Women's Job Recommendation System

Project: Women Job Recommendation System

Project Overview:

The Women Job Recommendation System is a web application designed to help women seeking job opportunities by providing job recommendations based on their skills. The system uses a pre-trained model to match user-provided skills with job descriptions, helping them find the most relevant job opportunities.

This project is built using Flask for the web framework, Scikit-learn for machine learning, and TF-IDF for text-based feature extraction. The system includes a model training script (`model\_training.ipynb`) for generating the recommendation model and an app script (`app.py`) for deploying the web service.

Project Structure:

- `app.py`: A Flask application that serves the job recommendations and integrates with the machine learning model.

- `model\_training.ipynb`: A Jupyter notebook for training the recommendation model using job skills data.

- `job\_skills.csv`: A dataset containing job descriptions and the required skills for each job.

- `templates/`: A folder containing the HTML templates for the user interface.

- `index.html`: The homepage where users can input their skills and receive job recommendations.

- `static/`: A folder for static assets like CSS and JavaScript files (if necessary).

Requirements:

- Python 3.x

- Flask

- pandas

- scikit-learn

- joblib

- Jupyter Notebook for model training

Installation Instructions:

1. Clone the Repository (if applicable):

If you haven't already, clone the repository to your local machine.

```bash

git clone <repository-url>

cd <project-folder>

```

2. Install Dependencies:

Create a virtual environment and install the required libraries.

```bash

python -m venv venv

source venv/bin/activate # For Linux/macOS

venv\Scripts\activate # For Windows

pip install -r requirements.txt

```

Or manually install the necessary libraries:

```bash

pip install flask pandas scikit-learn joblib

```

3. Prepare the Dataset:

Ensure that you have the `job\_skills.csv` file in the project directory. This dataset should contain job titles, job descriptions, and the required skills for each job. The CSV file should have at least the following columns:

- `job\_title`: The title of the job.

- `skills\_required`: A comma-separated list of skills required for the job.

- `job\_description`: A brief description of the job.

4. Model Training:

Run the `model\_training.ipynb` Jupyter notebook to generate the TF-IDF model for job recommendations.

1. Open the notebook in Jupyter.

2. Execute the cells to train the model.

3. The model will be saved as `job\_recommendation\_model.pkl`, which is later loaded in `app.py` for inference.

5. Running the Flask Application:

After setting up the model, you can run the Flask application to start the web server.

```bash

python app.py

```

The application will start a local web server on `http://127.0.0.1:5000/`. You can open this URL in a web browser to interact with the system.

6. Using the System:

- Open the browser and go to `http://127.0.0.1:5000/`.

- On the homepage, enter your skills as a comma-separated list (e.g., "Python, Java, Machine Learning").

- Click the "Recommend Jobs" button to receive a list of job titles that match the skills you entered.

- The server will respond with a list of recommended job titles based on the skills provided.

Code Walkthrough:

1. app.py:

- Flask App: This file sets up a simple Flask application with a home route (`/`) and a job recommendation route (`/recommend\_jobs`).

- Job Recommendation: The recommendation is done by loading a pre-trained model (`job\_recommendation\_model.pkl`) and comparing user-provided skills with job descriptions using TF-IDF vectorization and cosine similarity.

- API Endpoint: The `recommend\_jobs` endpoint receives the user input, processes it, and returns a list of recommended job titles based on skills.

2. model\_training.ipynb:

- Data Preprocessing: The `TfidfVectorizer` from scikit-learn is used to transform the `skills\_required` column of the dataset into a TF-IDF matrix.

- Model Saving: The trained TF-IDF vectorizer is saved as `job\_recommendation\_model.pkl` using the `joblib` library.

Steps to Run the Project:

1. Prepare your environment by setting up a virtual environment and installing dependencies.

2. Train the recommendation model by running the `model\_training.ipynb` Jupyter notebook to generate the `job\_recommendation\_model.pkl` file.

3. Run the Flask application using the `python app.py` command.

4. Use the web application to enter skills and get job recommendations.

Example Workflow:

1. The user inputs their skills in the web interface: "Python, Java, Data Science".

2. The Flask app processes the input and computes the cosine similarity between the user's skills and job descriptions.

3. The system returns a list of recommended job titles based on matching skills and the calculated similarity score.

Potential Improvements:

- Job Details: Include additional job details in the recommendations (e.g., company name, job location).

- User Feedback: Implement a feedback system where users can rate the job recommendations, improving future suggestions.

- Advanced NLP Models: Incorporate advanced NLP models like BERT or GPT-based models for better matching.

USING GEMINI AND ZAPIER

Ethical Considerations

This application follows ethical AI development practices, ensuring no gender bias in job recommendations. The system promotes transparency and privacy by using secure and responsible data handling mechanisms. The use of NLP is carefully designed to fairly evaluate the skills of users from all backgrounds.

1. Refining the Machine Learning Model

To enhance the job recommendation model, we'll use Gemini for natural language processing (NLP) to perform more sophisticated job-skill matching. We'll also use Zapier to notify users when new jobs that match their skills become available.

New Components:

- Gemini: Google’s NLP tool for advanced text matching.

- Zapier: Automation tool to notify users when new jobs are added to the system.

2. Required Libraries:

- Install Gemini SDK from Google’s API.

- Install Zapier Python library to handle automation:

```bash

pip install zapier-python

pip install google-gemini

```

3. Updated Project Structure:

- `app.py`: Flask server with Gemini API integration. A Flask application that serves the job recommendations and integrates with the machine learning model.

- `model\_training.ipynb`: Updated model using NLP-based job recommendation. A Jupyter notebook for training the recommendation model using job skills data.

- `zapier\_integration.py`: Script for Zapier to send notifications when new jobs are added.

- `job\_ skills.csv`: Updated dataset with job titles, skills, and descriptions.

- `templates/`: A folder containing the HTML templates for the user interface.

- `index.html`: The homepage where users can input their skills and receive job recommendations.

- `static/`: A folder for static assets like CSS and JavaScript files (if necessary).

4. Source Code

# File 1: `app.py` (with Gemini and Zapier Integration)

This script now uses Gemini for NLP-powered matching and Zapier to notify users about new job postings.

```python

from flask import Flask, render\_template, request, jsonify

import pandas as pd

import joblib

from google\_gemini import GeminiClient

import zapier\_python as zapier

app = Flask(\_\_name\_\_)

# Initialize Gemini Client

gemini\_client = GeminiClient(api\_key='YOUR\_GEMINI\_API\_KEY')

# Load pre-trained job recommendation model

model = joblib.load('job\_recommendation\_model.pkl')

skills\_df = pd.read\_csv('job\_data.csv')

# Zapier integration

zapier\_client = zapier.Zapier('YOUR\_ZAPIER\_WEBHOOK\_URL')

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/recommend\_jobs', methods=['POST'])

def recommend\_jobs():

skills = request.form['skills']

recommended\_jobs = recommend\_jobs\_for\_skills(skills)

# Send a Zapier notification about the job recommendation

zapier\_client.send({

'email': 'user@example.com',

'recommended\_jobs': recommended\_jobs

})

return jsonify(recommended\_jobs)

def recommend\_jobs\_for\_skills(skills):

user\_skills = set(skills.lower().split(','))

matched\_jobs = []

for idx, row in skills\_df.iterrows():

# Use Gemini to enhance the job recommendation based on NLP analysis

job\_skills = set(row['skills\_required'].lower().split(','))

job\_desc = row['job\_description']

# NLP similarity analysis using Gemini

similarity = gemini\_client.compare\_texts(skills, job\_desc)

if user\_skills.intersection(job\_skills) or similarity > 0.75:

matched\_jobs.append(row['job\_title'])

return matched\_jobs

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

```

Steps to get Gemini API Key

#Initialize Google PaLM Client

palm.configure(api\_key='YOUR\_PALM\_API\_KEY')

**Get an API Key**:

* Google PaLM API keys are typically accessible through Google Cloud services.
* Go to the Google Cloud Console, and navigate to the **API & Services** section.
* Enable the **PaLM API** (also known as the Generative Language API or the generativeai package).
  + To enable the PaLM API on Google Cloud, follow these steps:

**1. Set Up a Google Cloud Project**

* + Go to the Google Cloud Console.
  + If you don’t have a project already, click on **Select a project** at the top and then click **New Project**.
  + Name your project, and select **Create**.

**2. Enable the PaLM API**

* + In the Google Cloud Console, make sure you’re in your selected project.
  + Go to **APIs & Services** in the left-hand menu, then select **Library**.
  + Search for **Generative Language API** or **PaLM API**. You may see it listed as **PaLM API (generative language model)** or **Generative Language API**.
  + Click on it, then select **Enable**.

**3. Create an API Key**

* + After enabling the API, go to **APIs & Services** > **Credentials**.
  + Click **Create Credentials**, and choose **API Key**.
  + Copy the API key that is generated and store it securely, as this key will allow you to access the PaLM API.

**4. Set Up Billing (If Required)**

* + Some Google Cloud services may require a billing account. If prompted, set up billing by following the instructions, as this is often needed to enable access to certain APIs, especially generative models.

**5. Use the API Key in Your Application**

* + Use the API key to initialize the Google PaLM client in your code:
  + python
  + Copy code
  + import google.generativeai as palm
  + palm.configure(api\_key='YOUR\_PALM\_API\_KEY')
  + Once enabled, go to **Credentials** to create an API key for the PaLM API.

Steps to get Zapier Webhook URL

To get the YOUR\_ZAPIER\_WEBHOOK\_URL for integrating Zapier with your application, you’ll need to create a **Zap** that starts with a Webhook trigger. Here’s how to set it up:

**Steps to Create a Webhook in Zapier and Get the URL**

1. **Log in to Zapier**:
   * Go to [zapier.com](https://zapier.com/) and sign in to your account. If you don’t have an account, you’ll need to create one.
2. **Create a New Zap**:
   * On your Zapier dashboard, click **Create Zap**.
3. **Set Up the Webhook Trigger**:
   * For the **Trigger** app, search for **Webhooks by Zapier**.
   * Select the **Catch Hook** option. This will create a unique URL where you can send data.
   * Click **Continue**.
4. **Copy the Webhook URL**:
   * After choosing **Catch Hook** as the trigger, Zapier will provide a unique Webhook URL. Copy this URL—it’s your YOUR\_ZAPIER\_WEBHOOK\_URL.
   * This URL is where your application will send data to trigger the Zap.
5. **Set Up an Action**:
   * Next, you can choose any app for the **Action** step, such as sending an email, updating a spreadsheet, or notifying users in Slack.
   * Configure the action as desired, setting it up to respond to the data sent by the webhook.
6. **Test the Webhook**:
   * In your app, send a test payload to the webhook URL you copied. Zapier will capture the request data.
   * In Zapier, click **Test Trigger** to verify that the webhook is receiving data correctly.
7. **Turn On the Zap**:
   * Once your webhook and action are set up and tested, turn on your Zap. It will now run every time your application sends data to the YOUR\_ZAPIER\_WEBHOOK\_URL.

**Using the Webhook URL in Your Code**

After copying the YOUR\_ZAPIER\_WEBHOOK\_URL, you can use it in your code to send data to Zapier.

File 2: `model\_training.ipynb` (Refined ML Model)

This notebook refines the machine learning model by incorporating more advanced NLP-based techniques for text analysis using Gemini.

```python

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import joblib

from google\_gemini import GeminiClient

# Initialize Gemini Client

gemini\_client = GeminiClient(api\_key='YOUR\_GEMINI\_API\_KEY')

# Load job listing dataset

df = pd.read\_csv('job\_data.csv')

# Use Gemini for advanced text-based analysis on job descriptions and skills

def gemini\_similarity(skills, job\_desc):

return gemini\_client.compare\_texts(skills, job\_desc)

# Update model training to use Gemini's NLP text matching

df['similarity\_score'] = df.apply(lambda row: gemini\_similarity(row['skills\_required'], row['job\_description']), axis=1)

# Vectorize the skills and descriptions

vectorizer = TfidfVectorizer(stop\_words='english')

tfidf\_matrix = vectorizer.fit\_transform(df['skills\_required'] + ' ' + df['job\_description'])

# Save the model

joblib.dump(vectorizer, 'job\_recommendation\_model.pkl')

```

File 3: `zapier\_integration.py` (Sending Notifications using Zapier)

This script sends notifications to users when new jobs matching their skills are added.

```python

import zapier\_python as zapier

# Initialize Zapier client

zapier\_client = zapier.Zapier('YOUR\_ZAPIER\_WEBHOOK\_URL')

# Function to notify user when new jobs are available

def notify\_user(email, jobs):

zapier\_client.send({

'email': email,

'job\_list': jobs

})

# Example use

new\_jobs = ['Software Engineer', 'UX Designer']

notify\_user('user@example.com', new\_jobs)

```

5. Running the Application

Prerequisites:

- Install the necessary libraries:

```bash

pip install flask pandas scikit-learn joblib zapier-python google-gemini

```

Steps to Run the Code:

1. Start the Flask app:

```bash

python app.py

```

2. Run the Machine Learning Model:

- Open `model\_training.ipynb` in Jupyter Notebook.

- Run the notebook to train the model and save it as `job\_recommendation\_model.pkl`.

3. Test the Application:

- Open a browser and navigate to `http://127.0.0.1:5000`.

- Enter some skills like "Python, Java, Machine Learning."

- Receive job recommendations and see the output displayed in the browser.

- Users will also receive email notifications (via Zapier) when new jobs are added.