Job Classification and Recommendation System - Project Document

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/mlproject
- **Start Command**: `waitress-serve --port=$PORT train_matching:app` (Updated on Wed, 11 Jun 2025 04:17 PM IST to use `train_matching.py` instead of `jobs.py`).

Project Status:
This project meets the requirements of the Machine Learning Assignment. It implements job classification using a pre-trained Convolutional Neural Network (CNN), job recommendations using cosine similarity (as a fallback due to an untrained matching model), and a web interface with Tailwind CSS styling. Training is disabled in `train_matching.py` to ensure fast startup on Render's free tier, and a pre-trained classification model from `train_model.py` is used.

Requirements Fulfillment

The project aligns with the Machine Learning Assignment requirements as outlined below.

1. Job Classification

Objective: Classify job postings into categories (e.g., IT, Marketing, Healthcare) based on their descriptions.
Requirement:

- Use machine learning algorithms like Naive Bayes, SVM, or deep learning models (e.g., CNNs or LSTMs) for classification.

Implementation:

- **Approach**: Uses a CNN for text classification, implemented across all scripts (`jobs.py`, `train_model.py`, `train_matching.py`).
- **Model Architecture**:
  - `jobs.py` and `train_model.py`: Embedding → Conv1D (128 filters, kernel size 5) → GlobalMaxPooling1D → Dense (64 units, ReLU) → Dropout (0.7 in `train_model.py`, 0.5 in `jobs.py`) → Dense (softmax for 3 categories).
  - `train_matching.py`: Deeper architecture with Embedding → Conv1D (256 filters) → Conv1D (128 filters) → GlobalMaxPooling1D → Dense → Dropout → Dense (softmax).
- **Data Preprocessing**:
  - Clean text: Lowercase, remove punctuation, remove stopwords (NLTK).
  - Tokenize and pad sequences using Keras `Tokenizer` and `pad_sequences`.
- **Dataset**:
  - `jobs.py`: 9 job postings (hardcoded).
  - `train_model.py`: 45 job postings (15 per category).
  - `train_matching.py`: 60 job postings (20 per category, stored in SQLite).
- **Training**:
  - Pre-trained using `train_model.py` (45 samples, 20 epochs, dropout 0.7).
  - Saved to `models/job_matcher_cnn.keras` and copied to `models/cnn_classifier.keras` for `train_matching.py` compatibility.
- **Deployment**: The deployed app (`train_matching.py`) loads the pre-trained model without training at startup.

2. Job Recommendations

Objective: Recommend jobs to candidates based on their resumes, providing match scores.
Requirement:

- **Problem Definition**:
  - **Input**: User profile (e.g., skills, education, work experience), job listings (e.g., job title, description, required skills).
  - **Output**: Job recommendations or matching score.
- **Possible Approaches**:
  - Data Preprocessing: Clean job descriptions and resumes, extract features.
  - Model: Collaborative filtering, content-based filtering, or a ranking model using ML/NLP techniques.
  - Evaluation: Use metrics like Precision, Recall, F1-Score, or Mean Average Precision (MAP).

Implementation:

- **Problem Definition**:

    - **Input**: Resume text (e.g., "I am a Software Engineer with 4 years of experience in Python and Flask...").
    - **Output**: Top 3 recommended jobs with match scores.
- **Approach**:
    - **Data Preprocessing**:
        - Clean text: Lowercase, remove punctuation, remove stopwords.
        - Tokenize and pad sequences for resumes and job descriptions.
    - **Model**:
        - **Intended Model**: CNN-based binary classification to predict match scores (implemented in `jobs.py` and `train_matching.py`).
        - **Current Implementation**: Due to the lack of a trained matching model, `train_matching.py` falls back to content-based filtering using cosine similarity.
        - **Recommendation Logic**:
            - Compute cosine similarity between the resume and each job description.
            - Sort by similarity score and return the top 3 jobs.
    - **Evaluation**:
        - Metrics (Precision, Recall, F1-Score, MAP) are not computed for recommendations due to the untrained matching model.
- **Dataset**:
    - `jobs.py` and `train_matching.py` use a list of 9 job postings for recommendations.
    - `train_matching.py` can also use the 60 jobs in the SQLite database.
- **Deployment**: The deployed app (`train_matching.py`) uses cosine similarity for recommendations since the matching model isn't pre-trained.

3. Evaluation

Requirement:

- Use metrics like Precision, Recall, F1-Score, or Mean Average Precision (MAP) to evaluate performance.

Implementation:

- **Classification Evaluation**:
    - **Script**: `train_model.py` evaluates the classification model on a test set (20% of 45 samples = 9 samples).
    - **Metrics**:
        - Precision: 0.57
        - Recall: 0.44
        - F1-Score: 0.36
    - **Previous Evaluation in `train_matching.py`**:
        - Metrics: Precision: 0.00, Recall: 0.00, F1-Score: 0.00 (small test set of 3 samples).
- **Recommendation Evaluation**:

    - Not performed due to the untrained matching model.
- **Deployment**:
    - The `/metrics` endpoint in `train_matching.py` displays placeholder metrics (all 0.0) since training is disabled.

Project Functionality in Detail

1. Job Classification

- **Input**: Job description (e.g., "Software Engineer needed with Python skills").
- **Process**:
    - Clean, tokenize, and pad the text.
    - Classify using a pre-trained CNN model (loaded from `models/cnn_classifier.keras`).
- **Output**: Category (IT, Marketing, or Healthcare).
- **Performance**: F1-Score: 0.36 (from `train_model.py`).
- **Deployment**: Uses the pre-trained model for fast startup.

2. Job Recommendations

- **Input**: Resume (e.g., "I am a Software Engineer with 4 years of experience in Python and Flask...").
- **Process**:
    - Clean, tokenize, and pad the resume and job descriptions.
    - Compute cosine similarity (fallback in `train_matching.py`).
    - Return the top 3 jobs by similarity score.
- **Output**: Top 3 jobs with descriptions, categories, and match scores.
- **Performance**: Not evaluated.
- **Deployment**: Uses cosine similarity due to the untrained matching model.

3. Web Interface

- **Routes** (`train_matching.py`):
    - `/home`: Home page with forms for classification and recommendations.
    - `/classify_form`: Classifies a job posting.
    - `/recommend`: Provides top 3 recommended jobs.
    - `/`: Simple classification interface.
    - `/metrics`: Displays placeholder metrics (all 0.0).
- **UI**: Tailwind CSS-styled templates (updated at 04:10 PM IST).
- **Error Handling**: Displays error messages for invalid inputs.

What the Code Does

The project consists of three scripts:

1. `train_matching.py` (Deployed Script)

- **Purpose**: The main Flask application with database integration, classification, recommendations, and metrics display.
- **Key Features**:
  - **Database**: SQLite (`data/hr_database.db`) with 60 sample jobs.
  - **Classification**: Uses a pre-trained CNN.
  - **Recommendations**: Falls back to cosine similarity.
  - **Evaluation**: Displays placeholder metrics via `/metrics`.
  - **UI**: Tailwind CSS-styled interface.
- **Limitations**:
  - Matching model not trained.
  - Database resets on Render's free tier.

Full Code:

```
import pandas as pd
import numpy as np
import re
import string
import sqlite3
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,
Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import LabelEncoder
import joblib
import nltk
from nltk.corpus import stopwords
from flask import Flask, request, render_template, render_template_string
import logging
import tensorflow as tf

# Set up logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

```
# Download NLTK stopwords
nltk.download('stopwords', quiet=True)
stop_words = set(stopwords.words('english'))

# Initialize Flask app
app = Flask(__name__)

class JobMatcher:
    def __init__(self, db_path='data/hr_database.db', max_words=5000, max_length=200):
        self.tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
        self.max_length = max_length
        self.label_encoder = LabelEncoder()
        self.cnn_model = None
        self.matching_model = None
        self.db_path = db_path
        self.num_classes = None
        self.model_path = 'models/cnn_classifier.keras'
        self.matching_model_path = 'models/matching_model.keras'
        self.tokenizer_path = 'models/tokenizer.pkl'
        self.encoder_path = 'models/label_encoder.pkl'
        self.job_listings = [
            ("Software Engineer position requiring Python and machine learning skills.", "IT"),
            ("Data Scientist role needing TensorFlow expertise.", "IT"),
            ("Marketing Manager needed with experience in SEO.", "Marketing"),
            ("Content Writer for marketing team, skilled in social media.", "Marketing"),
            ("Nurse Practitioner required with patient care experience.", "Healthcare"),
            ("Medical Assistant needed for clinic with EHR knowledge.", "Healthcare"),
            ("Web Developer role requiring JavaScript and React.", "IT"),
            ("SEO Specialist needed with experience in keyword research.", "Marketing"),
            ("Doctor needed for hospital with 5 years of experience.", "Healthcare")
        ]
        self.resume_job_pairs = [
            ("Python developer with 3 years experience in machine learning.",
             "Software Engineer position requiring Python and machine learning skills.", 1),
            ("Experienced in SEO and digital marketing strategies.",
             "Marketing Manager needed with experience in SEO.", 1),
            ("Nurse with 5 years of patient care experience.",
             "Nurse Practitioner required with patient care experience.", 1),
            ("JavaScript developer skilled in React and Node.js.",
             "Web Developer role requiring JavaScript and React.", 1),
            ("Content writer with social media expertise.",
             "Content Writer for marketing team, skilled in social media.", 1),
            ("Python developer with 3 years experience in machine learning.",
             "Marketing Manager needed with experience in SEO.", 0),
            ("Experienced in SEO and digital marketing strategies.",
             "Nurse Practitioner required with patient care experience.", 0),
            ("Nurse with 5 years of patient care experience.",
             "Software Engineer position requiring Python and machine learning skills.", 0),
```

```
        ("JavaScript developer skilled in React and Node.js.",
         "Content Writer for marketing team, skilled in social media.", 0),
        ("Content writer with social media expertise.",
         "Doctor needed for hospital with 5 years of experience.", 0),
    ]
    self.classification_metrics = None
    self.recommendation_metrics = None

def clean_text(self, text):
    if not isinstance(text, str) or not text.strip():
        raise ValueError("Input text must be a non-empty string")
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = ' '.join(word for word in text.split() if word not in stop_words)
    logger.debug(f"Cleaned text: {text}")
    return text

def initialize_db(self):
    os.makedirs('data', exist_ok=True)
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS jobs (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            job_title TEXT,
            job_description TEXT,
            category TEXT,
            clean_desc TEXT
        )
    ''')
    conn.commit()
    conn.close()

def load_data(self):
    try:
        conn = sqlite3.connect(self.db_path)
        job_df = pd.read_sql_query("SELECT * FROM jobs", conn)
        conn.close()
        return job_df
    except Exception as e:
        logger.error(f"Error loading database: {e}")
        return pd.DataFrame()

def save_data(self, job_df):
    try:
        conn = sqlite3.connect(self.db_path)
        job_df.to_sql('jobs', conn, if_exists='replace', index=False)
        conn.close()
```

```
        except Exception as e:
            logger.error(f"Error saving to database: {e}")

    def build_cnn_model(self, num_classes, vocab_size):
        model = Sequential([
            Embedding(vocab_size, 128, input_length=self.max_length),
            Conv1D(256, 5, activation='relu'),
            Conv1D(128, 3, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(128, activation='relu'),
            Dropout(0.5),
            Dense(num_classes, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        return model

    def build_matching_model(self, vocab_size):
        model = Sequential([
            Embedding(vocab_size, 128, input_length=self.max_length * 2),
            Conv1D(128, 5, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
        return model

    def train_classifier(self, job_df):
        if len(job_df) < 10:
            logger.warning("Limited data may lead to poor model performance. Add more jobs.")
            return
        job_df['clean_desc'] = job_df['job_description'].apply(self.clean_text)
        self.tokenizer.fit_on_texts(job_df['clean_desc'])
        job_sequences = self.tokenizer.texts_to_sequences(job_df['clean_desc'])
        job_padded = pad_sequences(job_sequences, maxlen=self.max_length,
padding='post', truncating='post')

        y = self.label_encoder.fit_transform(job_df['category'])
        self.num_classes = len(self.label_encoder.classes_)
        y = to_categorical(y, num_classes=self.num_classes)

        X_train, X_test, y_train, y_test = train_test_split(job_padded, y, test_size=0.2,
random_state=42)
        logger.info(f"Training samples: {len(X_train)}, Test samples: {len(X_test)}")

        self.cnn_model = self.build_cnn_model(self.num_classes,
len(self.tokenizer.word_index) + 1)
```

```
        self.cnn_model.fit(
            X_train, y_train,
            epochs=15,
            batch_size=16,
            validation_split=0.2,
            verbose=1,
            callbacks=[EarlyStopping(patience=3, restore_best_weights=True)]
        )

        y_pred = self.cnn_model.predict(X_test, verbose=0)
        y_pred_classes = np.argmax(y_pred, axis=1)
        y_test_classes = np.argmax(y_test, axis=1)
        report = classification_report(y_test_classes, y_pred_classes,
target_names=self.label_encoder.classes_, zero_division=0, output_dict=True)
        logger.info("\n📊 Classification Report:")
        logger.info(classification_report(y_test_classes, y_pred_classes,
target_names=self.label_encoder.classes_, zero_division=0))

        self.classification_metrics = {
            'precision': round(report['weighted avg']['precision'], 2),
            'recall': round(report['weighted avg']['recall'], 2),
            'f1_score': round(report['weighted avg']['f1-score'], 2)
        }
        self.save_model()

    def train_matching(self):
        resumes, jobs, labels = zip(*self.resume_job_pairs)
        all_texts = list(resumes) + list(jobs)
        self.tokenizer.fit_on_texts(all_texts)
        resume_sequences = self.tokenizer.texts_to_sequences(resumes)
        job_sequences = self.tokenizer.texts_to_sequences(jobs)
        resume_padded = pad_sequences(resume_sequences, maxlen=self.max_length,
padding='post', truncating='post')
        job_padded = pad_sequences(job_sequences, maxlen=self.max_length,
padding='post', truncating='post')
        X = np.hstack((job_padded, resume_padded))
        y = np.array(labels)

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        logger.info(f"Matching Model - Training samples: {len(X_train)}, Test samples:
{len(X_test)}")

        self.matching_model = self.build_matching_model(len(self.tokenizer.word_index) + 1)
        self.matching_model.fit(
            X_train, y_train,
            epochs=10,
            batch_size=4,
            validation_split=0.2,
```

```
            verbose=1,
            callbacks=[EarlyStopping(patience=3, restore_best_weights=True)]
        )

        y_pred = (self.matching_model.predict(X_test, verbose=0) > 0.5).astype(int)
        precision = precision_score(y_test, y_pred, zero_division=0)
        recall = recall_score(y_test, y_pred, zero_division=0)
        f1 = f1_score(y_test, y_pred, zero_division=0)
        logger.info(f"Recommendation Metrics: Precision={precision:.2f}, Recall={recall:.2f},
F1-Score={f1:.2f}")

        self.recommendation_metrics = {
            'precision': round(precision, 2),
            'recall': round(recall, 2),
            'f1_score': round(f1, 2)
        }
        self.matching_model.save(self.matching_model_path)

    def save_model(self):
        os.makedirs('models', exist_ok=True)
        self.cnn_model.save(self.model_path)
        joblib.dump(self.tokenizer, self.tokenizer_path)
        joblib.dump(self.label_encoder, self.encoder_path)

    def preprocess_text(self, text):
        text = self.clean_text(text)
        sequences = self.tokenizer.texts_to_sequences([text])
        logger.debug(f"Tokenized sequences: {sequences}")
        if not sequences or not sequences[0]:
            logger.debug("Tokenization failed, returning default padded sequence")
            return np.zeros((1, self.max_length), dtype='int32')
        padded = pad_sequences(sequences, maxlen=self.max_length, padding='post',
truncating='post')
        logger.debug(f"Padded sequences: {padded}")
        return padded

    def classify_job(self, description):
        if not hasattr(self, 'cnn_model') or self.cnn_model is None:
            try:
                self.cnn_model = tf.keras.models.load_model(self.model_path)
                with open(self.tokenizer_path, 'rb') as f:
                    self.tokenizer = joblib.load(f)
                with open(self.encoder_path, 'rb') as f:
                    self.label_encoder = joblib.load(f)
            except Exception as e:
                logger.error(f"Error loading model or tokenizer: {e}")
                raise
        job_padded = self.preprocess_text(description)
```

```
        prediction = self.cnn_model.predict(job_padded, verbose=0)
        category_id = np.argmax(prediction, axis=1)[0]
        return self.label_encoder.inverse_transform([category_id])[0]

    def predict_match(self, job_description, resume):
        if not hasattr(self, 'matching_model') or self.matching_model is None:
            try:
                self.matching_model = tf.keras.models.load_model(self.matching_model_path)
                with open(self.tokenizer_path, 'rb') as f:
                    self.tokenizer = joblib.load(f)
            except Exception as e:
                logger.error(f"Error loading matching model: {e}")
                job_padded = self.preprocess_text(job_description)
                resume_padded = self.preprocess_text(resume)
                from sklearn.metrics.pairwise import cosine_similarity
                return cosine_similarity(job_padded, resume_padded)[0][0]
        job_padded = self.preprocess_text(job_description)
        resume_padded = self.preprocess_text(resume)
        combined_input = np.hstack((job_padded, resume_padded))
        score = self.matching_model.predict(combined_input, verbose=0)[0][0]
        return score

    def recommend(self, resume):
        scores = []
        for job_desc, category in self.job_listings:
            score = self.predict_match(job_desc, resume)
            scores.append((job_desc, category, score))
        scores.sort(key=lambda x: x[2], reverse=True)
        return scores[:3]

@app.route('/', methods=['GET', 'POST'])
def classify():
    if request.method == 'POST':
        desc = request.form.get('description', '').strip()
        if not desc:
            return render_template_string('''
                <h1>Job Classification</h1>
                <p style="color: red;">Please enter a job description.</p>
                <form method="post">
                    <textarea name="description" placeholder="Enter job
description"></textarea><br>
                    <input type="submit" value="Classify">
                </form>
                <p><a href="/home">Full Interface</a></p>
            ''')
        try:
            category = matcher.classify_job(desc)
            return render_template_string('''
```

```
            <h1>Job Classification</h1>
            <p>Description: {{ desc }}</p>
            <p>Category: {{ category }}</p>
            <a href="/">Back</a>
            <p><a href="/home">Full Interface</a></p>
        ''', desc=desc, category=category)
    except Exception as e:
        logger.error(f"Classification error: {e}")
        return render_template_string('''
            <h1>Job Classification</h1>
            <p style="color: red;">Error: {{ error }}</p>
            <form method="post">
                <textarea name="description" placeholder="Enter job description">{{ desc
}}</textarea><br>
                <input type="submit" value="Classify">
            </form>
            <p><a href="/home">Full Interface</a></p>
        ''', error=str(e), desc=desc)
    return render_template_string('''
        <h1>Job Classification</h1>
        <form method="post">
            <textarea name="description" placeholder="Enter job description"></textarea><br>
            <input type="submit" value="Classify">
        </form>
        <p><a href="/home">Full Interface</a></p>
    ''')

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

@app.route('/classify_form', methods=['POST'])
def classify_form():
    try:
        job_description = request.form.get('job_description', '').strip()
        if not job_description:
            return render_template('index.html', error="Please fill in the Job Description field.")
        category = matcher.classify_job(job_description)
        return render_template('result.html', category=category)
    except ValueError as e:
        return render_template('index.html', error=str(e))
    except Exception as e:
        logger.error(f"Classification error: {e}")
        return render_template('index.html', error=f"An error occurred: {str(e)}")

@app.route('/recommend', methods=['POST'])
def recommend():
    try:
```

```
        resume = request.form.get('resume', '').strip()
        if not resume:
            return render_template('index.html', error="Please fill in the Resume field.")
        recommendations = matcher.recommend(resume)
        return render_template('recommend.html', recommendations=recommendations)
    except ValueError as e:
        return render_template('index.html', error=str(e))
    except Exception as e:
        logger.error(f"Recommendation error: {e}")
        return render_template('index.html', error=f"An error occurred: {str(e)}")

@app.route('/metrics')
def metrics():
    if matcher.classification_metrics is None or matcher.recommendation_metrics is None:
        return render_template('metrics.html', error="Metrics not available. Please ensure
models are trained.")
    return render_template('metrics.html',
        classification_metrics=matcher.classification_metrics,
        recommendation_metrics=matcher.recommendation_metrics
    )

os.makedirs('templates', exist_ok=True)
index_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Job Classifier and Recommender</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
            <div class="space-x-4">
                <a href="/home" class="text-blue-600 hover:text-blue-800
font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
                <a href="/metrics" class="text-blue-600 hover:text-blue-800
font-medium">Metrics</a>
            </div>
        </div>
    </nav>
```

```
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Job Classifier and Recommender</h1>
        {% if error %}
            <p class="text-red-600 mb-4 text-center">{{ error }}</p>
        {% endif %}
        <div class="mb-8">
            <h2 class="text-2xl font-semibold text-gray-800 mb-4">Classify a Job Posting</h2>
            <form action="/classify_form" method="post" class="space-y-4">
                <textarea id="job_description" name="job_description" rows="5" placeholder="Enter job description..." class="w-full p-4 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500">{{ request.form.get('job_description', '') }}</textarea>
                <button type="submit" class="w-full bg-blue-600 text-white px-6 py-3 rounded-lg hover:bg-blue-700 transition duration-300">Classify</button>
            </form>
        </div>
        <div>
            <h2 class="text-2xl font-semibold text-gray-800 mb-4">Get Job Recommendations</h2>
            <form action="/recommend" method="post" class="space-y-4">
                <textarea id="resume" name="resume" rows="5" placeholder="Enter your resume..." class="w-full p-4 border rounded-lg focus:outline-none focus:ring-2 focus:ring-purple-500">{{ request.form.get('resume', '') }}</textarea>
                <button type="submit" class="w-full bg-purple-600 text-white px-6 py-3 rounded-lg hover:bg-purple-700 transition duration-300">Recommend</button>
            </form>
        </div>
        <p class="mt-4 text-center"><a href="/" class="text-blue-600 hover:text-blue-800">Simple Classification Interface</a></p>
    </div>
</body>
</html>
"""

result_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Classification Result</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
```

```
            <div class="space-x-4">
                <a href="/home" class="text-blue-600 hover:text-blue-800
font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
                <a href="/metrics" class="text-blue-600 hover:text-blue-800
font-medium">Metrics</a>
            </div>
        </div>
    </nav>
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Classification
Result</h1>
        <div class="p-4 bg-blue-50 rounded-lg">
            <p class="text-gray-600">Category: <span class="font-medium text-blue-600">{{
category }}</span></p>
        </div>
        <a href="/home" class="block mt-4 text-blue-600 hover:text-blue-800 text-center">Back
to Home</a>
    </div>
</body>
</html>
"""

recommend_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Job Recommendations</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
            <div class="space-x-4">
                <a href="/home" class="text-blue-600 hover:text-blue-800
font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
```

```
            <a href="/metrics" class="text-blue-600 hover:text-blue-800
font-medium">Metrics</a>
        </div>
    </div>
  </nav>
  <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
    <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Job
Recommendations</h1>
    <div class="space-y-4">
      {% for job_desc, category, score in recommendations %}
        <div class="p-4 bg-purple-50 rounded-lg">
          <p class="text-gray-600">{{ job_desc }} (Category: {{ category }}) - Match Score:
{{ "%.2f" % score }}</p>
        </div>
      {% endfor %}
    </div>
    <a href="/home" class="block mt-4 text-blue-600 hover:text-blue-800 text-center">Back
to Home</a>
  </div>
</body>
</html>
"""

metrics_html = """
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Model Metrics</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
  <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
    <div class="flex justify-between items-center">
      <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
      <div class="space-x-4">
        <a href="/home" class="text-blue-600 hover:text-blue-800
font-medium">Home</a>
        <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
        <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
        <a href="/metrics" class="text-blue-600 hover:text-blue-800
font-medium">Metrics</a>
      </div>
    </div>
```

```
    </nav>
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Model Metrics</h1>
        {% if error %}
            <p class="text-red-600 mb-4 text-center">{{ error }}</p>
        {% else %}
            <div class="mb-8">
                <h2 class="text-2xl font-semibold text-gray-800 mb-4">Classification Metrics</h2>
                <div class="p-4 bg-blue-50 rounded-lg">
                    <p class="text-gray-600">Precision: <span class="font-medium
text-blue-600">{{ classification_metrics.precision }}</span></p>
                    <p class="text-gray-600">Recall: <span class="font-medium text-blue-600">{{
classification_metrics.recall }}</span></p>
                    <p class="text-gray-600">F1-Score: <span class="font-medium
text-blue-600">{{ classification_metrics.f1_score }}</span></p>
                </div>
            </div>
            <div>
                <h2 class="text-2xl font-semibold text-gray-800 mb-4">Recommendation
Metrics</h2>
                <div class="p-4 bg-purple-50 rounded-lg">
                    <p class="text-gray-600">Precision: <span class="font-medium
text-purple-600">{{ recommendation_metrics.precision }}</span></p>
                    <p class="text-gray-600">Recall: <span class="font-medium text-purple-600">{{
recommendation_metrics.recall }}</span></p>
                    <p class="text-gray-600">F1-Score: <span class="font-medium
text-purple-600">{{ recommendation_metrics.f1_score }}</span></p>
                </div>
            </div>
        {% endif %}
        <a href="/home" class="block mt-4 text-blue-600 hover:text-blue-800 text-center">Back
to Home</a>
    </div>
</body>
</html>
"""

with open('templates/index.html', 'w') as f:
    f.write(index_html)
with open('templates/result.html', 'w') as f:
    f.write(result_html)
with open('templates/recommend.html', 'w') as f:
    f.write(recommend_html)
with open('templates/metrics.html', 'w') as f:
    f.write(metrics_html)

matcher = JobMatcher()
```

```
if __name__ == "__main__":
    matcher.initialize_db()
    job_df = matcher.load_data()
    if job_df.empty:
        if os.path.exists('jobs.csv'):
            logger.info("Loading data from jobs.csv")
            job_df = pd.read_csv('jobs.csv')
            matcher.save_data(job_df)
        else:
            logger.info("No data found. Initializing with sample data (60 jobs).")
            sample_jobs = pd.DataFrame({
                'job_title': [
                    'Software Engineer', 'Data Scientist', 'Web Developer', 'DevOps Engineer', 'AI
Researcher',
                    'Cybersecurity Analyst', 'Database Administrator', 'Cloud Architect', 'Mobile App
Developer', 'Systems Analyst',
                    'Machine Learning Engineer', 'Full Stack Developer', 'Network Engineer', 'QA
Engineer', 'Data Analyst',
                    'IT Project Manager', 'Blockchain Developer', 'Game Developer', 'Embedded
Systems Engineer', 'IT Consultant',
                    'Marketing Specialist', 'Content Creator', 'SEO Analyst', 'Brand Manager', 'Digital
Marketer',
                    'Social Media Manager', 'Public Relations Specialist', 'Market Research Analyst',
'Advertising Manager', 'Copywriter',
                    'Graphic Designer', 'Email Marketing Specialist', 'Content Strategist', 'Event
Planner', 'Influencer Marketing Manager',
                    'Product Marketing Manager', 'Marketing Coordinator', 'Media Buyer', 'Creative
Director', 'UX Researcher',
                    'Registered Nurse', 'Physical Therapist', 'Medical Assistant', 'Pharmacist',
'Surgeon',
                    'Emergency Room Nurse', 'Pediatrician', 'Radiologist', 'Anesthesiologist',
'Clinical Laboratory Technician',
                    'Occupational Therapist', 'Speech-Language Pathologist', 'Dental Hygienist',
'Paramedic', 'Cardiologist',
                    'Psychiatrist', 'Nurse Practitioner', 'Health Informatics Specialist', 'Medical Social
Worker', 'Orthopedic Surgeon'
                ],
                'job_description': [
                    'Develop software applications using Python and Java.',
                    'Build machine learning models with TensorFlow and PyTorch.',
                    'Create responsive websites using JavaScript and React.',
                    'Manage cloud infrastructure with AWS and Docker.',
                    'Research advanced AI algorithms and neural networks.',
                    'Protect systems from cyber threats and conduct penetration testing.',
                    'Manage and optimize SQL and NoSQL databases.',
                    'Design scalable cloud solutions on Azure and GCP.',
                    'Build iOS and Android apps using Swift and Kotlin.',
                    'Analyze business systems and recommend IT solutions.',
```

'Design and deploy ML models for predictive analytics.',
'Develop front-end and back-end web applications.',
'Configure and maintain network infrastructure.',
'Test software to ensure quality and reliability.',
'Analyze data using Python and SQL to generate insights.',
'Lead IT projects and coordinate teams.',
'Develop decentralized applications using Ethereum.',
'Create video games using Unity and C#.',
'Program embedded systems for IoT devices.',
'Provide IT consulting services to optimize business processes.',
'Manage social media campaigns and branding strategies.',
'Produce engaging content for blogs and social media.',
'Optimize websites for search engine rankings.',
'Develop brand strategies for product launches.',
'Create and manage digital ad campaigns.',
'Oversee social media platforms and engagement.',
'Handle media relations and corporate communications.',
'Conduct surveys and analyze consumer trends.',
'Plan and execute advertising campaigns.',
'Write compelling copy for marketing materials.',
'Design visual content using Adobe Creative Suite.',
'Develop email marketing campaigns to boost engagement.',
'Plan content strategies for brand consistency.',
'Organize corporate events and conferences.',
'Collaborate with influencers to promote products.',
'Market products to target audiences.',
'Support marketing campaigns and logistics.',
'Purchase advertising space for campaigns.',
'Lead creative projects and teams.',
'Conduct user research to improve product design.',
'Provide patient care in hospital settings.',
'Assist patients with physical rehabilitation programs.',
'Support physicians in clinical and administrative tasks.',
'Dispense medications and counsel patients.',
'Perform surgical procedures in operating rooms.',
'Provide critical care in emergency departments.',
'Diagnose and treat children’s illnesses.',
'Interpret medical imaging for diagnoses.',
'Administer anesthesia during surgeries.',
'Analyze biological samples in labs.',
'Help patients improve daily living skills.',
'Treat communication and swallowing disorders.',
'Clean teeth and educate patients on oral health.',
'Provide emergency medical care in ambulances.',
'Diagnose and treat heart conditions.',
'Treat mental health disorders with therapy and medication.',
'Provide primary care as an advanced practice nurse.',
'Manage healthcare data and IT systems.',

```
                'Support patients and families with social services.',
                'Perform surgeries on bones and joints.'
            ],
            'category': [
                'IT', 'IT', 'IT', 'IT', 'IT',
                'IT', 'IT', 'IT', 'IT', 'IT',
                'IT', 'IT', 'IT', 'IT', 'IT',
                'IT', 'IT', 'IT', 'IT', 'IT',
                'Marketing', 'Marketing', 'Marketing', 'Marketing', 'Marketing',
                'Marketing', 'Marketing', 'Marketing', 'Marketing', 'Marketing',
                'Marketing', 'Marketing', 'Marketing', 'Marketing', 'Marketing',
                'Marketing', 'Marketing', 'Marketing', 'Marketing', 'Marketing',
                'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare',
                'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare',
                'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare',
                'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare', 'Healthcare'
            ]
        })
        matcher.save_data(sample_jobs)
    job_df = matcher.load_data()
# Training disabled to improve startup speed on Render
# if not job_df.empty:
#     matcher.train_classifier(job_df)
# matcher.train_matching()
matcher.classification_metrics = {'precision': 0.0, 'recall': 0.0, 'f1_score': 0.0}
matcher.recommendation_metrics = {'precision': 0.0, 'recall': 0.0, 'f1_score': 0.0}
from waitress import serve
logger.info("Starting Waitress server on 0.0.0.0:5000")
serve(app, host='0.0.0.0', port=5000, threads=4)
```

2. `jobs.py`

- **Purpose**: An earlier Flask application without database integration, now replaced by `train_matching.py`.
- **Key Features**:
  - Classification and recommendations using CNNs.
  - Tailwind CSS-styled UI.
- **Limitations**:
  - No database; uses 9 hardcoded jobs.
  - No `/metrics` endpoint.
  - Matching model not pre-trained.

Full Code:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,
Dropout
from flask import Flask, request, render_template
import pickle
import logging
import re
from nltk.corpus import stopwords
import nltk

# Download NLTK stopwords data
nltk.download('stopwords', quiet=True)

# Set up logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

# Initialize Flask app
app = Flask(__name__)

# Stop words for text cleaning
stop_words = set(stopwords.words('english'))

class JobMatcher:
    def __init__(self, vocab_size=5000, max_length=500, embedding_dim=100):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.embedding_dim = embedding_dim
        self.tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
        self.classification_model = self.build_classification_model()
        self.matching_model = self.build_matching_model()
        self.classification_model_path = 'models/job_matcher_cnn.keras'
        self.matching_model_path = 'models/job_matcher_matching.keras'
        self.tokenizer_path = 'models/tokenizer.pkl'
        self.categories = {0: "IT", 1: "Marketing", 2: "Healthcare"}
        self.job_listings = [
            ("Software Engineer position requiring Python and machine learning skills.", "IT"),
            ("Data Scientist role needing TensorFlow expertise.", "IT"),
            ("Marketing Manager needed with experience in SEO.", "Marketing"),
            ("Content Writer for marketing team, skilled in social media.", "Marketing"),
            ("Nurse Practitioner required with patient care experience.", "Healthcare"),
            ("Medical Assistant needed for clinic with EHR knowledge.", "Healthcare"),
            ("Web Developer role requiring JavaScript and React.", "IT"),
            ("SEO Specialist needed with experience in keyword research.", "Marketing"),
            ("Doctor needed for hospital with 5 years of experience.", "Healthcare")
        ]
```

```
    def build_classification_model(self):
        model = Sequential([
            Embedding(self.vocab_size, self.embedding_dim),
            Conv1D(128, 5, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(3, activation='softmax')
        ])
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
        return model

    def build_matching_model(self):
        model = Sequential([
            Embedding(self.vocab_size, self.embedding_dim),
            Conv1D(128, 5, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
        return model

    def preprocess_text(self, text):
        if not isinstance(text, str) or not text.strip():
            raise ValueError("Input text must be a non-empty string")
        text = text.lower()
        text = re.sub(r'[^\w\s]', '', text)
        text = ' '.join(word for word in text.split() if word not in stop_words)
        logger.debug(f"Cleaned text: {text}")
        sequences = self.tokenizer.texts_to_sequences([text])
        logger.debug(f"Tokenized sequences: {sequences}")
        if not sequences or not sequences[0] or any(x is None for x in sequences[0]):
            logger.debug("Tokenization failed or contains None values, returning default padded
sequence")
            return np.zeros((1, self.max_length), dtype='int32')
        padded = pad_sequences(sequences, maxlen=self.max_length, padding='post',
truncating='post')
        logger.debug(f"Padded sequences: {padded}")
        return padded

    def train_classification(self, job_descriptions, categories):
        all_texts = job_descriptions
        self.tokenizer.fit_on_texts(all_texts)
        sequences = self.tokenizer.texts_to_sequences(job_descriptions)
```

```
        padded = pad_sequences(sequences, maxlen=self.max_length, padding='post',
truncating='post')
        y = np.array([{"IT": 0, "Marketing": 1, "Healthcare": 2}[cat] for cat in categories])
        self.classification_model.fit(padded, y, epochs=10, validation_split=0.2, batch_size=32)
        self.classification_model.save(self.classification_model_path)
        with open(self.tokenizer_path, 'wb') as f:
            pickle.dump(self.tokenizer, f)

    def train_matching(self, job_descriptions, resumes, labels):
        all_texts = job_descriptions + resumes
        self.tokenizer.fit_on_texts(all_texts)
        job_sequences = self.tokenizer.texts_to_sequences(job_descriptions)
        resume_sequences = self.tokenizer.texts_to_sequences(resumes)
        job_padded = pad_sequences(job_sequences, maxlen=self.max_length,
padding='post', truncating='post')
        resume_padded = pad_sequences(resume_sequences, maxlen=self.max_length,
padding='post', truncating='post')
        X = np.hstack((job_padded, resume_padded))
        y = np.array(labels)
        self.matching_model.fit(X, y, epochs=10, validation_split=0.2, batch_size=32)
        self.matching_model.save(self.matching_model_path)
        with open(self.tokenizer_path, 'wb') as f:
            pickle.dump(self.tokenizer, f)

    def predict_category(self, job_description):
        if not hasattr(self, 'classification_model') or not hasattr(self, 'tokenizer'):
            self.classification_model =
tf.keras.models.load_model(self.classification_model_path)
            with open(self.tokenizer_path, 'rb') as f:
                self.tokenizer = pickle.load(f)
        job_padded = self.preprocess_text(job_description)
        prediction = self.classification_model.predict(job_padded)
        category_id = np.argmax(prediction, axis=1)[0]
        return self.categories[category_id]

    def predict_match(self, job_description, resume):
        if not hasattr(self, 'matching_model') or not hasattr(self, 'tokenizer'):
            self.matching_model = tf.keras.models.load_model(self.matching_model_path)
            with open(self.tokenizer_path, 'rb') as f:
                self.tokenizer = pickle.load(f)
        job_padded = self.preprocess_text(job_description)
        resume_padded = self.preprocess_text(resume)
        combined_input = np.hstack((job_padded, resume_padded))
        score = self.matching_model.predict(combined_input)[0][0]
        return score

    def recommend(self, resume):
```

```
        if not hasattr(self, 'matching_model') or not hasattr(self, 'classification_model') or not
hasattr(self, 'tokenizer'):
            self.classification_model =
tf.keras.models.load_model(self.classification_model_path)
            self.matching_model = tf.keras.models.load_model(self.matching_model_path)
            with open(self.tokenizer_path, 'rb') as f:
                self.tokenizer = pickle.load(f)
        scores = []
        for job_desc, category in self.job_listings:
            score = self.predict_match(job_desc, resume)
            scores.append((job_desc, category, score))
        scores.sort(key=lambda x: x[2], reverse=True)
        return scores[:3]

matcher = JobMatcher()

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

@app.route('/classify', methods=['POST'])
def classify():
    try:
        job_description = request.form.get('job_description', '').strip()
        if not job_description:
            return render_template('index.html', error="Please fill in the Job Description field.")
        category = matcher.predict_category(job_description)
        return render_template('result.html', category=category)
    except ValueError as e:
        return render_template('index.html', error=str(e))
    except Exception as e:
        return render_template('index.html', error=f"An error occurred: {str(e)}")

@app.route('/recommend', methods=['POST'])
def recommend():
    try:
        resume = request.form.get('resume', '').strip()
        if not resume:
            return render_template('index.html', error="Please fill in the Resume field.")
        recommendations = matcher.recommend(resume)
        return render_template('recommend.html', recommendations=recommendations)
    except ValueError as e:
        return render_template('index.html', error=str(e))
    except Exception as e:
        return render_template('index.html', error=f"An error occurred: {str(e)}")

index_html = """
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Job Classifier and Recommender</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
            <div class="space-x-4">
                <a href="/" class="text-blue-600 hover:text-blue-800 font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
            </div>
        </div>
    </nav>
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Job Classifier and
Recommender</h1>
        {% if error %}
            <p class="text-red-600 mb-4 text-center">{{ error }}</p>
        {% endif %}
        <div class="mb-8">
            <h2 class="text-2xl font-semibold text-gray-800 mb-4">Classify a Job Posting</h2>
            <form action="/classify" method="post" class="space-y-4">
                <textarea id="job_description" name="job_description" rows="5"
placeholder="Enter job description..." class="w-full p-4 border rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-500">{{ request.form.get('job_description', '') }}</textarea>
                <button type="submit" class="w-full bg-blue-600 text-white px-6 py-3 rounded-lg
hover:bg-blue-700 transition duration-300">Classify</button>
            </form>
        </div>
        <div>
            <h2 class="text-2xl font-semibold text-gray-800 mb-4">Get Job
Recommendations</h2>
            <form action="/recommend" method="post" class="space-y-4">
                <textarea id="resume" name="resume" rows="5" placeholder="Enter your
resume..." class="w-full p-4 border rounded-lg focus:outline-none focus:ring-2
focus:ring-purple-500">{{ request.form.get('resume', '') }}</textarea>
                <button type="submit" class="w-full bg-purple-600 text-white px-6 py-3 rounded-lg
hover:bg-purple-700 transition duration-300">Recommend</button>
            </form>
        </div>
```

```
    </div>
</body>
</html>
"""

result_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Classification Result</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
            <div class="space-x-4">
                <a href="/" class="text-blue-600 hover:text-blue-800 font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
            </div>
        </div>
    </nav>
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Classification
Result</h1>
        <div class="p-4 bg-blue-50 rounded-lg">
            <p class="text-gray-600">Category: <span class="font-medium text-blue-600">{{
category }}</span></p>
        </div>
        <a href="/" class="block mt-4 text-blue-600 hover:text-blue-800 text-center">Back to
Home</a>
    </div>
</body>
</html>
"""

recommend_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Job Recommendations</title>
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="min-h-screen bg-gradient-to-r from-blue-500 to-purple-600 flex flex-col
items-center justify-center p-6">
    <nav class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-4 mb-6">
        <div class="flex justify-between items-center">
            <h1 class="text-2xl font-bold text-gray-800">Job Matcher Tool</h1>
            <div class="space-x-4">
                <a href="/" class="text-blue-600 hover:text-blue-800 font-medium">Home</a>
                <a href="/#classify" class="text-blue-600 hover:text-blue-800
font-medium">Classify Job</a>
                <a href="/#recommend" class="text-blue-600 hover:text-blue-800
font-medium">Get Recommendations</a>
            </div>
        </div>
    </nav>
    <div class="w-full max-w-4xl bg-white shadow-lg rounded-lg p-8">
        <h1 class="text-3xl font-semibold text-gray-800 mb-6 text-center">Job
Recommendations</h1>
        <div class="space-y-4">
            {% for job_desc, category, score in recommendations %}
                <div class="p-4 bg-purple-50 rounded-lg">
                    <p class="text-gray-600">{{ job_desc }} (Category: {{ category }}) - Match Score:
{{ "%.2f" % score }}</p>
                </div>
            {% endfor %}
        </div>
        <a href="/" class="block mt-4 text-blue-600 hover:text-blue-800 text-center">Back to
Home</a>
    </div>
</body>
</html>
"""

with open('templates/index.html', 'w') as f:
    f.write(index_html)
with open('templates/result.html', 'w') as f:
    f.write(result_html)
with open('templates/recommend.html', 'w') as f:
    f.write(recommend_html)

if __name__ == "__main__":
    app.run(debug=True)
```

3. `train_model.py`

- **Purpose**: Training script to pre-train the classification model on a larger dataset.
- **Key Features**:
    - Dataset: 45 job postings.
    - Model: Increases dropout to 0.7, trains for 20 epochs.
    - Evaluation: Precision: 0.57, Recall: 0.44, F1-Score: 0.36.
- **Limitations**:
    - No web interface.
    - Does not train the matching model.

Full Code:

```
from jobs import JobMatcher
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,
Dropout
import pickle

# Expanded dataset for classification (45 samples)
job_descriptions = [
    # IT (15 samples)
    "Software Engineer position requiring Python, Flask, and machine learning skills.",
    "Data Scientist role needing expertise in TensorFlow and data analysis.",
    "We are hiring a Software Engineer to work on machine learning projects requiring Python
and TensorFlow.",
    "Looking for an ideal candidate with experience in CNN models and text classification.",
    "We are hiring a Software Engineer to work on machine learning projects. The ideal
candidate should have experience with Python, Flask, and TensorFlow. Knowledge of CNN
models is a plus.",
    "Web Developer role requiring JavaScript and React.",
    "Backend Developer with experience in Node.js and MongoDB.",
    "AI Researcher needed for advanced algorithm development.",
    "Full Stack Developer proficient in Python and Angular.",
    "DevOps Engineer with AWS and Docker experience.",
    "Cybersecurity Analyst with expertise in network security.",
    "Database Administrator skilled in SQL and Oracle.",
    "Mobile App Developer with experience in Swift and Kotlin.",
    "Cloud Architect needed for AWS infrastructure design.",
    "Machine Learning Engineer with experience in PyTorch.",
    # Marketing (15 samples)
    "Marketing Manager needed with experience in digital campaigns and SEO.",
    "Content Writer for marketing team, skilled in copywriting and social media.",
    "Digital Marketing Specialist with expertise in Google Ads and analytics.",
    "Marketing Coordinator required for event planning and branding.",
    "Social Media Manager needed with experience in content creation and strategy.",
```

```
    "SEO Specialist needed with experience in keyword research.",
    "Brand Strategist for product launches and campaigns.",
    "Advertising Manager with experience in media buying.",
    "Marketing Analyst with data analysis skills.",
    "PR Specialist for public relations and media outreach.",
    "Email Marketing Specialist with experience in campaign automation.",
    "Market Research Analyst to analyze consumer trends.",
    "Influencer Marketing Manager for social media collaborations.",
    "Content Marketing Specialist with SEO writing skills.",
    "Product Marketing Manager for product launches.",
    # Healthcare (15 samples)
    "Nurse Practitioner required with experience in patient care and diagnostics.",
    "Medical Assistant needed for clinic, must have knowledge of EHR systems.",
    "Healthcare Administrator needed to manage hospital operations.",
    "Registered Nurse with 3 years of experience in emergency care.",
    "Physical Therapist required for patient rehabilitation and therapy.",
    "Doctor needed for hospital with 5 years of experience.",
    "Pharmacist to dispense medications and counsel patients.",
    "Surgeon needed for advanced surgical procedures.",
    "Radiologist with experience in imaging diagnostics.",
    "Therapist for mental health counseling and support.",
    "Pediatrician with experience in child healthcare.",
    "Cardiologist needed for heart-related treatments.",
    "Dental Hygienist for patient dental care.",
    "Medical Laboratory Technician for diagnostic testing.",
    "Occupational Therapist for patient rehabilitation."
]

categories = [
    "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT", "IT",
    "Marketing", "Marketing", "Marketing", "Marketing", "Marketing", "Marketing", "Marketing",
"Marketing", "Marketing", "Marketing", "Marketing", "Marketing", "Marketing", "Marketing",
"Marketing",
    "Healthcare", "Healthcare", "Healthcare", "Healthcare", "Healthcare", "Healthcare",
"Healthcare", "Healthcare", "Healthcare", "Healthcare", "Healthcare", "Healthcare",
"Healthcare", "Healthcare", "Healthcare"
]

class JobMatcherUpdated(JobMatcher):
    def build_classification_model(self):
        model = Sequential([
            Embedding(self.vocab_size, self.embedding_dim),
            Conv1D(128, 5, activation='relu'),
            GlobalMaxPooling1D(),
            Dense(64, activation='relu'),
            Dropout(0.7),
            Dense(3, activation='softmax')
        ])
```

```
        model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
        return model

matcher = JobMatcherUpdated()

all_texts = job_descriptions
matcher.tokenizer.fit_on_texts(all_texts)
sequences = matcher.tokenizer.texts_to_sequences(job_descriptions)
padded = pad_sequences(sequences, maxlen=matcher.max_length, padding='post',
truncating='post')
y = np.array([{"IT": 0, "Marketing": 1, "Healthcare": 2}[cat] for cat in categories])

X_train, X_test, y_train, y_test = train_test_split(padded, y, test_size=0.2, random_state=42)
matcher.classification_model.fit(X_train, y_train, epochs=20, validation_split=0.2,
batch_size=32)

y_pred = np.argmax(matcher.classification_model.predict(X_test), axis=1)
precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred, average='weighted',
zero_division=0)
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")

matcher.classification_model.save(matcher.classification_model_path)
with open(matcher.tokenizer_path, 'wb') as f:
    pickle.dump(matcher.tokenizer, f)

print("Classification model trained and saved successfully!")
```

How the Scripts Work Together

- **`train_matching.py`**: The deployed app, providing classification, recommendations, and a metrics endpoint with database support.
- **`jobs.py`**: An earlier implementation, replaced by `train_matching.py` for its database and metrics features.
- **`train_model.py`**: Pre-trains the classification model used by `train_matching.py`.

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 (top 3 jobs with match scores via cosine similarity).
    - View metrics (placeholder values: all 0.0).
- **Missing Features**:
    - Matching model not pre-trained.
- **Database**: SQLite database with 60 jobs (resets on Render's free tier).
- **Performance**:
    - Classification: F1-Score: 0.36.
    - Recommendations: Not evaluated.
- **UI**: Tailwind CSS-styled interface.

Future Improvements

- **Train the Matching Model**: Add a dataset of resume-job pairs and evaluate with metrics.
- **Add Metrics for Recommendations**: Compute Precision, Recall, F1-Score, and MAP.
- **Larger Dataset**: Collect more job postings (e.g., 1,000+).
- **Persistent Database**: Use an external database (e.g., PostgreSQL).
- **File Upload**: Allow resume uploads (e.g., PDF).

Deployment Instructions

- **Current URL**:https://ml-project-yzfq.onrender.com/
- **Start Command**: `waitress-serve --port=$PORT train_matching:app`
- **Steps to Redeploy**:
    1. Push changes to `https://github.com/NikhilRao1-ai/mlproject`.
    2. In the Render dashboard, select the `mlproject` service.
    3. Trigger a manual deploy with the latest commit.

How to Use

- **Access the App**: Visit https://ml-project-yzfq.onrender.com/.
- **Classify a Job Posting**: Enter a job description to get its category.
- **Get Job Recommendations**: Enter your resume to get the top 3 recommended jobs.
- **View Metrics**: Visit `/metrics` to see placeholder metrics.
- **Tips**:
    - Include relevant keywords in your resume.
    - Wait if the app spins down on Render's free tier.

Conclusion

This project fulfills the Machine Learning Assignment requirements:

- **Job Classification**: Achieved using a pre-trained CNN (F1-Score: 0.36).
- **Job Recommendations**: Implemented using cosine similarity (content-based filtering).
- **Evaluation**: Classification metrics computed; recommendation metrics pending.
- **Deployment**: Deployed on Render with a Tailwind CSS-styled interface.