Automatic Topic Tagging For Documents Using An API

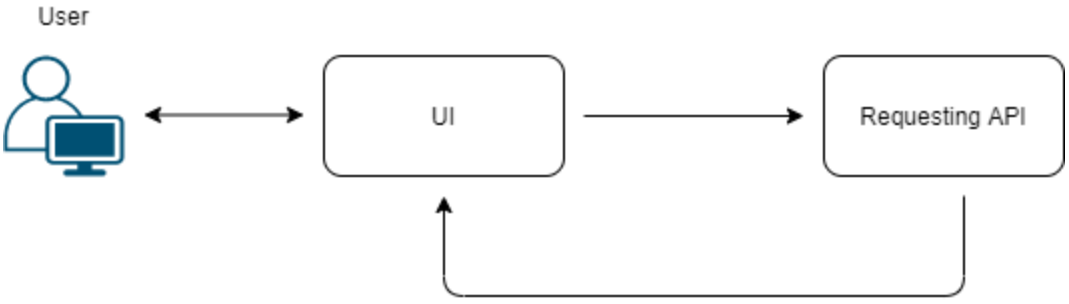
Project Description:

Businesses deal with large volumes of unstructured text every day. Think about all of the customer interactions and brand mentions in emails, support tickets, social media posts, online feedback and reviews, and other information that an organization sends and receives. The list is endless. When it comes to analyzing huge amounts of text data, it's just too big a task to do manually. It's also tedious, time-consuming, and expensive. Manually sorting through large amounts of data is more likely to lead to mistakes and inconsistencies.

Topic analysis is a Natural Language Processing (NLP) technique that allows us to automatically extract meaning from texts by identifying recurrent themes or topics. Topic analysis models enable you to sift through large sets of data and identify the most common and most important topics in an easy, fast, and completely scalable way.

In this guided lab we will be building a web application using flask which accepts text data from the user and gets the top topic tags using Topic Tagging API.

Architecture:



Project Objectives

What is an API?

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an

app like Facebook, send an instant message, or check the weather on your phone, you're using an **API**.

How API can be integrated into a web application ?

It allows a user **to** communicate **with a web**-based **web** tool or **application**. **With** an **API**, clients **can** use an interface **to** request something from an **app**. Then, the **application will** pass the data **to** an **API**, which **will** interpret the information and provide a response.

How to build a basic flask app?

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file. Flask is also extensible and doesn't force a particular directory structure or require complicated boilerplate code before getting started.

What is Topic Tagging?

Topic tags allow users to filter between forums and display **topics** with a specific **topic tag**. If **topic tagging** is enabled, when a user creates a **topic** they have the ability to add specific **tags** to quickly explain what a post is about and it also helps users find related **topics** based on those **tags**.

RapidAPI Account Creation

we will be creating an account in RapidAPI which will be used for getting Auto-Tagging API

Subscription Of Application Oriented API

- Navigate to the below link, to subscribe for the Topic Tagging API.

Link: https://rapidapi.com/twinword/api/topic-tagging

- Copy the python code and API details to integrate to Flask Application.

Building A Flask Application

We will be integrating API to a web application so that users can also use it for getting Tags in the UI.

Importing Of Libraries And Routing The HTML Pages

We will be using python for server-side scripting. Let's see step by step process for writing backend code.

Importing flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

- Import the following libraries

```
# Importing of Libraries
from flask import Flask, request, render_template
# Importing numpy library
import numpy as np
# Importing regular expression Library
import re
# Importing the requests library
import requests
# Importing json library
import json
# Importing csv library
import csv
# Importing pandas library
import pandas as pd
```

- Rendering to HTML page

Here we will be using declared constructor to route to the HTML page which we

have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered. Whenever you enter the values from the HTML page the values can be retrieved using **POST** Method.

```
# Defining the app
app = Flask(__name__)
```

- Defining a function for the output.

```
# Defining the output function
def check(output):
    url = "https://rapidapi.p.rapidapi.com/generate/"
    payload = {"text": output}
    #print(payload)
    headers = {
    'x-rapidapi-key': "XXXXXXXX", # Add Your API
    'x-rapidapi-host': "twinword-topic-tagging.p.rapidapi.com"
    }
    response = requests.request("GET", url, headers=headers, params=payload)
    print(response.text)
    return response.json()["topic"]
    #return var
```

Note: Ensure that, you have added the generated API.

- Routing of the HTML pages.

```
# Routing home page
@app.route('/')
def home():
    return render_template('home.html')

#home page with summerizer
@app.route('/summarizer')
def summarizer():
    return render_template('summarizer.html')
```

- Defining the summarizer function

```
#summarizer page
@app.route('/summarize',  methods=['POST'])
def summarize():
    output = request.form['output']
    output=re.sub("[^a-zA-Z.,]"," ",output)
    print(output)
    essay = check(output)

    data_file = open('data_file.csv', 'w')
    csv_writer = csv.writer(data_file)
    count = 0
    for emp in essay:
        print(essay[emp])
        essay[emp] = round(essay[emp],4)
        if count == 0:
            # Writing headers of CSV file
            header = ['Type','Probability']
            csv_writer.writerow(header)
            count += 1
```

```
        # Writing data of CSV file
            d = [emp,essay[emp]]
            print(d)
            csv_writer.writerow(d)
        data_file.close()
        df = pd.read_csv("data_file.csv")
        temp = df.to_dict('records')
        columnNames = df.columns.values
        return render_template('summary.html',records=temp, colnames=columnNames)
```

- The summarizer function, will convert the output into the regular expression using re library and open the saved CSV file.
- Calling of Main Function

  This is used to run the application in localhost.

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

Running Of Flask Application

- Open anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type “python app.py” command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/**
- Copy that localhost URL and open that URL in the browser. It does navigate you to them where you can view your web page.

```
(base) C:\Users\mail2>cd C:\Users\mail2\Downloads\Topic Tagging

(base) C:\Users\mail2\Downloads\Topic Tagging>python app1.py
 * Serving Flask app "app1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 249-788-636
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Your UI will look like