import pandas as pd

import re

import nltk

import string

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import seaborn as sns

# Download necessary NLTK resources

nltk.download("stopwords")

nltk.download("punkt")

nltk.download("wordnet")

# Load dataset

df = pd.read\_csv("path\_to\_resume\_data.csv")

# Check dataset structure

print("Dataset Overview:")

print(df.head())

# Check for missing values

print("\nMissing Values:\n", df.isnull().sum())

# Text Preprocessing Function

def preprocess\_text(text):

text = text.lower() # Convert to lowercase

text = re.sub(r"\s+", " ", text) # Remove extra spaces

text = re.sub(r"http\S+", "", text) # Remove URLs

text = re.sub(r"\d+", "", text) # Remove numbers

text = text.translate(str.maketrans("", "", string.punctuation)) # Remove punctuation

words = word\_tokenize(text) # Tokenize words

words = [word for word in words if word not in stopwords.words("english")] # Remove stopwords

lemmatizer = WordNetLemmatizer()

words = [lemmatizer.lemmatize(word) for word in words] # Lemmatization

return " ".join(words)

# Apply preprocessing to the "resume" column

df["cleaned\_resume"] = df["resume"].astype(str).apply(preprocess\_text)

# Encode job categories

categories = df["category"].unique()

category\_mapping = {category: idx for idx, category in enumerate(categories)}

df["category\_label"] = df["category"].map(category\_mapping)

# Feature Extraction using TF-IDF

vectorizer = TfidfVectorizer(max\_features=5000)

X = vectorizer.fit\_transform(df["cleaned\_resume"]) # Transform text into numerical features

y = df["category\_label"] # Target variable

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Train Logistic Regression Model

clf = LogisticRegression(max\_iter=1000)

clf.fit(X\_train, y\_train)

# Model Evaluation

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("\nModel Accuracy:", accuracy)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Confusion Matrix Visualization

plt.figure(figsize=(10, 6))

sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt="d", cmap="Blues", xticklabels=categories, yticklabels=categories)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# Save the trained model and vectorizer

import joblib

joblib.dump(clf, "resume\_classifier.pkl")

joblib.dump(vectorizer, "tfidf\_vectorizer.pkl")

print("\nModel and Vectorizer Saved Successfully!")

import streamlit as st

from PyPDF2 import PdfReader

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

*# Function to extract text from PDF*

**def** extract\_text\_from\_pdf(file):

    pdf = PdfReader(file)

    text = ""

    for page in pdf.pages:

        text += page.extract\_text()

    return text

*# Function to rank resumes based on job description*

**def** rank\_resumes(job\_description, resumes):

*# Combine job description with resumes*

    documents [job\_description] + resumes

    vectorizer = TfidfVectorizer().fit\_transform(documents)

    vectors vectorizer.toarray()

*# Calculate cosine similarity*

    job\_description\_vector = vectors[0]

    resume\_vectors = vectors [1:]

    cosine\_similarities = cosine\_similarity ([job\_description\_vector], resume\_vectors).flatten()

    return cosine\_similarities

*# Streamlit app*

st.title("AI Resume Screening & Candidate Ranking System")

*# Job description input*

st.header("Job Description")

job\_description= st.text\_area ("Enter the job description")

*# File uploader*

st.header("Upload Resumes")

uploaded\_files = st.file\_uploader("Upload PDF files", type=["pdf"], accept\_multiple\_files=True)

if uploaded\_files and job\_description:

    st.header("Ranking Resumes")

    resumes = []

    for file in uploaded\_files:

        text = extract\_text\_from\_pdf(file)

        resumes.append(text)

*# Rank resumes*

    scores = rank\_resumes(job\_description, resumes)

*# Display scores*

    results = pd.DataFrame({"Resume": [file.name for file in uploaded\_files], "Score": score })

    results = results.sort\_values(by="Score", ascending=False)

    st.write(results)