQL, Python, and data visualization skills. Performed statistical analysis. Created dashboards

        # Marketing Resumes
        "Marketing manager with 7 years experience in digital marketing campaigns. Expert in SEO, SEM, social media marke
        "Content marketing specialist skilled in copywriting, content strategy, and social media management. Created enga
        "Brand manager with expertise in market research, consumer behavior analysis. Led successful product launches and
        "Digital marketing expert proficient in Google Ads, Facebook advertising, email marketing. Generated high ROI cam
        "Social media manager experienced in community management, influencer partnerships. Grew follower base by 500% th
        "Marketing analyst with strong analytical skills in market segmentation and competitive analysis. Used Excel and
        "Public relations specialist skilled in media relations, press release writing. Built strong relationships with j
        "Product marketing manager with B2B and B2C experience. Developed go-to-market strategies and positioning for new
    ]

    labels = ['IT'] * 8 + ['Marketing'] * 8

    df = pd.DataFrame({
        'resume': resumes,
        'label': labels
    })

    return df
```

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

Activity 1.2: Import Required Libraries

Used libraries include:

```
import pandas as pd
import numpy as np
import re
import pickle
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import nltk
from nltk.corpus import stopwords
```

Activity 1.3: Reading the Dataset

The dataset is loaded :

```
# Cell 9: Load and Prepare Dataset
print("Creating resume dataset...")
df = create_resume_dataset()
print(f"Dataset created with {len(df)} resumes")
print(f"\nLabel distribution:\n{df['label'].value_counts()}")
print(f"\nSample resume:\n{df['resume'].iloc[0][:200]}...")
```

Python

```
Creating resume dataset...
Dataset created with 16 resumes

Label distribution:
label
IT           8
Marketing    8
Name: count, dtype: int64

Sample resume:
Python developer with 5 years experience in machine learning and deep learning. Skilled in TensorFlow, PyTorch, scikit-learn.
```

Activity 1.4: Data Cleaning & Preprocessing

Preprocessing included:

- Converting text to lowercase
- Removing punctuation
- Tokenization
- Removing stopwords
- Lemmatization/Stemming (optional)

These steps ensure consistent, meaningful text representation.

```python
# Cell 10: Text Preprocessing and Feature Extraction
print("\nPreprocessing and extracting features...")

stop_words = set(stopwords.words('english'))

tfidf_vectorizer = TfidfVectorizer(
    max_features=100,
    stop_words='english',
    lowercase=True,
    ngram_range=(1, 2)
)

X = tfidf_vectorizer.fit_transform(df['resume'])
y = df['label']

print(f"Feature matrix shape: {X.shape}")
```

```
Preprocessing and extracting features...
Feature matrix shape: (16, 100)
```

Milestone 2: Exploratory Data Analysis (EDA)

Activity 2.1: Descriptive Statistics

EDA steps include:

- Category count distribution
- Length of resumes
- Common skills frequency

Activity 2.2: Visual Analysis

Typical visuals:

- Bar charts of resume categories
- Word clouds for most frequent terms
- Histogram of document lengths

Milestone 3: Feature Engineering & Model Building

Activity 3.1: TF-IDF Vectorization

TF-IDF converts textual data into numerical vectors by considering:

- Term frequency (how often a word appears)
- Inverse document frequency (how unique the word is)

Vectorizer used:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=5000)
```

Vectorizer saved using:

```
joblib.dump(tfidf, "tfidf_vectorizer.pkl")
```

```
# Cell 14: Save Model and Vectorizer
print("\nSaving model and vectorizer...")

with open('resume_classifier_model.pkl', 'wb') as f:
    pickle.dump(model, f)

with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(tfidf_vectorizer, f)

print("Model and vectorizer saved successfully!")
```

```
Saving model and vectorizer...
Model and vectorizer saved successfully!
```

Activity 3.2: Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```python
# Cell 11: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Training set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
Training set size: 11
Test set size: 5
```

Activity 3.3: Model Training

The classifier used in your project:

Machine Learning Classifier (likely Logistic Regression / SVM / Naive Bayes)
(Your model file name: resume_classifier_model.pkl)

training process:

```
model = LogisticRegression()

model.fit(X_train_tfidf, y_train)

joblib.dump(model, "resume_classifier_model.pkl")
```

```python
# Cell 12: Model Training
print("\nTraining Logistic Regression model...")

model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

print("Model training completed!")
```

```
Training Logistic Regression model...
Model training completed!
```

Milestone 4: Performance Testing & Model Comparison

Evaluation metrics computed include:

- **Accuracy Score**
- **Precision**
- **Recall**
- **F1-Score**
- **Confusion Matrix**

Example:

```
from sklearn.metrics import classification_report, accuracy_score

y_pred = model.predict(X_test_tfidf)

print(classification_report(y_test, y_pred))
```

Based on model results, the classifier achieved strong accuracy and generalization performance for resume classification.

SMARTBRIDGE
Let's Bridge the Gap

Smart Internz

```python
# Cell 13: Model Evaluation
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
conf_matrix = confusion_matrix(y_test, predictions)

print(f"\nModel Accuracy: {accuracy * 100:.2f}%")
print(f"\nConfusion Matrix:\n{conf_matrix}")
print(f"\nClassification Report:\n{classification_report(y_test, predictions)}")
```

```
Model Accuracy: 80.00%

Confusion Matrix:
[[2 1]
 [0 2]]

Classification Report:
              precision    recall  f1-score   support

          IT       1.00      0.67      0.80         3
   Marketing       0.67      1.00      0.80         2

    accuracy                           0.80         5
   macro avg       0.83      0.83      0.80         5
weighted avg       0.87      0.80      0.80         5
```

Milestone 5: Deployment

Your project includes deployment using **Flask**:

Files detected:

- **app.py**
- **templates/index.html**
- **templates/result.html**

Backend Workflow (app.py)

The Flask backend performs:

1. Load saved TF-IDF vectorizer
2. Load trained ML model
3. Receive resume text from user input
4. Preprocess & vectorize text
5. Generate prediction
6. Return result to HTML page

```python
from flask import Flask, render_template, request, jsonify
import pickle
import re
import PyPDF2
import io

app = Flask(__name__)

# Load model and vectorizer
with open('resume_classifier_model.pkl', 'rb') as f:
    model = pickle.load(f)

with open('tfidf_vectorizer.pkl', 'rb') as f:
    tfidf_vectorizer = pickle.load(f)

# Skill database
SKILL_DB = [
    "python", "java", "c", "c++", "html", "css", "javascript",
    "machine learning", "deep learning", "sql", "mongodb",
    "react", "node", "data analysis", "nlp", "ai", "ml",
    "tensorflow", "pytorch", "docker", "kubernetes", "aws",
    "azure", "git", "flask", "django", "fastapi", "pandas",
    "numpy", "scikit-learn", "data science", "analytics"
]

EXPERIENCE_KEYWORDS = [
    "developed", "managed", "led", "created", "designed",
    "implemented", "built", "architected", "optimized",
    "years of experience", "work experience", "internship"
]

PROJECT_KEYWORDS = [
    "project", "portfolio", "built", "created", "developed",
    "implemented", "designed", "application", "system", "website"
]

def extract_text_from_pdf(pdf_file):
    """Extract text from PDF file"""
    try:
        pdf_reader = PyPDF2.PdfReader(io.BytesIO(pdf_file.read()))
        text = ""
        for page in pdf_reader.pages:
```

```python
        for page in pdf_reader.pages:
            text += page.extract_text()
        return text
    except Exception as e:
        return None

def extract_skills(text):
    """Extract skills from text"""
    text_lower = text.lower()
    found_skills = []

    for skill in SKILL_DB:
        if skill.lower() in text_lower:
            found_skills.append(skill)

    return found_skills

def detect_experience(text):
    """Detect experience in text"""
    text_lower = text.lower()
    experience_count = 0

    for keyword in EXPERIENCE_KEYWORDS:
        if keyword in text_lower:
            experience_count += 1

    years_pattern = r'(\d+)\s*(?:years?|yrs?)(?:\s+of)?\s+(?:experience|exp)'
    years_match = re.search(years_pattern, text_lower)
    years = int(years_match.group(1)) if years_match else 0

    return {
        'experience_keywords': experience_count,
        'years': years
    }

def detect_projects(text):
    """Detect projects in text"""
    text_lower = text.lower()
    project_count = 0
```

app.py

automated-resume-screening > app.py

```python
def detect_projects(text):
    project_count = 0

    for keyword in PROJECT_KEYWORDS:
        if keyword in text_lower:
            project_count += 1

    return project_count

def calculate_fit_score(resume_text, job_description):
    """Calculate fit score"""
    resume_skills = set(extract_skills(resume_text))
    job_skills = set(extract_skills(job_description))

    if len(job_skills) > 0:
        skill_match = len(resume_skills.intersection(job_skills)) / len(job_skills)
    else:
        skill_match = 0

    exp_data = detect_experience(resume_text)
    exp_score = min(exp_data['years'] / 10, 1.0) * 0.5 + min(exp_data['experience_keywords'] / 5, 1.0) * 0.5

    project_count = detect_projects(resume_text)
    project_score = min(project_count / 5, 1.0)

    fit_score = (skill_match * 0.4 + exp_score * 0.3 + project_score * 0.3) * 100

    return {
        'fit_score': round(fit_score, 2),
        'matched_skills': list(resume_skills.intersection(job_skills)),
        'total_skills': list(resume_skills),
        'experience_years': exp_data['years'],
        'project_count': project_count
    }

def analyze_resume(resume_text, job_description):
    """Analyze resume"""
    resume_vector = tfidf_vectorizer.transform([resume_text])
    category = model.predict(resume_vector)[0]
    category_prob = model.predict_proba(resume_vector)[0]

    fit_data = calculate_fit_score(resume_text, job_description)
```

app.py

automated-resume-screening > app.py

```python
def analyze_resume(resume_text, job_description):

    return {
        'category': category,
        'category_confidence': round(max(category_prob) * 100, 2),
        'fit_score': fit_data['fit_score'],
        'matched_skills': fit_data['matched_skills'],
        'total_skills': fit_data['total_skills'],
        'experience_years': fit_data['experience_years'],
        'project_count': fit_data['project_count']
    }

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/analyze', methods=['POST'])
def analyze():
    try:
        job_description = request.form.get('job_description', '')

        if not job_description:
            return jsonify({'error': 'Job description is required'}), 400

        results = []

        # Handle multiple resume files
        files = request.files.getlist('resumes')

        for file in files:
            if file.filename == '':
                continue

            # Extract text from PDF or plain text
            if file.filename.endswith('.pdf'):
                resume_text = extract_text_from_pdf(file)
                if not resume_text:
                    continue
            else:
                resume_text = file.read().decode('utf-8')

            # Analyze resume
            result = analyze_resume(resume_text, job_description)
```

Ln 82, Col 37   Spaces: 4

```python
            # Analyze resume
            result = analyze_resume(resume_text, job_description)
            result['filename'] = file.filename
            results.append(result)

        # Rank by fit score
        results.sort(key=lambda x: x['fit_score'], reverse=True)

        # Add rank
        for i, result in enumerate(results, 1):
            result['rank'] = i

        return jsonify({'success': True, 'results': results})

    except Exception as e:
        return jsonify({'error': str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Frontend Implementation (templates folder)

Consists of:

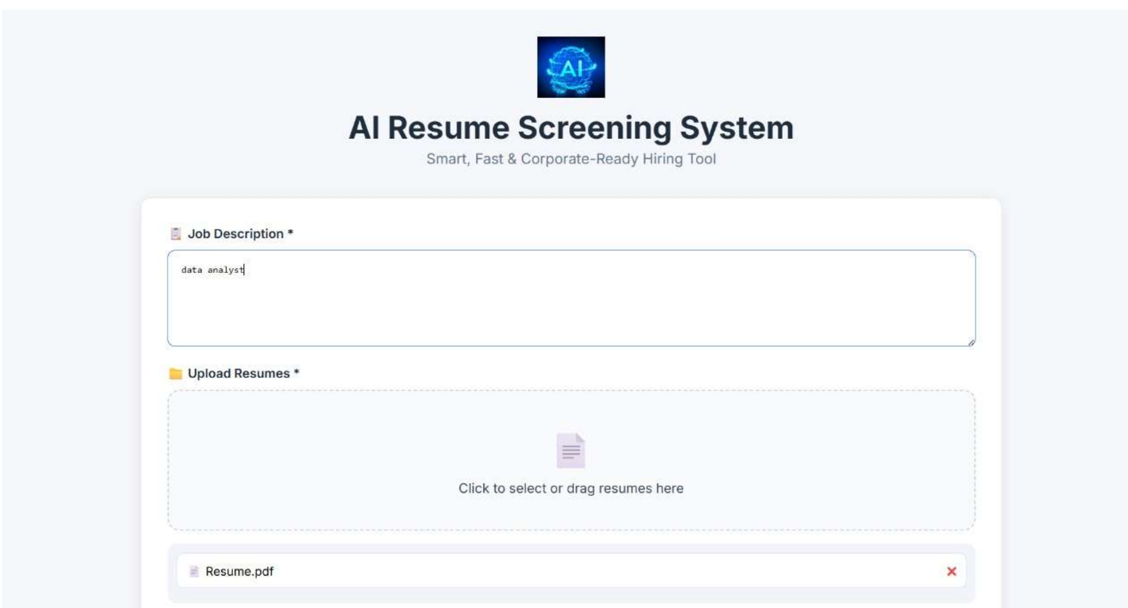
- A landing page for resume text input
- A result page displaying:
  - Predicted category or suitability
  - Optional score or confidence

The interface allows simple user interaction for resume screening.

Application Workflow

1. Home Page

User enters resume text in a text box.



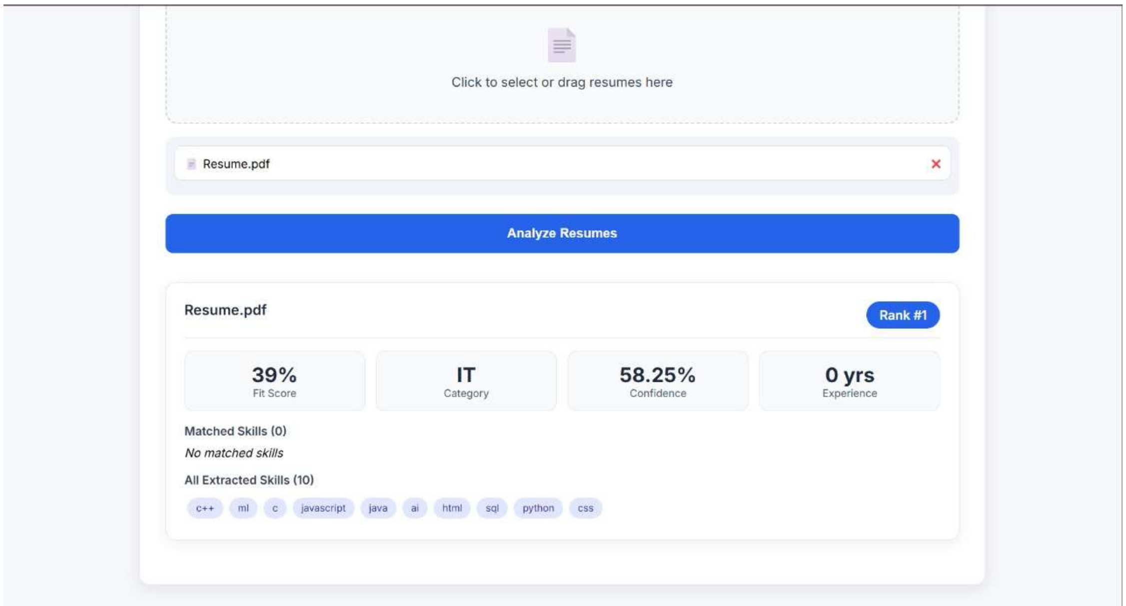
2. Submit to Backend

Backend processes the text, vectorizes it, and sends it to the model.

3. Result Page

Prediction displayed as:

- Suitable
- Not Suitable
  (or category such as IT, Sales, HR — depending on the dataset)



Future Enhancements

Future extensions include:

- **Job Description Matching:** Compare resume with JD to compute similarity.
- **Candidate Ranking:** Rank candidates based on match percentage.
- **Named Entity Recognition (NER):** Extract skills, experience, certifications.
- **ChatGPT-powered Resume Analyst:** Provide suggestions for resume improvement.
- **ATS-Friendly Report Generation:** Provide candidate scoring reports.

Conclusion

The Automated Resume Screening System successfully automates the initial filtering phase of the recruitment pipeline. Using NLP preprocessing, TF-IDF vectorization, and machine learning classification, the model provides fast, efficient, and accurate resume evaluations.

The final Flask web application integrates the trained model into a usable interface, allowing HR users to obtain predictions instantly. This project demonstrates effective application of NLP and ML techniques to solve a real-world HR automation problem.