# **Job Classification and Recommendation System**

## **Project Overview**

**Title**: Job Classification and Recommendation System  
**Objective**:

* Classify job postings into categories (IT, Marketing, Healthcare) based on their descriptions.
* Recommend job postings to candidates based on their resumes, providing match scores.

**Current Deployment**:

* **URL**: https://ml-project-yzfq.onrender.com/
* **Git Repository**: https://github.com/NikhilRao1-ai/ML\_Project (as set up previously)
* **Start Command**: web: waitress-serve --port=$PORT jobs:app (set in the Render dashboard)

## **What the Code Does**

### **1. Core Functionality**

The project consists of three main scripts (jobs.py, train\_model.py, train\_matching.py), each contributing to the overall system. Here’s what each script does:

#### **jobs.py**

* **Purpose**: This is an earlier version of the Flask web application that implements job classification and recommendation without a persistent database.
* **Key Features**:
  + **Job Classification**:
    - Uses a Convolutional Neural Network (CNN) to classify job postings into IT, Marketing, or Healthcare.
    - Preprocesses text by converting to lowercase, removing punctuation, removing stopwords, tokenizing, and padding sequences.
    - Model architecture: Embedding → Conv1D → GlobalMaxPooling1D → Dense → Dropout → Dense (softmax for 3 categories).
    - Trains on a small, hardcoded list of 9 job postings.
  + **Job Recommendations**:
    - Uses a separate CNN to predict a match score between a job description and a resume.
    - Combines padded sequences of the job description and resume, then predicts a match score (0 to 1) using a binary classification model.
    - Recommends the top 3 jobs by ranking match scores.
    - Note: The matching model in jobs.py is not trained due to the lack of a matching dataset; it’s a placeholder.
  + **Web Interface**:
    - Built with Flask and styled using Tailwind CSS.
    - Routes:
      * /: Displays the home page with forms for classification and recommendations.
      * /classify: Accepts a job description and returns its category.
      * /recommend: Accepts a resume and returns the top 3 recommended jobs with match scores.
    - Templates (index.html, result.html, recommend.html) provide a user-friendly interface with navigation and error handling.
* **Limitations**:
  + Small dataset (9 job postings).
  + Matching model is not trained; recommendations rely on the model architecture but lack actual training data.
  + No persistent database; job postings are hardcoded.
  + No evaluation metrics are computed or displayed.

#### **train\_model.py**

* **Purpose**: Extends jobs.py to train the classification model on a larger dataset and evaluate its performance.
* **Key Features**:
  + **Dataset**:
    - Expands the dataset to 45 job postings (15 each for IT, Marketing, Healthcare).
  + **Model Updates**:
    - Inherits from JobMatcher in jobs.py but increases the Dropout rate (from 0.5 to 0.7) to reduce overfitting.
    - Trains the classification model for 20 epochs (up from 10 in jobs.py).
  + **Evaluation**:
    - Splits data into training and test sets (80% train, 20% test).
    - Evaluates using Precision, Recall, and F1-Score (weighted average).

Example output (from previous CMD results):  
Precision: 0.57

Recall: 0.44

F1-Score: 0.36

* + **Output**:
    - Saves the trained model and tokenizer to models/job\_matcher\_cnn.keras and models/tokenizer.pkl.
* **Limitations**:
  + No web interface; this script is for training and evaluation only.
  + Metrics indicate moderate performance, likely due to the small dataset and class imbalance.
  + Does not train the matching model.

#### **train\_matching.py**

* **Purpose**: The main script currently deployed on Render, providing an improved version of the web application with a database, training, and evaluation metrics.
* **Key Features**:
  + **Database**:
    - Uses SQLite (data/hr\_database.db) to store job postings.
    - Initializes with 60 sample jobs (20 each for IT, Marketing, Healthcare) if no data exists.
    - Persists job data between sessions (though Render’s free tier resets the database on redeployment).
  + **Job Classification**:
    - Uses a CNN to classify job postings, similar to jobs.py but with a deeper architecture (two Conv1D layers: 256 filters, then 128 filters).
    - Preprocesses text (lowercase, remove punctuation, remove stopwords, tokenize, pad).
    - Trains on the 60 sample jobs, with early stopping (patience=3) to prevent overfitting.
    - Saves the model, tokenizer, and label encoder to models/.
  + **Job Recommendations**:
    - Implements a fallback recommendation system using cosine similarity (since the matching model training is not fully implemented).
    - Recommends the top 3 jobs based on similarity scores between the resume and job descriptions.
  + **Evaluation**:
    - Computes a Classification Report (Precision, Recall, F1-Score) for the classifier.

Example output (from previous logs):  
Classification Report:

precision recall f1-score support

Healthcare 0.00 0.00 0.00 2.0

IT 0.00 0.00 0.00 1.0

Marketing 0.00 0.00 0.00 0.0

* + - Note: Poor performance due to a very small test set (3 samples).
  + **Web Interface**:
    - Flask app with routes:
      * /home: Home page with forms for classification and recommendations.
      * /classify\_form: Classifies a job posting.
      * /recommend: Provides job recommendations.
      * /: A simpler classification interface (from an earlier version).
    - Templates (index.html, result.html, recommend.html) provide a basic UI (less styled than jobs.py).
* **Deployment**:
  + Deployed on Render at https://ml-project-yzfq.onrender.com/.
  + Uses Waitress as the production server (web: waitress-serve --port=$PORT jobs:app).
* **Limitations**:
  + Matching model training is not implemented; recommendations use cosine similarity as a fallback.
  + No /metrics endpoints in this version of the code (though they were added in later updates, as seen in previous responses).
  + Poor classification performance due to small test set.
  + Basic UI compared to jobs.py.

### **2. How the Scripts Work Together**

* **jobs.py**: Provides the initial implementation with a styled UI but lacks a database and proper training for the matching model.
* **train\_model.py**: Enhances the classification model by training on a larger dataset (45 jobs) and evaluating its performance, but does not integrate with the web app.
* **train\_matching.py**: The deployed script, combining classification, a database, and a fallback recommendation system. It’s the most complete implementation but lacks the matching model training and metrics endpoints in this version.

### **3. Current Deployment Status**

* **URL**: https://ml-project-yzfq.onrender.com/
* **Features Available**:
  + Classify job postings (e.g., "Software Engineer needed with Python skills" → "Category: IT").
  + Get job recommendations (e.g., resume input returns top 3 jobs with match scores, using cosine similarity).
* **Missing Features**:
  + The /metrics, /matching\_metrics, and /recommendation\_metrics endpoints are not present in this version of train\_matching.py (though they were added in later updates, as discussed previously).
  + Matching model is not trained; recommendations rely on cosine similarity.
* **Database**:
  + SQLite database resets on Render’s free tier, but train\_matching.py repopulates it with 60 sample jobs on startup.

## **Project Functionality in Detail**

### **Job Classification**

* **Input**: A job description (e.g., "Software Engineer needed with Python skills").
* **Process**:
  + Text is cleaned (lowercase, remove punctuation, remove stopwords).
  + Tokenized and padded using the Tokenizer from Keras.
  + Classified using a CNN model.
* **Output**: A category (IT, Marketing, or Healthcare).
* **Performance**:
  + train\_model.py shows moderate performance (Precision: 0.57, Recall: 0.44, F1-Score: 0.36).
  + train\_matching.py shows poor performance due to a small test set (all metrics 0.00).

### **Job Recommendations**

* **Input**: A resume (e.g., "I am a Software Engineer with 4 years of experience in Python and Flask...").
* **Process**:
  + Text is preprocessed (cleaned, tokenized, padded).
  + Cosine similarity is computed between the resume and each job description in the database.
  + Top 3 jobs are selected based on similarity scores.
* **Output**: A list of 3 jobs with their descriptions, categories, and match scores.
* **Performance**:
  + Not evaluated in this version (metrics like Precision, Recall, F1-Score, and MAP are missing due to lack of matching model training).

### **Web Interface**

* **Home Page** (/home): Provides forms for classification and recommendations.
* **Classification** (/classify\_form): Displays the predicted category.
* **Recommendations** (/recommend): Shows the top 3 recommended jobs.
* **UI**: Basic HTML templates with minimal styling (compared to the Tailwind CSS version in jobs.py).

## **Limitations**

* **Small Dataset**:
  + Classification: 60 jobs in train\_matching.py, 45 in train\_model.py.
  + Matching: No training data for the matching model; uses cosine similarity as a fallback.
* **Poor Classification Performance**:
  + Small test set in train\_matching.py leads to metrics of 0.00.
  + Moderate performance in train\_model.py (F1-Score: 0.36).
* **Missing Features**:
  + Matching model is not trained.
  + Metrics endpoints (/metrics, etc.) are not present in this version of train\_matching.py.
* **Deployment**:
  + Render’s free tier resets the database and may spin down the app after inactivity.
  + Basic UI in the deployed version (train\_matching.py).

## **Future Improvements**

* **Train the Matching Model**:
  + Add a dataset of resume-job pairs to train the matching CNN in train\_matching.py.
  + Evaluate using Precision, Recall, F1-Score, and MAP.
* **Add Metrics Endpoints**:
  + Include routes to display evaluation metrics (as seen in previous updates).
* **Larger Dataset**:
  + Collect more job postings (e.g., 1,000+) to improve classification and recommendation accuracy.
* **Improved UI**:
  + Use the Tailwind CSS styling from jobs.py in train\_matching.py for a better user experience.
* **Persistent Database**:
  + Use a paid Render tier or an external database (e.g., PostgreSQL) for persistence.
* **File Upload**:
  + Allow users to upload resume files (e.g., PDF) instead of pasting text.

## **Deployment Instructions**

* **Current URL**: https://ml-project-new.onrender.com/
* **Start Command**: waitress-serve --port=$PORT train\_matching:app (set in Render dashboard).
* **Note**: If redeploying, ensure this start command is used to run train\_matching.py.

## **How to Use**

1. **Access the App**:
   * Visit https://ml-project-yzfq.onrender.com/.
2. **Classify a Job Posting**:
   * Enter a job description (e.g., "Software Engineer needed with Python skills").
   * Output: Category (e.g., IT).
3. **Get Job Recommendations**:
   * Enter your resume (e.g., "I am a Software Engineer with 4 years of experience in Python and Flask...").
   * Output: Top 3 recommended jobs with match scores.
4. **Tips**:
   * Include relevant skills and keywords in your resume for better matches.
   * The app may spin down on Render’s free tier; wait a few minutes if unavailable.