Practice-Webapis

October 6, 2024

[41]:
```
# Name: Mrudu lahari Malayanur
```

1 Swagger Demo: Web APIs

1.1 Overview

The key to making device consumption work are Web API's (Application Program Interfaces accessible over the Internet). Products we use everyday like smartphones, Amazon's Alexa, and gaming consoles all rely on web API's. They seem "smart" and "powerful" but in actuality they're only interfacing with smart and powerful services in the cloud.

API consumption is the new reality of programming; it is why we cover it in this course. Once you undersand how to conusme API's you can write a program to do almost anything and harness the power of the internet to make **your own programs look "smart" and "powerful."**

1.2 CENT IoT Portal

This is a walk-through for how to use some of the Web API's in the iSchool IoT Portal. For each web API we will discover common use cases.

Before you start the lab login to the IoT portal with your netid and copy your APIKey

https://cent.ischool-iot.net

Note you will also need to visit the Api Docs page, here:

https://cent.ischool-iot.net/doc

[12]:
```
# start by importing the modules we will need
import requests
import json
import pandas as pd

APIKEY = "22538d68063b9ec4596c665a"
```

1.3 Azure AI API

The Azure AI API allows you to extract meaning from text. Here, we will explore 3 services:

- Entity Recognition - Identify and extract entities from text.
- Key Phrase extraction - Identify and key phrases from large quantities of text.

- Sentiment - Dervie sentiment / mood from text.

These are 3 common activities found in the discipline known as **text mining**, which is a form of **NLP (Natural Language Processing**

1.4 Entity Recognition

Find `/api/azure/entityrecognition` in the API Docs.

How is this API called? Is it a get or a Post? What is the input? What is the output format?

With that decided, let's extract entities from the following phrase:

`"I would not pay $5 to see that Star Wars movie next week when I am in Buffalo."`

Here is code to call the API:

[13]:
```
text_input = {"text": "I would not pay $5 to see that Star Wars movie next week␣
 ↪when I am in Buffalo."}

url = "https://cent.ischool-iot.net/api/azure/entityrecognition"  #URL
headers = {"X-API-KEY": APIKEY}  # Header input
response = requests.post(url, headers=headers, data=text_input) # Make the␣
 ↪request
response.raise_for_status() # Raise Error if not 200
data = response.json() # deserialize
data
```

[13]:
```
{'kind': 'EntityRecognitionResults',
 'results': {'documents': [{'id': '1',
    'entities': [{'text': '$5',
      'category': 'Quantity',
      'subcategory': 'Currency',
      'offset': 16,
      'length': 2,
      'confidenceScore': 1.0},
     {'text': 'next week',
      'category': 'DateTime',
      'subcategory': 'DateRange',
      'offset': 47,
      'length': 9,
      'confidenceScore': 1.0},
     {'text': 'Buffalo',
      'category': 'Location',
      'subcategory': 'City',
      'offset': 70,
      'length': 7,
      'confidenceScore': 1.0}],
    'warnings': []}],
  'errors': [],
```

```
 'modelVersion': '2023-09-01'}}
```

What happened? It looks like the API extracted 3 entities. These are located at the following key: `data['results']['documents'][0]['entities']`

For this API call, this is where the entities will be. It should be noted that where to look for what you need will depend on the API call and the results.

Let's put these results, which are a list into a dataframe! **Important: Make sure you see the comparision between the DataFrame and the JSON output in the cell above**

[14]:
```
df = pd.json_normalize(data['results']['documents'][0]['entities'])
df
```

[14]:

	text	category	subcategory	offset	length	confidenceScore
0	$5	Quantity	Currency	16	2	1.0
1	next week	DateTime	DateRange	47	9	1.0
2	Buffalo	Location	City	70	7	1.0

Breaking down the request with another example The approach to coding a web API call is always the same:

1. Setup inputs necessary for the API
2. Make request (call web API)
3. Make sure response is ok / not an error
4. Deserialize the response
5. Do something with the outputs

[15]:
```
# read in the email file
with open("email.txt", "r") as f:
    text = f.read()

# INPUT
text_input = {"text": text }

# PROCESS
url = "https://cent.ischool-iot.net/api/azure/entityrecognition"  #URL
headers = {"X-API-KEY": APIKEY }  # Header input
response = requests.post(url, headers=headers, data=text_input) # Make the␣
 ↪request
response.raise_for_status() # Raise Error if not 200
data = response.json() # deserialize

#OUTPUT
df = pd.json_normalize(data['results']['documents'][0]['entities']) # find what␣
 ↪we need in the output

df # Show the output... Warning! Its a lot of data!
```

[15]:
```
                                    text      category  offset  length  \
0                 stephen.marquard@uct.ac.za         Email       5      26
1                   Sat Jan  5 09:14:16 2008      DateTime      32      24
2         postmaster@collab.sakaiproject.org         Email      71      34
3                              mail.umich.edu           URL     130      14
4                               141.211.14.90     IPAddress     146      13
..                                        …           …     …     …
102                            Sakai Collab  Organization    2768      12
103  https://collab.sakaiproject.org/portal)           URL    2782      39
104                       someone@hotmail.com         Email    2920      19
105                  louis@media.berkeley.edu         Email    2946      24
106                  Fri Jan  4 18:10:48 2008      DateTime    2971      24

     confidenceScore subcategory
0               0.80         NaN
1               0.93    DateTime
2               0.80         NaN
3               0.80         NaN
4               0.80         NaN
..               …          …
102             0.98         NaN
103             0.80         NaN
104             0.80         NaN
105             0.80         NaN
106             0.99    DateTime

[107 rows x 6 columns]
```

1.4.1 Curious as to what categories were detected?

Let's check out the categories! You can see entity recognition will attempt to detect a lot of different things from People, to Dates, quantities, and organizations, to more structured data like email addresses, phone numbers, credir cards, and urls.

[16]:
```
df['category'].value_counts()
```

[16]:
```
category
Quantity        38
DateTime        21
URL             21
Email           13
IPAddress        7
Product          4
Skill            1
PersonType       1
Organization     1
Name: count, dtype: int64
```

It finds a lot of entities, but we only want the emails! In a dataframe, so that's just a boolean index filter! Let5's just get the extracted emails.

[17]:
```
emaildf = df[ df['category'] == "Email"]
emaildf
```

[17]:
```
                                                  text category  offset  \
0                             stephen.marquard@uct.ac.za    Email       5
2                      postmaster@collab.sakaiproject.org    Email      71
39   200801051412.m05ECIaH010327@nakamura.uits.iupu…    Email     896
49                         source@collab.sakaiproject.org    Email    1143
56                         source@collab.sakaiproject.org    Email    1365
66                         source@collab.sakaiproject.org    Email    1589
76                         source@collab.sakaiproject.org    Email    1772
83                             stephen.marquard@uct.ac.za    Email    1944
84                         source@collab.sakaiproject.org    Email    1984
85                             stephen.marquard@uct.ac.za    Email    2021
89                             stephen.marquard@uct.ac.za    Email    2250
104                                 someone@hotmail.com    Email    2920
105                               louis@media.berkeley.edu    Email    2946

     length  confidenceScore subcategory
0        26              0.8         NaN
2        34              0.8         NaN
39       51              0.8         NaN
49       30              0.8         NaN
56       30              0.8         NaN
66       30              0.8         NaN
76       30              0.8         NaN
83       26              0.8         NaN
84       30              0.8         NaN
85       26              0.8         NaN
89       26              0.8         NaN
104      19              0.8         NaN
105      24              0.8         NaN
```

List of unique emails To get a list of unique emails, we can now deduplicate and sort the series. We have done this a few times before.

[18]:
```
emails = sorted(list(emaildf['text'].dropna().unique()))
print(emails)
```

```
['200801051412.m05ECIaH010327@nakamura.uits.iupui.edu',
'louis@media.berkeley.edu', 'postmaster@collab.sakaiproject.org',
'someone@hotmail.com', 'source@collab.sakaiproject.org',
'stephen.marquard@uct.ac.za']
```

1.4.2 1.1 You Code create a function

Its useful to wrap the Web API calls in a function. It makes it easier to reuse them and purpose them in the `apply()` method of a dataframe.

Re-write the examples from the cells above in this section as a single function.

Take text as input, output a list of unique emails from the text. The function definition, and an detailed algorithm are provided.

```
def extract_emails(text: str) -> list:
```

```
1) setup the inputs from text
2) call the web api
3) raise if not successful
4) deserialize
5) create a dataframe from list of named entities
6) filter data frame to only emails
7) sort and deduplicate the emails
8) return list of emails
```

In the cell below 1.1 some test code has been provided for you. Again the goal is to write the function to pass the test.

[19]:
```
# TODO Write function defintion code here
# INPUT

def extract_emails(text: str) -> list:
    input_text = {"text": text } #1) setup the inputs from text
    # PROCESS
    my_url = "https://cent.ischool-iot.net/api/azure/entityrecognition"␣
 ↪#creating link
    my_headers = {"X-API-KEY": APIKEY }  # Header input
    my_response = requests.post(url, headers=headers, data=input_text) # Make␣
 ↪the request
    my_response.raise_for_status() #3) raise if not successful
    my_data = my_response.json() # 4 deserialize

    #OUTPUT
    my_df = pd.json_normalize(my_data['results']['documents'][0]['entities'])␣
 ↪#5 create a dataframe from list of named entities
    email_df = my_df[ my_df['category'] == "Email"] #6) filter data frame to␣
 ↪only emails
    my_emails = sorted(list(email_df['text'].dropna().unique())) #7) sort and␣
 ↪deduplicate the emails
    return my_emails #8) return list of emails
```

Test code for your function in 1.1 Run this code to test the function you wrote in 1.1

[23]:
```
## Test!
text = '''As of this year, my primary email address is test@syr.edu but I
    also use testing@gmail.com and tester@aol.com from time to time,
    but test@syr.edu is the main one.'''
expect = ['test@syr.edu', 'tester@aol.com', 'testing@gmail.com']
actual = extract_emails(text)
print("EXPECT:", expect, "ACTUAL:", actual)
assert expect == actual
```

```
EXPECT: ['test@syr.edu', 'tester@aol.com', 'testing@gmail.com'] ACTUAL:
['test@syr.edu', 'tester@aol.com', 'testing@gmail.com']
```

1.5 Key Phrase Extraction

Find `/api/azure/keyphrasextration` in the Api Docs. Review how to call this API.

Key Phrase extraction extracts the subjects from the text. This can be used to determine what someone it talking about.

Let's try it out: Here's are three sample reviews of a restaraunt. What are these is this reviewers talking about?

```
'''
review1 = "I don't think I will ever order the eggs again. They were runny Yuk!"
review2 = "Went there last Wednesday and it was busy, which is good to see. The pancakes and eg
review3 = "Not sure who is running the place but the eggs benedict were not that great. No flav
'''
```

Let's call the API with the 1st review.

[24]:
```
review1 = "I don't think I will ever order the scrambled eggs again. They were␣
 ↪runny. It looked like snot!"
review2 = "Went there last Wednesday and it was busy, which is good to see. The␣
 ↪pancakes and eggs were spot on! I enjoyed my meal and would recommend a␣
 ↪visit."
review3 = "Not sure who is running the place but the eggs benedict were not␣
 ↪that great. No flavor. At least my toast wasn't burnt."

text_input = {"text": review1}

# PROCESS
url = "https://cent.ischool-iot.net/api/azure/keyphrasextraction"
headers = {"X-API-KEY": APIKEY}
response = requests.post(url, headers=headers, data=text_input)
response.raise_for_status()
data = response.json()

#OUTPUT
data
```

[24]:
```
{'kind': 'KeyPhraseExtractionResults',
 'results': {'documents': [{'id': '1',
    'keyPhrases': ['scrambled', 'eggs', 'snot'],
    'warnings': []}],
  'errors': [],
  'modelVersion': '2022-10-01'}}
```

Extracting the phrases For this API, the output is under `data['results']['documents'][0]['keyPhrases']`

Let's extract them:

[25]:
```
key_phrases = data['results']['documents'][0]['keyPhrases']
print(key_phrases)
```

```
['scrambled', 'eggs', 'snot']
```

1.5.1 Rewriting Key Phrase Extraction as a function

Once again, let's wrap this API call in a user-defined function. This time the function will be written for you and you must write the test.

`def extract_keyphrases(text: str) -> list:`

[26]:
```
# Function written for you
def extract_keyphrases(text: str) -> list:
    #INPUT
    text_input = {"text": text}

    # PROCESS
    url = "https://cent.ischool-iot.net/api/azure/keyphrasextraction"
    headers = {"X-API-KEY": APIKEY}
    response = requests.post(url, headers=headers, data=text_input)
    response.raise_for_status()
    data = response.json()

    #OUTPUT
    phrases = data['results']['documents'][0]['keyPhrases']
    return phrases
```

1.5.2 1.2 You Code testing The `extract_keyphrases()` function

Write code similar to the test code in 1.1 but this time you should test the `extract_keyphrases()` function. Just write a single test.

Here's the process:

```
1) Make up some text phrase. Make sure it has a subject that can be extracted. examples:
    "The Xbox is the best video gaming console."
    "The wise consumer uses their credit cards sparingly"
```

```
    You are STRONGLY encouraged to use your own phrase here.
2) Call your function with your example as input and observe the output. Did it work?
3) Re-write as test code:
    a) input_text = your text phrase
    b) expected = output from 2)
    c) actual = code from 2)
    d) print and assert expected == actual
```

[31]:
```
# TODO Write one test here
input_text = 'The Xbox is the best video gaming console.'
expected = ['best video gaming console', 'The Xbox']
actual_output=extract_keyphrases(input_text)
print("EXPECT:", expected, "ACTUAL:", actual_output)
assert expected == actual_output
```

```
EXPECT: ['best video gaming console', 'The Xbox'] ACTUAL: ['best video gaming
console', 'The Xbox']
```

1.6 Sentiment Analysis

Find `/api/azure/sentiment` in the Api Docs. Review how to call this API

Sentiment determines the mood of the text. It is positive or negative, for example?

We saw sentiment before, but this time it will be much more accurate!

Let's try it out: Again the three sample reviews of a restaraunt. Are these reviews "positive" or "negative"?

```
'''
review1 = "I don't think I will ever order the eggs again. They were runny Yuk!"
review2 = "Went there last Wednesday. It was crowded and the pancakes and eggs were spot on! I
review3 = "Not sure who is running the place but the eggs benedict were not that great. No fla
'''
```

Let's call the API with the 2nd review.

[32]:
```
text_input = {"text": review2}

# PROCESS
url = "https://cent.ischool-iot.net/api/azure/sentiment"
headers = {"X-API-KEY": APIKEY}
response = requests.post(url, headers=headers, data=text_input)
response.raise_for_status()
data = response.json()

#OUTPUT
data
```

[32]:
```
{'kind': 'SentimentAnalysisResults',
 'results': {'documents': [{'id': '1',
    'sentiment': 'positive',
    'confidenceScores': {'positive': 0.98, 'neutral': 0.02, 'negative': 0.0},
    'sentences': [{'sentiment': 'positive',
      'confidenceScores': {'positive': 0.98, 'neutral': 0.02, 'negative': 0.0},
      'offset': 0,
      'length': 65,
      'text': 'Went there last Wednesday and it was busy, which is good to see.
'},
     {'sentiment': 'positive',
      'confidenceScores': {'positive': 0.97,
       'neutral': 0.02,
       'negative': 0.01},
      'offset': 65,
      'length': 36,
      'text': 'The pancakes and eggs were spot on! '},
     {'sentiment': 'positive',
      'confidenceScores': {'positive': 1.0, 'neutral': 0.0, 'negative': 0.0},
      'offset': 101,
      'length': 46,
      'text': 'I enjoyed my meal and would recommend a visit.'}],
    'warnings': []}],
  'errors': [],
  'modelVersion': 'latest'}}
```

Sentiment output is complex! You can see the input text was broken up into individual sentences. Each sentence was analyzed for sentiment.

The overall sentiment is under the key: `data['results']['documents'][0]['sentiment']`

The scores are under the key: `data['results']['documents'][0]['confidenceScores']`

Let's extract:

[33]:
```
sentiment = data['results']['documents'][0]['sentiment']
scores = data['results']['documents'][0]['confidenceScores']
print(sentiment, scores)
```

```
positive {'positive': 0.98, 'neutral': 0.02, 'negative': 0.0}
```

1.6.1 1.3 Writing the `extract_sentiment()` function

Once again, let's write a function to wrap the web API.

```
def extract_sentiment(text: str) -> tuple[str, str]:
```

No algorithm is provided this time, **by now you should realize the structure of these functions is always the same.** What differs is the actual URL and how exactly to extract what you want from the JSON output, and the code above demonstrates how that is done.

In this case, we want to return a two values:

`return sentiment, scores`

[37]:
```
# TODO write function code here
def extract_sentiment(text: str) -> tuple[str, str]:
    text_input = {"text": text}
    # PROCESS
    url = "https://cent.ischool-iot.net/api/azure/sentiment"
    headers = {"X-API-KEY": APIKEY}
    response = requests.post(url, headers=headers, data=text_input)
    response.raise_for_status() #checking status of API response
    data = response.json()
    sentiment = data['results']['documents'][0]['sentiment']
    scores = data['results']['documents'][0]['confidenceScores']
    return sentiment, scores
```

1.6.2 Testing the `extract_sentiment()` function

Run this code to ensure your function was written properly!

[38]:
```
# Test
input_text = 'I like scotch. Scotchy-scotch, scotch. Tastes so good.'
expect_sentiment = 'positive'
expect_scores = {'positive': 0.99, 'neutral': 0.01, 'negative': 0.0}
sentiment, scores = extract_sentiment(input_text)
print("EXPECT:", expect_sentiment, expect_scores, "ACTUAL:", sentiment, scores)
assert expect_sentiment == sentiment
assert expect_scores == scores
```

```
EXPECT: positive {'positive': 0.99, 'neutral': 0.01, 'negative': 0.0} ACTUAL:
positive {'positive': 0.99, 'neutral': 0.01, 'negative': 0.0}
```

1.7 Now that you know about the fundamentals of text AI, what can you do with it?

There are a variety of uses for text AI. Let me provide an example of how this could be used if you owned a chain of restaurants.

1. You can take customer Yelp and Google reviews and run sentiment analysis to determine how customers feel about it.
2. Extract key phrases to understand what the are talking about. For example, they like the food but dislike the service. The like the pizza.
3. Use named entity recognition get indications of pricing. "$15 is too high for a hamburger", or location / date "I visited from Buffalo last thursday and the service was slow."
4. Use key phrase extraction to determine what they are talking about? Pancakes, breakfast sandwiches, eggs, pizza, food, service, location, etc…

What I've outlined is a form of **opinion mining**, which is a very specific application of **text mining**

1.8 Last Example: Sentiment and Key Phrase Extraction:

"Two great tastes that taste great together."

When we combine sentiment with key phrase extraction we not only know what they are talking about but how they feel about it. This is a powerful form of analysis that is essential to opinion mining.

Let's write a program that takes input text then outputs the sentiment and keyphrases. Call the functions you wrote in 1.2 and 1.3 to complete this activity. Make sure your output follows the example here. The entire program is 4 lines of code.

```
EXAMPLE RUN  #1
Enter Text: Their pizza is the best in town
The text mentions ['pizza', 'town'] in a positive way.

EXAMPLE RUN  #2
Enter Text: Goats milk is gross.
The text mentions ['Goats milk'] in a negative way.

EXAMPLE RUN  #3
Enter Text: The new york yankees as the most despised baseball program in the major leagues.
The text mentions ['new york yankees', 'baseball program', 'major leagues'] in a negative way.
```

1.8.1 1.4 You Code

Write the program here. No need to use an interact a simple `input()` and `print()` will suffice.

[40]:
```
# TODO write code here
text = input("enter the input text")
phrases=extract_keyphrases(text)
sentiment_result=extract_sentiment(text)

print(f"The given input has fowllowing phrases: {phrases} and emotions␣
 ↪mentioned are: {sentiment_result}.")
```

```
enter the input text Scripting in python is Nice
```

```
The given input has fowllowing phrases: ['Scripting', 'python'] and emotions
mentioned are: ('positive', {'positive': 0.93, 'neutral': 0.07, 'negative':
0.0}).
```

[ ]:

[ ]: