Resume Classifier Using Machine Learning

AI-Powered Resume Classification System

Technical Report

Veridia Internship Project

November 9, 2025

Abstract

This report presents the development of an automated resume classification system using Machine Learning techniques. The system classifies resumes into 24 professional categories with a good accuracy, utilizing Natural Language Processing (NLP) for text preprocessing and Random Forest algorithm for classification. The implementation includes a complete REST API built with FastAPI and an interactive web interface for real-time testing.

Contents

1 Introduction

1.1 Project Overview

The Resume Classifier is an automated system designed to categorize professional resumes into specific job categories. This system addresses the challenge faced by HR departments and Applicant Tracking Systems (ATS) in efficiently sorting and categorizing large volumes of resumes.

1.2 Objectives

- Develop a machine learning model capable of accurately classifying resumes into 24+ professional categories
- Implement robust text preprocessing using NLP techniques
- Create a production-ready REST API for model deployment
- Build an intuitive web interface for easy testing and demonstration
- Achieve good accuracy and F1-score metrics on test data

1.3 Dataset

The project uses the **Resume Dataset** from Kaggle[1], which contains:

- **Total samples:** 2,484 resumes
- **Categories:** 24 professional fields
- **Features:** Resume text (plain and HTML format), unique ID, and category label
- **Data completeness:** 100% (no missing values)

[1] https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset/data

2 Tech Stack and Frameworks

2.1 Core Technologies

2.1.1 Backend Framework

- **FastAPI (latest):** Modern, high-performance web framework for building APIs
- **Uvicorn:** Lightning-fast ASGI server implementation
- **Pydantic:** Data validation using Python type annotations

2.1.2 Machine Learning Stack

- **scikit-learn:** Machine learning algorithms and evaluation metrics
  - Random Forest Classifier
  - CountVectorizer & TfidfVectorizer
  - GridSearchCV for hyperparameter tuning
  - Cross-validation utilities
- **imbalanced-learn:** Data balancing techniques (RandomUnderSampler, RandomOverSampler)

2.1.3 Natural Language Processing

- **NLTK (Natural Language Toolkit):**
  - Tokenization (word_tokenize)
  - Stopwords removal (English & Spanish)
  - Lemmatization (WordNetLemmatizer)

2.1.4 Data Processing

- **pandas:** Data manipulation and analysis
- **numpy:** Numerical computing
- **PyPDF2:** PDF text extraction for file uploads

2.1.5 Visualization and Development

- **matplotlib & seaborn:** Data visualization for exploratory analysis
- **Jupyter Notebook:** Interactive development and experimentation

2.1.6 Frontend

- **HTML5/CSS3:** Modern, responsive web interface
- **Vanilla JavaScript:** Client-side logic and API communication
- **Fetch API:** Asynchronous HTTP requests

2.2 Project Architecture

The system follows a modular architecture with clear separation of concerns:

```
app/
        main.py                  # FastAPI app & endpoints
        train_pipeline.py        # Complete training pipeline
        preprocessing.py         # Text preprocessing functions
        schemas.py              # Pydantic models
        config.py               # Configuration parameters
        utils.py                # Utility functions
        models/                 # Serialized ML models
            vectorizer.pkl
            model.pkl
```

Listing 1: Project Structure

3 Methodology and Approach

3.1 Data Understanding and Exploration

3.1.1 Dataset Structure

The initial exploration revealed the following characteristics:

Attribute	**Value**
Total Samples	2,484
Number of Features	4 (ID, Resume_str, Resume_html, Category)
Target Categories	24
Missing Values	0 (100% complete)
Duplicate Rows	0
Unique Resumes	2,482

Table 1: Dataset Characteristics

3.1.2 Resume Text Statistics

Metric	**Characters**	**Words**
Mean	6,295.31	811.33
Median	5,886.50	757.00
Std. Deviation	2,769.25	371.01
Minimum	21.00	0.00
Maximum	38,842.00	5,190.00

Table 2: Resume Text Length Statistics

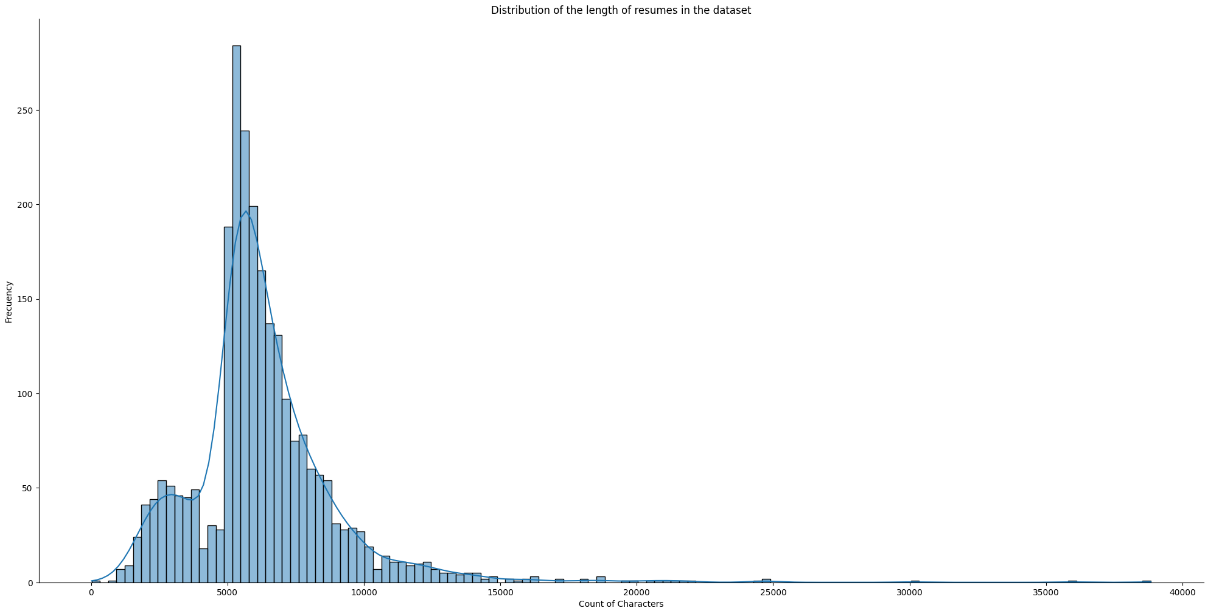

Figure 1: Distribution of resume lengths in characters

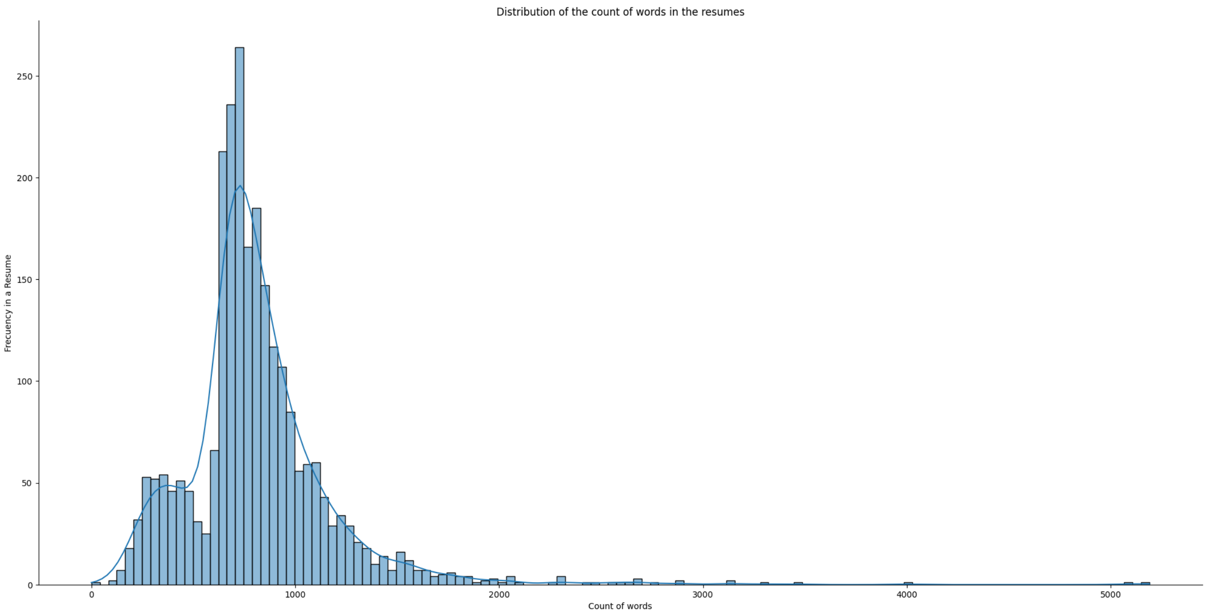

Figure 2: Distribution of word counts in resumes

3.1.3 Category Distribution

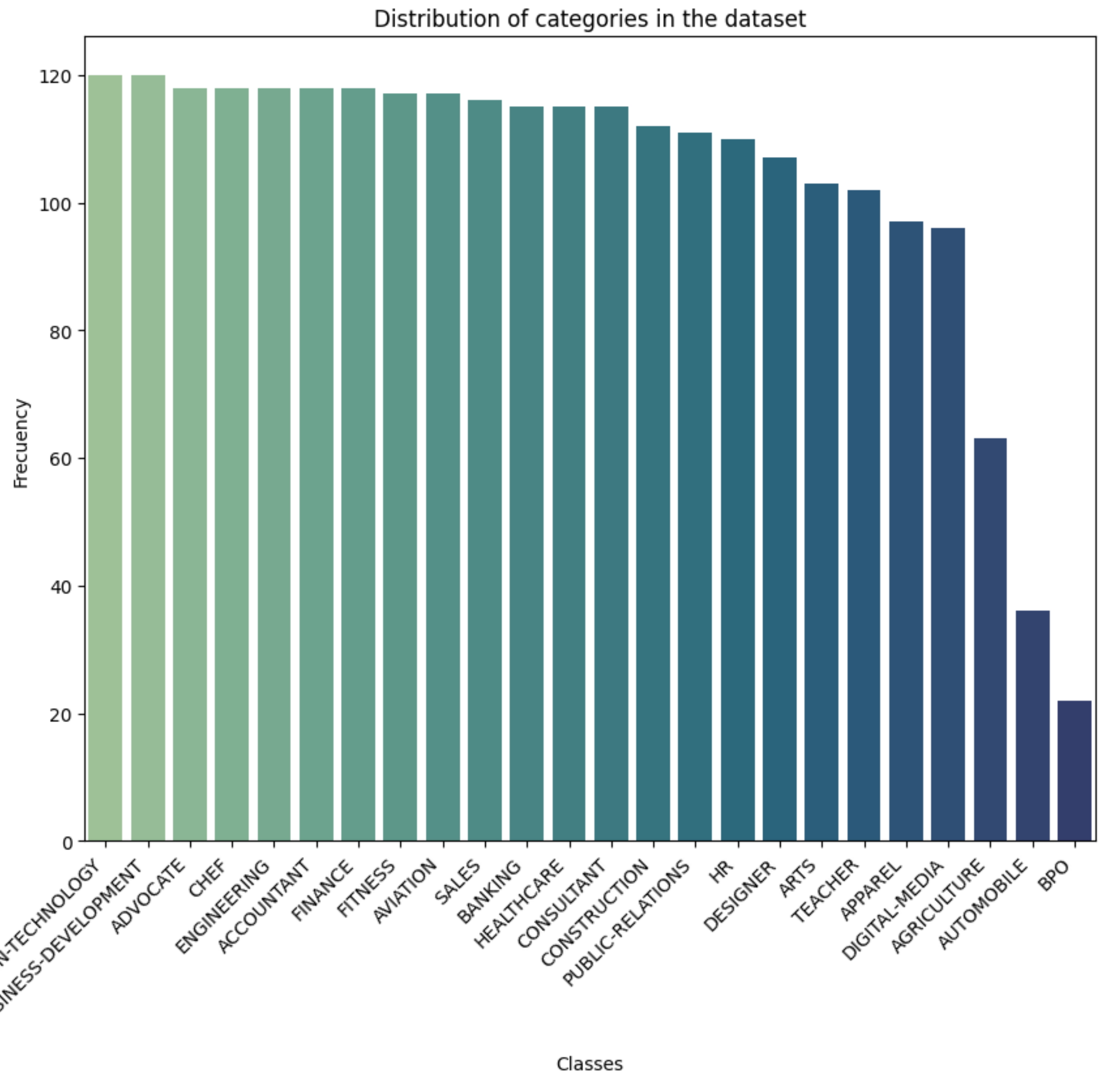

Figure 3: Distribution of resume categories showing class imbalance

3.2 Data Preprocessing Pipeline

The preprocessing pipeline consists of multiple stages to transform raw resume text into suitable input for machine learning models.

3.2.1 Column Selection

Selected relevant features from the dataset:

- `Resume_str`: Plain text content
- `Category`: Target label
- Dropped: `ID`, `Resume_html`

3.2.2 Data Splitting

- **Split ratio:** 80% training, 20% testing
- **Strategy:** Stratified split to maintain category distribution
- **Random state:** 42 (for reproducibility)
- **Results:** 1,987 training samples, 497 test samples

3.2.3 Text Cleaning Process

The text preprocessing follows a systematic approach:

1. **Tokenization**
   - Used NLTK's `word_tokenize()`
   - Splits text into individual words
2. **Punctuation Removal**
   - Remove special characters and symbols
   - Preserve meaningful symbols (+, #, -, /, &)
   - Remove URLs and email addresses
   - Clean multiple whitespaces
3. **Lowercase Conversion**
   - Normalize all text to lowercase
   - Ensures consistency in word matching
4. **Non-ASCII Character Handling**
   - Convert to ASCII encoding
   - Remove accents and special characters
5. **Stopword Removal**
   - Combined English and Spanish stopwords
   - Removes common words without semantic value
   - Examples: "the", "is", "at", "which", "on"
6. **Lemmatization**
   - Used WordNetLemmatizer
   - Reduces words to their base form
   - Filters words shorter than 3 characters
   - Example: "running" → "run", "better" → "good"

3.2.4 Class Balancing

To address class imbalance, a two-step balancing strategy was implemented:

1. **Under-sampling**
   - Applied to majority classes (>100 samples)
   - Cap: 100 samples per class
   - Method: RandomUnderSampler
2. **Over-sampling**
   - Applied to minority classes (< 80 samples)
   - Target: 80 samples minimum
   - Method: RandomOverSampler

Stage	**Samples**	**Notes**
Original Training Data	1,987	Imbalanced
After Under-sampling	~1,800	Capped at 100
After Over-sampling	2,135	Min 80 per class

Table 3: Data Balancing Results

3.3 Feature Engineering

3.3.1 Vectorization Approaches

Two vectorization methods were evaluated:

1. **Count Vectorizer**
   - Counts term frequency in documents
   - N-gram range: (1, 2) - unigrams and bigrams
   - Lowercase: False (already processed)
   - Creates sparse matrix representation
2. **TF-IDF Vectorizer**
   - Term Frequency-Inverse Document Frequency
   - Weights terms by importance across corpus
   - N-gram range: (1, 2)
   - Lowercase: False

3.4 Model Development

3.4.1 Algorithm Selection

Random Forest Classifier was selected due to:

- Excellent performance on text classification
- Handles high-dimensional sparse features
- Robust to overfitting
- Provides feature importance
- Works well with imbalanced data

3.4.2 Hyperparameter Tuning

Grid Search Cross-Validation was used to find optimal parameters:

Table 4: Hyperparameter Search Space

Parameter	Values Tested
max_features	['log2', 0.5]
criterion	['log_loss', 'entropy']
random_state	42 (fixed)

3.4.3 Cross-Validation Strategy

- **Method:** 5-fold cross-validation
- **Scoring metrics:** F1-macro and Accuracy
- **Refit strategy:** Best F1-macro score

3.4.4 Final Model Configuration

Best hyperparameters found:

Listing 2: Optimal Model Configuration

```
RandomForestClassifier(
    random_state=42,
    max_features=0.5,
    criterion='log_loss'
)
```

3.5 Training Pipeline

The complete training pipeline implemented in `train_pipeline.py`:

1. Load balanced training and test data
2. Sanitize data (handle missing values, empty strings)
3. Vectorize text using selected method
4. Train Random Forest model
5. Evaluate on test set
6. Save serialized models (vectorizer + classifier)

4 Implementation Details

4.1 API Development with FastAPI

4.1.1 Endpoints

Method	Endpoint	Description
GET	/	Serves web interface
GET	/health	API and model status check
POST	/predict	Predict category from JSON text
POST	/predict/file	Predict from uploaded file (TXT/PDF)
GET	/categories	List all available categories
GET	/docs	Interactive API documentation (Swagger)
GET	/redoc	Alternative documentation (ReDoc)

Table 5: API Endpoints

4.1.2 Error Handling

Comprehensive error handling for:

- Invalid file types
- File size limits (10MB)
- Empty text input
- Missing models
- Processing errors

4.2 Web Interface

4.2.1 Features

- Two input modes: Text paste and File upload
- Drag-and-drop file support
- Real-time character counter
- File validation (type and size)
- Visual results display with confidence bars
- Top-5 probabilities chart
- Responsive design
- Error messages with retry options

4.2.2 User Interface Components

1. **Tab Navigation**: Switch between text input and file upload
2. **Text Input Area**:
    - Character counter (minimum 100 characters)
    - Large textarea for resume text
    - Analyze button with loading state
3. **File Upload Area**:
    - Drag-and-drop zone
    - File type validation (TXT, PDF)
    - File size validation (10MB max)
    - File info display with remove option
4. **Results Display**:
    - Predicted category badge
    - Confidence percentage with progress bar
    - Top 5 probabilities with horizontal bars
    - Reset button for new analysis

5 Results and Performance Metrics

5.1 Model Performance

5.1.1 Cross-Validation Results

Vectorization Method	F1-Macro (CV)	Accuracy (CV)
Count Vectorizer	0.7230	0.7367
TF-IDF Vectorizer	0.7192	0.7315

Table 6: Cross-Validation Performance (5-fold)

5.1.2 Test Set Performance

Random Forest with Count Vectorizer (Selected Model):

Metric	Score
Accuracy	0.69
Macro Average Precision	0.70
Macro Average Recall	0.65
Macro Average F1-Score	0.65
Weighted Average Precision	0.71
Weighted Average Recall	0.69
Weighted Average F1-Score	0.68

Table 7: Test Set Performance - Overall Metrics

5.1.3 Per-Category Performance

Category	Precision	Recall	F1-Score	Support
ACCOUNTANT	0.69	1.00	0.81	24
ADVOCATE	0.81	0.54	0.65	24
AGRICULTURE	0.40	0.31	0.35	13
APPAREL	0.50	0.32	0.39	19
ARTS	0.56	0.43	0.49	21
AUTOMOBILE	0.33	0.14	0.20	7
AVIATION	0.94	0.67	0.78	24
BANKING	0.60	0.39	0.47	23
BPO	0.00	0.00	0.00	4
BUSINESS-DEV	0.57	0.88	0.69	24
CHEF	1.00	0.83	0.91	24
CONSTRUCTION	0.88	0.95	0.91	22
CONSULTANT	0.54	0.30	0.39	23
DESIGNER	0.80	0.95	0.87	21
DIGITAL-MEDIA	0.67	0.53	0.59	19
ENGINEERING	0.73	0.79	0.76	24
FINANCE	0.68	0.71	0.69	24
FITNESS	0.75	0.65	0.70	23
HEALTHCARE	0.54	0.65	0.59	23
HR	0.73	0.73	0.73	22
INFO-TECHNOLOGY	0.83	0.79	0.81	24
PUBLIC-RELATIONS	0.62	0.68	0.65	22
SALES	0.51	0.78	0.62	23
TEACHER	0.56	0.95	0.70	20

Table 9: Classification Report by Category (Test Set)

5.2 Performance Analysis

5.2.1 Best Performing Categories

Categories with F1-Score > 0.80:

- **CONSTRUCTION** (0.91): High precision and recall
- **CHEF** (0.91): Perfect precision
- **DESIGNER** (0.87): Excellent overall performance
- **ACCOUNTANT** (0.81): Perfect recall
- **INFORMATION-TECHNOLOGY** (0.81): Balanced metrics

5.2.2 Challenging Categories

Categories with F1-Score < 0.50:

- **BPO** (0.00): Very low support (4 samples)

- **AUTOMOBILE** (0.20): Low support and class imbalance
- **AGRICULTURE** (0.35): Limited training data
- **APPAREL** (0.39): Class confusion
- **CONSULTANT** (0.39): Overlapping terminology

5.2.3 Key Observations

1. **Impact of Sample Size**:
   - Categories with more samples show better performance
   - BPO's poor performance directly correlates with minimal samples (4)
2. **Domain-Specific Terminology**:
   - Technical categories (IT, Engineering, Aviation) perform well
   - Clear domain vocabulary aids classification
3. **Cross-Category Confusion**:
   - Business-related categories show some overlap
   - Consultant role overlaps with multiple domains
4. **Model Robustness**:
   - Macro F1-score of 0.65 indicates good generalization
   - Weighted metrics slightly higher due to larger category representation

6 Conclusions

6.1 Achievements

1. Successfully developed an automated resume classification system with 69% accuracy
2. Implemented comprehensive NLP preprocessing pipeline improving text quality
3. Created production-ready REST API with FastAPI for easy integration
4. Built intuitive web interface for non-technical users
5. Achieved F1-macro score of 0.65 across 24 diverse categories

6.2 Challenges Addressed

- **Class Imbalance**: Mitigated using under-sampling and over-sampling techniques
- **Text Variability**: Handled through robust preprocessing pipeline
- **Multiple Categories**: Managed 24 distinct professional categories
- **Real-world Deployment**: Implemented complete API with error handling

6.3 Limitations

1. **Sample Size**: Some categories have limited training data (e.g., BPO with 4 samples)
2. **Category Overlap**: Business-related roles show confusion due to similar terminology
3. **Language Support**: Currently optimized for English resumes only
4. **Format Constraints**: PDF extraction may lose formatting information

6.4 Future Improvements

6.4.1 Model Enhancements

- Experiment with advanced models (XGBoost, Neural Networks)
- Implement ensemble methods combining multiple classifiers
- Use pre-trained language models (BERT, GPT) for better semantic understanding
- Add feature engineering for skills, experience years, education level

6.4.2 Data Improvements

- Collect more samples for underrepresented categories
- Augment data using paraphrasing techniques
- Include multilingual support (Spanish, French, etc.)
- Add domain-specific dictionaries for technical terms

6.4.3 Performance Optimization

- Implement caching for common predictions
- Add async processing for large files
- Optimize vectorizer for faster inference
- Deploy using Docker containers
- Set up load balancing for production

7 References

1. Resume Dataset, Kaggle: https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset/data
2. FastAPI Documentation: https://fastapi.tiangolo.com/
3. Scikit-learn User Guide: https://scikit-learn.org/stable/user_guide.html
4. NLTK Documentation: https://www.nltk.org/
5. Imbalanced-learn Documentation: https://imbalanced-learn.org/
6. Random Forest Classifier: Breiman, L. (2001). "Random Forests". Machine Learning. 45 (1): 5–32