

Automatic Topic Tag Recommendation

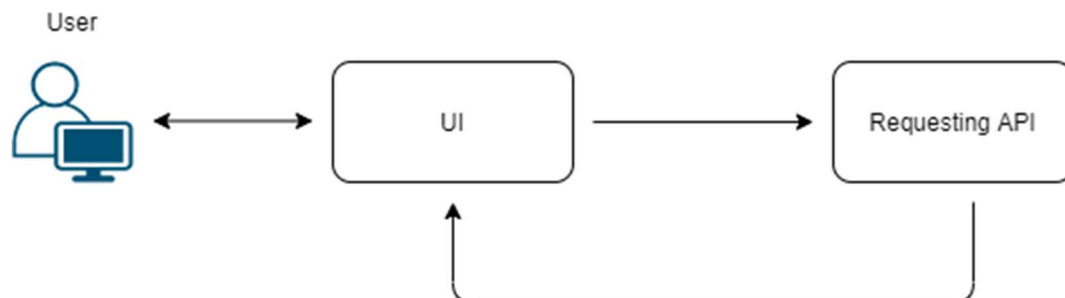
Project Idea:

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. Plus, it doesn't scale well.

Solution:

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.

Architecture:



Project Work Flow:

1. Generate a RapidAPI
2. Requesting an API
3. Display the Output

Learning Outcomes:

By the end of this project:

- You'll be able to work on API's Integration.
- Flask Application Development.

Milestones:

To accomplish this, complete all the milestones & activities listed below

1. Installation of Pre-requisites
 - Installation of Anaconda IDE / Anaconda Navigator.
 - Installation of Python packages.
2. RapidAPI Account Creation.
3. Subscription of Application Oriented API.
4. Building a Flask Application.
 - Importing of Libraries and routing the html pages
 - Running of flask Application.

Milestone 1: Installation of Pre-requisites

To complete the project successfully, you need to install following software & packages:

Activity 1: Install Anaconda IDE / Anaconda Navigator.

- In order develop a solution to this problem statement, we need an environment to write and test the code.
- We use Anaconda IDE (Integrated Developing Environment).
- Refer to the below link to download & install Anaconda Navigator.

Link: <https://www.youtube.com/watch?v=5mDYijMfSzs>

Activity 2: Installation of Python Packages

- To write a code for video streaming, we need to install OpenCV (open computer vision) library. Follow the below steps to install it.
 - Open Anaconda Navigator as administrator.
 - Type “pip install requests” and press enter.

The above steps allows to install the opencv packages.

With this we are done with the completion of milestone 2.

Milestone 2: RapidAPI Account Creation.

Now, the milestone-2 is all about creation of an account in the RapidAPI platform, to generate desired application-oriented API.

Browse the below link to navigate to RapidAPI platform.

Link: <https://rapidapi.com/>

Go through the below link for the complete process.

Link: <https://www.youtube.com/watch?v=ReSCo5unPrQ>

Milestone 3: Subscription of Application Oriented API

In this milestone, you need to subscribe the desired application.

- 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.

Note: Go through the reference link provided in the milestone 2, to subscribe for the desired API.

Milestone 4: Building a Flask Application

In this milestone, you will be developing an html page and integrate the generated API.

Name	Size	Type	Date Modified
static		File Folder	11/5/2020 5:54 PM
1.png	251 KB	png File	11/5/2020 5:50 PM
templates		File Folder	11/5/2020 5:54 PM
debug.log	371 bytes	log File	11/5/2020 1:41 PM
home.html	6 KB	html File	11/5/2020 6:45 PM
sample.json	294 bytes	json File	11/5/2020 4:27 PM
summarizer.html	6 KB	html File	11/5/2020 6:44 PM
summary.html	6 KB	html File	11/5/2020 4:47 PM
1.png	251 KB	png File	11/5/2020 5:50 PM
app1.py	1 KB	py File	11/5/2020 5:12 PM
data_file.csv	184 bytes	csv File	11/5/2020 6:48 PM
sample.json	289 bytes	json File	11/5/2020 5:05 PM
test.py	657 bytes	py File	11/5/2020 4:29 PM

- In order to process proceed with this milestone, arrange all your project files in the below format.
- All the above files will be used to develop a flask application.
- In the static folder we will be storing all the flask background images.

- In the templates, you will storing all the rendering files and html pages.
- In the app1.py files, you will be writing the python script for the flask application to integrate to user interface.
- In the test.py , you will be writing a code to save csv & json files.
- You will storing all the probabilities of the output values in the form of csv (comma separated value i.e. data_file.csv) & will be stored in the form of json.

Activity 1: 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 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 web server is opened in 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': "XXXXXXX", # 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('/summerizer')
def summerizer():
    return render_template('summerizer.html')
```

- Defining the summerizer 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 local host.

```

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

```

Note: Ensure that, you have written the code properly without indentation errors.

Activity 2: Running of flask Application

- Open anaconda prompt from 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 local host URL and open that URL in browser. It does navigate you to the 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)
```

Activity 3: User Interface Should Look Like This.


Automated Topic Tagging
Home
Summarizer

Detect and generate human like topics for the given text.

Topic generation and tagging may be a common task for us, but it is a huge challenge for computers, mainly because computers do not understand languages. How many applications will benefit if computers start to understand text like humans? With this tool, news feeders can easily classify their contents and advertisers can also place ads on the web in a more relevant way, matching the content of web pages to the advertiser's industry.

Get started by choosing an article or text

Generate Topics



Automatic Topic Tagging
Home

Topic generation and tagging may be a common task for us, but it is a huge challenge for computers, mainly because computers do not understand languages. How many applications will benefit if computers start to understand text like humans?

Enter text to generate topics:

Select an option

Generate Topics

Topic Tagging

[Home](#) [Analyse](#)**Detected Topics are..**

Type	Probability
computer science	0.5011
study	0.3002
machine	0.2309
system	0.2309
human	0.2309
art	0.2078