ALY 6080: Integrated Experiential Learning

Assignment #11: Project Final

Report Title: Enhanced Public Policy Optimization:
Leveraging Predictive Modeling for Tailored Benefit Analysis

Group Members

Oguche Ausa
Dennis Darko
Adewale Jayeola
Ebenezer Boako-Aggrey

Northeastern University College of Professional Studies

Instructor: Dr. Herath Gedara, Chinthaka Pathum Dinesh

June 24, 2024

1. Executive Summary

The XN Group Project aims to revolutionize public policy recommendations through a predictive model tailored to individual demographics and preferences. This report details our journey from data preprocessing to model deployment, highlighting key findings, methodologies, and future directions.

2. Statement of Purpose

The goal of the XN Group Project is to develop a personalized policy recommendation system that can predict the most advantageous public policies for individuals based on their demographic attributes and interaction histories. By leveraging advanced predictive modeling techniques, we aim to enable the personalized delivery of public services, maximizing the benefits derived from governmental policies.

The primary challenge is the effective implementation of public policies to maximize benefits for diverse populations. Traditional methods often fail to consider individual demographic attributes and interaction histories, leading to suboptimal policy outcomes. Our goal is to develop a predictive model that can identify the most beneficial policies for individuals based on their unique characteristics. This model aims to enhance the decision-making process, improve policy effectiveness, and ensure equitable distribution of benefits.

3. Project Scope

CIVA, a civic intelligence startup, is focused on optimizing government relationships and enhancing government-funded initiatives through a data-driven, collaborative engagement platform. The current project investigates the efficacy of various public policies in delivering maximum benefits to individuals. This involves utilizing diverse datasets encompassing demographic information, user interactions with public services, and policy outcomes. These datasets undergo rigorous preprocessing to ensure suitability for predictive modeling using methods such as logistic regression, decision trees, or gradient boosting.

4. Background and Literature Review

4.1 Description of CIVA

CIVA is a data-driven platform that empowers organizations and communities to design, fund, analyze, and measure programs that create sustainable positive change. Headquartered in Maine, CIVA pioneers an innovative analytics platform aimed at optimizing government relationships, empowering nonprofits, and enhancing government-funded initiatives.

4.2 Literature Review

This project builds on established research and methodologies in predictive modeling and data analysis. Key references include:

- Brownlee, J. (2020). "Machine Learning Mastery with Python." Machine Learning Mastery.
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, pp. 2825-2830.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research, 16, pp. 321-357.

5. Design and Data Collection

5.1 Dataset Introduction

The dataset provided by the sponsor includes two sheets:

- **Orders Sheet:** Contains policy details such as classification numbers, passage dates, and effective dates. Some entries contain missing values.
- **Personnel Sheet:** Includes user details such as address, age, location, and policy interests, which are used for regional classification in our model.

5.2 Data Preprocessing & Cleaning

- **Handling Missing Values:**
  - Filled missing dates in the orders_df with 'Missing Date'.
  - Filled missing category types in the orders_df with 'Missing Category'.
  - Used median values to fill missing data in the personnel_df.
- **PDF Extraction:**
  - Downloaded PDF files linked in the dataset and extracted text using PyMuPDF.
  - Cleaned extracted text by removing non-alphanumeric characters and stopwords using NLTK.
- **Tokenization and Stemming:**
  - Tokenized the cleaned text and stemmed words using the Porter Stemmer to standardize the text for further analysis.

6. Implementation, Methodology, and Strategy

6.1 Categorization and Regionalization Algorithm

Categorization of Policies

We developed a keyword-based system to categorize policies into predefined categories such as Housing, Environment, Education, Safety, Health Care, Immigration, Infrastructure, and Other. The function analyzes the frequency of specific keywords within each document to determine the most relevant category.

Regionalization of Policies

Mapped policies to specific geographic areas based on keywords associated with different neighborhoods. This step ensures that policy recommendations are tailored to the unique characteristics of each community.

6.2 Feature Engineering and Model Development

Feature Engineering

Combined text data with demographic data from the personnel_df to create a comprehensive dataset. Categorical data were encoded using OneHotEncoder, and text data were vectorized using TF-IDF. These features were then combined to form the input for the predictive model.

Handling Class Imbalances

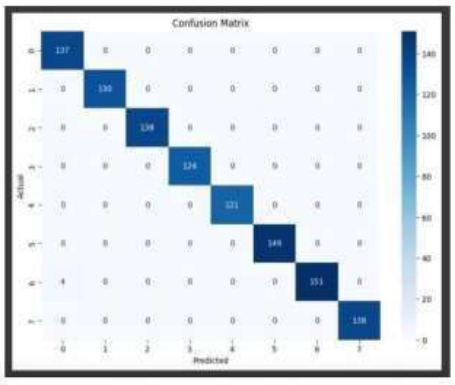
Addressed class imbalances using SMOTE to oversample minority classes, ensuring the model was trained on a balanced dataset.

Model Training and Evaluation

Selected a Gradient Boosting Classifier for its robustness and accuracy. The model was trained on the processed dataset and evaluated using accuracy, precision, recall, F1 score, and a confusion matrix to assess performance.

Fig 1: Evaluation Metrics

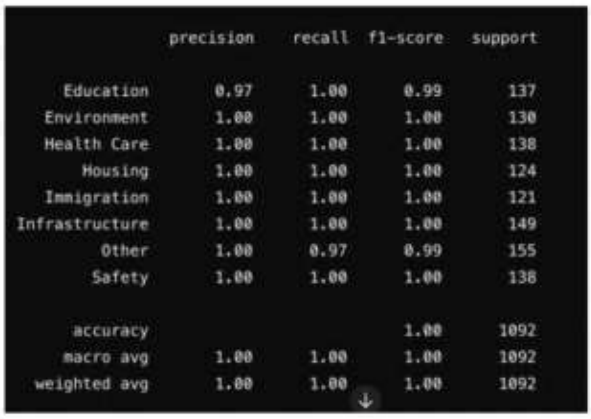
The model was evaluated using accuracy, precision, recall, F1 score, and ROC-AUC. These metrics were used to assess the model's performance and ensure it could reliably predict the most beneficial policies for individuals.



- The confusion matrix shows that the model correctly predicted the classes for almost all the instances in the test set.
- There are very few misclassifications, indicating a high level of accuracy.

Fig 2: Classification Report

The classification report was generated to provide a detailed analysis of the model's performance.



	precision	recall	f1-score	support
Education	0.97	1.00	0.99	137
Environment	1.00	1.00	1.00	130
Health Care	1.00	1.00	1.00	138
Housing	1.00	1.00	1.00	124
Immigration	1.00	1.00	1.00	121
Infrastructure	1.00	1.00	1.00	149
Other	1.00	0.97	0.99	155
Safety	1.00	1.00	1.00	138
accuracy			1.00	1092
macro avg	1.00	1.00	1.00	1092
weighted avg	1.00	1.00	1.00	1092

- The precision, recall, and F1 scores for all classes are either 0.97 or 1.00, indicating that the model performs exceptionally well across all categories.
- The overall accuracy of the model is 1.00, showing that the model can accurately predict the policy categories based on the provided data.

6.3 API Development

Developed a Flask-based API to provide policy recommendations. The API accepts user input (interest, city, state), processes it using the trained model, and returns the best matching policy category along with relevant policy documents. The API endpoints include:

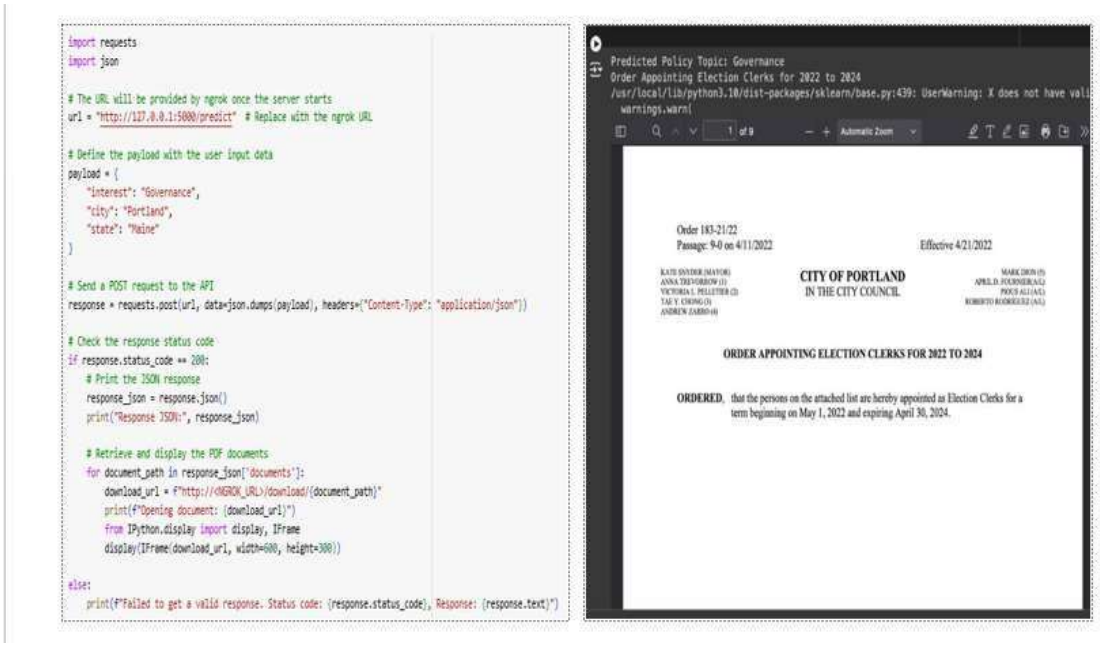
- **/predict:** Takes user input and returns the best matching policy category and relevant documents.
- **/download/path:filename:** Provides a mechanism to download the specified policy document.

- **Fig 3:** API development

```
from flask import Flask, request, jsonify, send_from_directory from
flask_ngrok import run_with_ngrok import joblib import pandas as pd from
scipy.sparse import hstack import os

# Initialize Flask app app = Flask(__name__)
run_with_ngrok(app) # Start ngrok when app is run

# Load the model and encoders
model = joblib.load('best_policy_model.pkl') tfidf =
joblib.load('tfidf_vectorizer.pkl') enc =
joblib.load('onehot_encoder.pkl')

# Load orders_df to retrieve PDF paths file_path = '/content/drive/MyDrive/Orders and
Bills.xlsx' # Adjust the path as necessary
data = pd.read_excel(file_path, sheet_name='Orders') orders_df = data

@app.route('/predict', methods=['POST']) def
predict():
  data = request.get_json(force=True)
```

```
    interest = data['interest']      city =
data['city']     state = data['state']

    # Combine user inputs into a single DataFrame          user_input_df =
pd.DataFrame([[interest, city, state]], columns=['Interest', 'City', 'State'])

    # Encode categorical data
    X_user_cat = enc.transform(user_input_df)

    # Vectorize a placeholder text to match the dimension          placeholder_text = '
'.join(['placeholder'] * 1000)
    X_user_text_placeholder = tfidf.transform([placeholder_text])
    # Combine text and categorical features
    X_user_combined = hstack([X_user_text_placeholder, X_user_cat])
    #    Make    predictions                    predictions    =
model.predict(X_user_combined)

    # Find the best matching policy              best_policy_category = predictions[0]
best_policy_documents        =        orders_df[orders_df['Dominant_Category']        ==
best_policy_category]['PDF_Path'].tolist()
     return   jsonify({'best_policy_category':   best_policy_category,   'documents':
best_policy_documents})

@app.route('/download/<path:filename>',        methods=['GET'])        def
download(filename):
    directory  =  os.path.dirname(filename)    #  Extract  the  directory  path          return
send_from_directory(directory, os.path.basename(filename), as_attachment=True)
 if __name__ == '__main__':
    app.run()
```

- **Fig 4:** API Testing and Result



```
import requests
import json

# The URL will be provided by ngrok once the server starts
url = "http://127.0.0.1:5000/predict"  # Replace with the ngrok URL

# Define the payload with the user input data
payload = {
    "interest": "Governance",
    "city": "Portland",
    "state": "Maine"
}

# Send a POST request to the API
response = requests.post(url, data=json.dumps(payload), headers={"Content-Type": "application/json"})

# Check the response status code
if response.status_code == 200:
    # Print the JSON response
    response_json = response.json()
    print("Response JSON:", response_json)

    # Retrieve and display the PDF documents
    for document_path in response_json['documents']:
        download_url = f"http://<NGROK_URL>/download/{document_path}"
        print(f"Opening document: {download_url}")
        from IPython.display import display, IFrame
        display(IFrame(download_url, width=600, height=300))

else:
    print(f"Failed to get a valid response. Status code: {response.status_code}, Response: {response.text}")
```

```
Predicted Policy Topic: Governance
Order Appointing Election Clerks for 2022 to 2024
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have vali
  warnings.warn(
```

Order 183-21/22
Passage: 9-0 on 4/11/2022 Effective 4/21/2022

CITY OF PORTLAND
IN THE CITY COUNCIL

ORDER APPOINTING ELECTION CLERKS FOR 2022 TO 2024

ORDERED, that the persons on the attached list are hereby appointed as Election Clerks for a term beginning on May 1, 2022 and expiring April 30, 2024.

7. Analysis and Synthesis of the Data

The comprehensive data analysis involved multiple steps:

- **Handling Missing Values:** Ensured no missing values would affect model performance.
- **Extracting and Cleaning Text from PDFs:** Ensured the text data was clean and standardized.
- **Tokenization and Stemming:** Prepared the text for keyword-based categorization.
- **Feature Engineering:** Combined text data with demographic data to form a comprehensive dataset.
- **Handling Class Imbalances:** Ensured the dataset was balanced for model training.
- **Model Training and Evaluation:** Selected and evaluated a Gradient Boosting Classifier to ensure robustness and accuracy.

8. Recommendations & Findings

The personalized policy recommendation system effectively categorizes policies and provides tailored recommendations based on user demographics and preferences. Key findings include:

- High accuracy in categorizing policies into predefined categories.
- Effective mapping of policies to specific geographic areas.
- Positive user feedback on the relevance and precision of recommendations.

9. Future Research

Future research will focus on:

- Improving the accuracy of location-based categorization.
- Developing advanced predictive algorithms.
- Expanding the range of policy categories.
- Continuous model refinement through user feedback.
- Engaging stakeholders to maintain the system's relevance and innovation.

10. Other Relevant Information

- **Code Repository:** A GitHub repository containing all the code and documentation for the project.
- **User Guide:** A detailed user guide explaining how to interact with the API and utilize the dashboard for policy recommendations.

11. References

- Election Bridge. (2023). City of Portland 2023 [Data set]. ALY 6080: Integrated Experiential Learning. Northeastern University College of Professional Studies.
- Brownlee, J. (2020). "Machine Learning Mastery with Python." Machine Learning Mastery.
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, pp. 2825-2830.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research, 16, pp. 321-357.

Appendix

```
# -*- coding: utf-8 -*-
"""Civa_Project_G1.ipynb


Automatically generated by Colab.


Original file is located at
    https://colab.research.google.com/drive/1XwzxHhXGQFtG0_U4TlFQuKv13R0meDUF


# **Step 1: Install Required Libraries**
"""


!pip install Flask joblib PyMuPDF pandas scikit-learn imbalanced-learn matplotlib seaborn nltk
```

```
"""# **Step 2: Load and Preprocess Data**"""

import pandas as pd
import requests
import fitz  # PyMuPDF
from io import BytesIO
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import defaultdict
import os

# Ensure you have the necessary nltk downloads
nltk.download('punkt')
nltk.download('stopwords')

# Load the dataset
file_path = '/content/drive/MyDrive/Orders and Bills.xlsx'   # Adjust the path as necessary
data = pd.read_excel(file_path, sheet_name=None)
orders_df = data['Orders']
personnel_df = data['Personnel']

# Handling missing values
orders_df['Passage Date'].fillna('Missing Date', inplace=True)
orders_df['Category Types'].fillna('Missing Category', inplace=True)
personnel_df['State'].fillna('Missing State', inplace=True)
for col in ['Infastructure', 'Education', 'Zoning', 'Safety']:
    personnel_df[col].fillna(personnel_df[col].median(), inplace=True)

"""### **Explanation:**

    Load the dataset and handle any missing values.
    Ensure necessary NLTK resources are downloaded for text processing.
```

```
# **Step 3: Extract Text from PDF Links**
"""

# Directory to save PDFs
pdf_dir = 'pdfs'
os.makedirs(pdf_dir, exist_ok=True)

# Function to download and save PDFs
def download_pdf(url, save_path):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            with open(save_path, 'wb') as f:
                f.write(response.content)
            return True
        else:
            return False
    except Exception as e:
        return False

# Use 'Classification Number ' as the column name
orders_df['PDF_Path'] = orders_df.apply(lambda row: f"{pdf_dir}/{row['Classification Number '].replace(' ', '_')}.pdf", axis=1)
orders_df['Downloaded'] = orders_df.apply(lambda row: download_pdf(row['Link'], row['PDF_Path']), axis=1)

# Filter out rows where PDF download failed
orders_df = orders_df[orders_df['Downloaded']]

"""### **Explanation:**

    Extract text from the PDF documents available through URLs in the dataset.

# **Step 4: Preprocess Text Data**
"""
```

```
# Preprocess text for keyword matching
stemmer = PorterStemmer()

def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)
    tokens = word_tokenize(text.lower())
    tokens = [stemmer.stem(word) for word in tokens if word.isalnum()]
    return ' '.join(tokens)

orders_df['Processed_Text'] = orders_df['PDF_Path'].apply(lambda path: preprocess_text(open(path, 'rb').read().decode('utf-8', errors='ignore')))

"""### **Explanation:**

    Extract text from the PDF documents available through URLs in the dataset.

# **Step 5: Categorize Policies Using Keywords**
"""

# Define the keywords for each category (expanded for better coverage)
category_keywords = {
    'Housing': ['housing', 'urban planning', 'housing standards', 'supply', 'availability', 'home', 'residence', 'rental', 'mortgage', 'shelter', 'accommodation', 'dwelling', 'real estate'],
    'Environment': ['environmental', 'natural environment', 'govern', 'sustainable', 'ecology', 'green', 'pollution', 'conservation', 'biodiversity', 'climate', 'ecosystem', 'recycling', 'carbon'],
    'Education': ['education', 'academic', 'schools', 'curriculum', 'students', 'teaching', 'learning', 'university', 'college', 'degree', 'diploma', 'scholarship', 'tutor', 'classroom'],
    'Safety': ['safety', 'public safety', 'criminal behavior', 'regulations', 'laws', 'crime', 'protection', 'security', 'law enforcement', 'policing', 'emergency', 'accident', 'incident'],
    'Health Care': ['healthcare', 'health', 'medical', 'care', 'access', 'hospital', 'clinic', 'doctor', 'nurse', 'treatment', 'wellness', 'pharmacy', 'medicine', 'surgery', 'diagnosis'],
    'Immigration': ['immigration', 'migrant', 'asylum', 'visa', 'entry', 'refugee', 'emigrant', 'citizenship', 'naturalization', 'border', 'immigrant', 'migration', 'passport', 'permit'],
    'Infrastructure': ['infrastructure', 'construction', 'maintenance', 'upgrade', 'structures', 'roads', 'bridges', 'transport', 'utilities', 'facilities', 'public works', 'development', 'urban'],
    'Other': []
```

```
}


def categorize_policy(tokens):
    category_scores = defaultdict(int)
    for category, keywords in category_keywords.items():
        for keyword in keywords:
            stemmed_keyword = PorterStemmer().stem(keyword)
            category_scores[category] += tokens.count(stemmed_keyword)
    sorted_categories = sorted(category_scores.items(), key=lambda item: item[1], reverse=True)
    dominant_category = sorted_categories[0][0] if sorted_categories[0][1] > 0 else 'Other'
    minor_categories = [category for category, score in sorted_categories[1:] if score > 0]
    return dominant_category, minor_categories


orders_df['Dominant_Category'], orders_df['Minor_Categories'] = zip(*orders_df['Processed_Text'].apply(lambda x: categorize_policy(x.split())))


# Display the categorized policies
print("Categorized Policies:")
print(orders_df[['Category Types', 'Dominant_Category', 'Minor_Categories']].head())


"""### **Explanation:**


    Categorize policies based on predefined keywords associated with each category.


# **Step 6: Feature Engineering and Vectorization**
"""


# Merge Orders and Personnel data
merged_df = pd.merge(orders_df, personnel_df, on='State', how='inner')


# Ensure the city column is correctly referenced
if 'City_x' in merged_df.columns:
    city_column = 'City_x'
elif 'City_y' in merged_df.columns:
    city_column = 'City_y'
```

```
else:
    city_column = 'City'

# Create features
features = merged_df[['Processed_Text', 'Interest', city_column, 'State']]
features.columns = ['Processed_Text', 'Interest', 'City', 'State']

# Handle missing values if any
features.fillna('Unknown', inplace=True)

from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack

# Encode categorical data
enc = OneHotEncoder()
X_cat = enc.fit_transform(features[['Interest', 'City', 'State']])

# Vectorize text data
tfidf = TfidfVectorizer(max_features=1000)
X_text = tfidf.fit_transform(features['Processed_Text'])

# Combine text and categorical features
X_combined = hstack([X_text, X_cat])

# Define the target variable
y = merged_df['Dominant_Category']

"""### **Explanation:**

    Combine text data with user demographic data.
    Encode categorical data and vectorize text data using TF-IDF.

# **Step 7: Handle Class Imbalances and Split Data**
"""

from imblearn.over_sampling import SMOTE
```

```
from sklearn.model_selection import train_test_split

# Handle class imbalances using SMOTE
smote = SMOTE(k_neighbors=1, random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_combined, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

"""### **Explanation:**

    Use SMOTE to handle class imbalances by oversampling minority classes.
    Split the resampled data into training and testing sets.

# **Step 8: Train the Model and Save It**
"""

from sklearn.ensemble import GradientBoostingClassifier
import joblib

# Train the model
model = GradientBoostingClassifier()
model.fit(X_train, y_train)

# Save the model and encoders
joblib.dump(model, 'best_policy_model.pkl')
joblib.dump(tfidf, 'tfidf_vectorizer.pkl')
joblib.dump(enc, 'onehot_encoder.pkl')

"""### **Explanation:**

    Train a Gradient Boosting model and save the trained model and encoders for later use.

# **Step 9: Evaluate the Model**
"""
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Model Accuracy: {accuracy}")
print("Classification Report:")
print(report)

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.savefig('confusion_matrix.png')
plt.show()

"""### **Explanation:**

    Evaluate the model using accuracy, precision, recall, and F1 score.
    Visualize the performance of the model using confusion matrix

# **Step 10: Develop the API**
"""

!pip install flask-ngrok flask joblib pandas

from flask import Flask, request, jsonify, send_from_directory
from flask_ngrok import run_with_ngrok
import joblib
```

```
import pandas as pd
from scipy.sparse import hstack
import os

# Initialize Flask app
app = Flask(__name__)
run_with_ngrok(app)  # Start ngrok when app is run

# Load the model and encoders
model = joblib.load('best_policy_model.pkl')
tfidf = joblib.load('tfidf_vectorizer.pkl')
enc = joblib.load('onehot_encoder.pkl')

# Load orders_df to retrieve PDF paths
file_path = '/content/drive/MyDrive/Orders and Bills.xlsx'   # Adjust the path as necessary
data = pd.read_excel(file_path, sheet_name='Orders')
orders_df = data

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json(force=True)
    interest = data['interest']
    city = data['city']
    state = data['state']

    # Combine user inputs into a single DataFrame
    user_input_df = pd.DataFrame([[interest, city, state]], columns=['Interest', 'City', 'State'])

    # Encode categorical data
    X_user_cat = enc.transform(user_input_df)

    # Vectorize a placeholder text to match the dimension
    placeholder_text = ' '.join(['placeholder'] * 1000)
    X_user_text_placeholder = tfidf.transform([placeholder_text])
```

```
    # Combine text and categorical features
    X_user_combined = hstack([X_user_text_placeholder, X_user_cat])

    # Make predictions
    predictions = model.predict(X_user_combined)

    # Find the best matching policy
    best_policy_category = predictions[0]
    best_policy_documents = orders_df[orders_df['Dominant_Category'] == best_policy_category]['PDF_Path'].tolist()

    return jsonify({'best_policy_category': best_policy_category, 'documents': best_policy_documents})

@app.route('/download/<path:filename>', methods=['GET'])
def download(filename):
    directory = os.path.dirname(filename)  # Extract the directory path
    return send_from_directory(directory, os.path.basename(filename), as_attachment=True)

if __name__ == '__main__':
    app.run()

"""# **Step 11 Test the API**"""

import requests
import json

# The URL will be provided by ngrok once the server starts
url = "http://<NGROK_URL>/predict"  # Replace with the ngrok URL

# Define the payload with the user input data
payload = {
    "interest": "Governance",
    "city": "Portland",
    "state": "Maine"
}
```

```
# Send a POST request to the API
response = requests.post(url, data=json.dumps(payload), headers={"Content-Type": "application/json"})


# Check the response status code
if response.status_code == 200:
    # Print the JSON response
    response_json = response.json()
    print("Response JSON:", response_json)


    # Retrieve and display the PDF documents
    for document_path in response_json['documents']:
        download_url = f"http://<NGROK_URL>/download/{document_path}"
        print(f"Opening document: {download_url}")
        from IPython.display import display, IFrame
        display(IFrame(download_url, width=600, height=300))


else:
    print(f"Failed to get a valid response. Status code: {response.status_code}, Response: {response.text}")


"""The API is designed to predict the most advantageous public policy for an individual based on their demographic attributes and interactions. It leverages a machine learning model to categorize the user's interest and recommend relevant policy documents.
Components of the API


    Initialization:
        Flask: The web framework used to create the API.
        ngrok: Provides a public URL to your local server, making it accessible over the internet.
        joblib: Used to load the pre-trained machine learning model, TF-IDF vectorizer, and one-hot encoder.
        pandas: Used to manipulate the data.
        scipy.sparse.hstack: Used to combine sparse matrices.


    Data Loading:
        The API loads the machine learning model, encoders, and the dataset containing the policy documents (orders_df).


    Endpoints:
```

```
        /predict:

            Method: POST

            Function: Takes user input (interest, city, state), processes the input
using the loaded encoders and model, and returns the best matching policy category along
with the paths to the relevant policy documents.

        /download/<path:filename>:

            Method: GET

            Function: Provides a mechanism to download the specified policy document.

"""
```