Data Glacier

Data Science Intern at Data Glacier

Week 5: Cloud API Development

Name: Itohowo Charles

Batch Code: LISUM12

Date: 25 August 2022

Submitted to: Data Glacier

Table of Contents:

1. Introduction

In this project, we are going to deploying machine learning model (SVM) using the Flask Framework. As a demonstration, our model help to predict the spam and ham comment of YouTube.

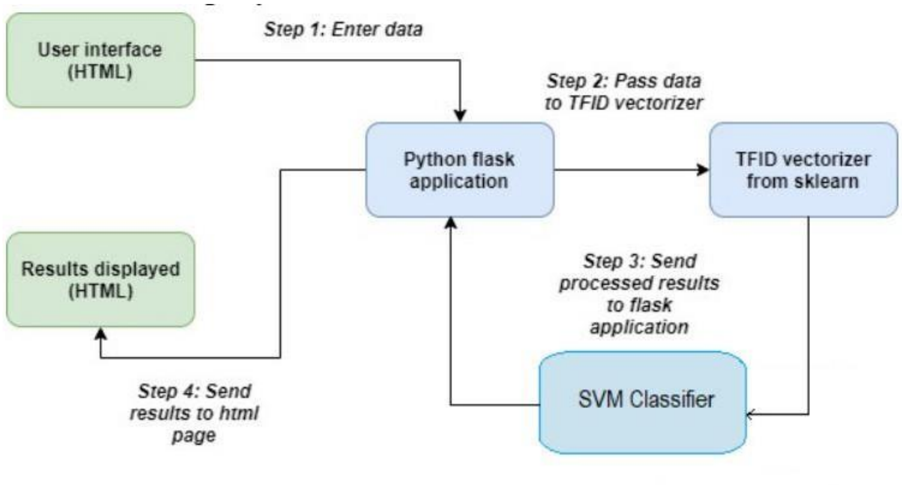


Figure 1.1: Application Workflow

we will focus on both: building a machine learning model for YouTube Comments SD, then create an API for the model, using Flask, the Python micro-framework for building web applications. This API allows us to utilize predictive capabilities through HTTP requests.

2. Data Information

The samples were extracted from the comments section of five videos that were among the 10 most viewed on YouTube during the collection period. The table below lists the datasets, the YouTube video ID, the number of samples in each class and the total number of samples per dataset.

Table 2.1: Dataset Information

Dataset	YouTube ID	Spam	Ham	Total
Psy	9bZkp7q19f0	175	175	350
KatyPerry	CevxZvSJLk8	175	175	350
LMFAO	KQ6zr6kCPj8	236	202	438
Eminem	uelHwf8o7_U	245	203	448
Shakira	pRpeEdMmmQ0	174	196	370

2.1 Attribute Information

The collection is composed of one CSV file per dataset, where each line has the following attributes:

Table 2.2: Attribute Information

Attributes	Example (1 instance)
COMMENT_ID	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU
AUTHOR	Julius NM
DATE	2013-11-07 T 06:20:48
CONTENT	Huh, anyway check out this YouTube channel: kobyoshi02
Class	1 (Spam)

3. Building a Model

3.1 Import Required Libraries and Dataset

In this part, we import libraires and dataset which contain the information of five most commented video.

```
In [1]: # import Libaries & Packages
        import numpy as np                # Import Numpy for data statistical analysis
        import pandas as pd               # Import Pandas for data manipulation using dataframes
        import seaborn as sns             # Statistical data visualization
        import matplotlib.pyplot as plt   # Import matplotlib for data visualisation

In [2]: # Import Youtube Ham or Spam dataset taken from UCI
        df1 = pd.read_csv("dataset/Youtube01-Psy.csv")          # Psy youtube channel most viewed video comments dataset
        df2 = pd.read_csv("dataset/Youtube02-KatyPerry.csv")    # KatyPerry youtube channel most viewed video comments dataset
        df3 = pd.read_csv("dataset/Youtube03-LMFAO.csv")        # Psy LMFAO channel most viewed video comments dataset
        df4 = pd.read_csv("dataset/Youtube04-Eminem.csv")       # Eminem youtube channel most viewed video comments dataset
        df5 = pd.read_csv("dataset/Youtube05-Shakira.csv")      # Shakira youtube channel most viewed video comments dataset

In [3]: # Merge all the datasset into single file
        frames = [df1,df2,df3,df4,df5]                          # make a list of all file
        df_merged = pd.concat(frames)                           # concatenate the all the file into single
        keys = ["Psy","KatyPerry","LMFAO","Eminem","Shakira"]   # Merging with Keys
        df_with_keys = pd.concat(frames,keys=keys)              # concatenate data with keys
        dataset=df_with_keys

In [4]: # Infomation about dataset
        print(dataset.size)               # size of dataset
        print(dataset.shape)              # shape of datadet
        print(dataset.keys())             # attributes of dataset

        9780
        (1956, 5)
        Index(['COMMENT_ID', 'AUTHOR', 'DATE', 'CONTENT', 'CLASS'], dtype='object')
```

3.2 Data Preprocessing

The dataset used here is split into 80% for the training set and the remaining 20% for the test set. We fed our dataset into a Term Frequency-Inverse document frequency (TF-IDF) vectorizer which transforms words into numerical features (numpy arrays) for training and testing

```
# working with text content
dataset = dataset[["CONTENT" , "CLASS"]]          # context = comments of viewers & Class = ham or Spam
```

```
# Predictor and Target attribute
dataset_X = dataset['CONTENT']                    # predictor attribute
dataset_y = dataset['CLASS']                      # target attribute
```

```
# Feature Extraction from Text using  TF-IDF model
from sklearn.feature_extraction.text import TfidfVectorizer   # import TF-IDF model from scikit Learn
```

```
# Extract Feature With TF-IDF model
corpus = dataset_X                              # declare the variable
cv = TfidfVectorizer()                          # initialize the TF-IDF  model
X = cv.fit_transform(corpus).toarray()          # fit the corpus data into BOW model
```

```
# Split the dataset into Train and Test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, dataset_y, test_size=0.2, random_state=0)
```

```
# shape of predictor attrbute after Extract Features
X.shape
```

```
(1956, 4454)
```

3.3 Build Model

After data preprocessing, we implement machine learning model to classify the YouTube spam comments. For this purpose, we implement Support Vector Machine (SVM) using scikit-learn. After importing and initialize SVM model we fit into training dataset.

```
# import the model from sklean
from sklearn.svm import SVC          # import the Support Vector Machine Classifier model
```

```
 # initialize the model
classifier = SVC(kernel = 'linear', random_state= 0)
```

```
# fit the dataset into our classifier model for training
classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

3.4 Save the Model

After that we save our model using pickle

```
# import pickle library
import pickle             # pickle used for serializing and de-serializing a Python object structure
```

```
# save the model
Support_Vector_Machine = open("model.pkl","wb")          # open the file for writing
pickle.dump(classifier,Support_Vector_Machine)           # dumps an object to a file object
Support_Vector_Machine.close()                           # here we close the fileObject
```

4. Turning Model into Web Application

We develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of spam or ham(not spam).

First, we create a folder for this project called YouTube Spam Filtering, this is the directory tree inside the folder. We will explain each file.

Table 3.1: Application Folder File Directory

```
app.py
templates/
        home.html
        result.html
static/
    style.css

model/
     model.pkl

dataset/

     Youtube01-Psy.csv

     Youtube02-KatyPerry.csv

     Youtube03-LMFAO.csv

     Youtube04-Eminem.csv

     Youtube05-Shakira.csv
```

The sub-directory templates are the directory in which Flask will look for static HTML files for rendering in the web browser, in our case, we have two HTML files: *home.html* and *result.html*.

4.1 App.py

The `app.py` file contains the main code that will be executed by the Python interpreter to run the Flask web application, it included the ML code for classifying SD.

```python
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict',methods=['POST'])
def predict():
    df1 = pd.read_csv("dataset/Youtube01-Psy.csv")              # Psy youtube channel most viewed video comments dataset
    df2 = pd.read_csv("dataset/Youtube02-KatyPerry.csv")        # KatyPerry youtube channel most viewed video comments dataset
    df3 = pd.read_csv("dataset/Youtube03-LMFAO.csv")            # Psy LMFAO channel most viewed video comments dataset
    df4 = pd.read_csv("dataset/Youtube04-Eminem.csv")           # Eminem youtube channel most viewed video comments dataset
    df5 = pd.read_csv("dataset/Youtube05-Shakira.csv")          # Shakira youtube channel most viewed video comments dataset

    # Merge all the datasset into single file
    frames = [df1,df2,df3,df4,df5]                          # make a list of all file
    df_merged = pd.concat(frames)                           # concatenate the all the file into single
    keys = ["Psy","KatyPerry","LMFAO","Eminem","Shakira"]   # Merging with Keys
    df_with_keys = pd.concat(frames,keys=keys)              # concatenate data with keys
    dataset=df_with_keys

    # working with text content
    dataset = dataset[["CONTENT" , "CLASS"]]             # context = comments of viewers & Class = ham or Spam

    # Predictor and Target attribute
    dataset_X = dataset['CONTENT']                       # predictor attribute
    dataset_y = dataset['CLASS']                         # target attribute

    # Extract Feature With TF-IDF model
    corpus = dataset_X                               # declare the variable
    cv = TfidfVectorizer()                           # initialize the TF-IDF  model
    X = cv.fit_transform(corpus).toarray()           # fit the corpus data into BOW model


    # import pickle file of my model
    model = open("model/model.pkl","rb")
    clf = pickle.load(model)

    if request.method == 'POST':
        comment = request.form['comment']
        data = [comment]
        vect = cv.transform(data).toarray()
        my_prediction = clf.predict(vect)
        return render_template('result.html',prediction = my_prediction)


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

Figure 3.1: App.py

- We ran our application as a single module; thus we initialized a new Flask instance with the argument *name*_____to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
- Next, we used the route decorator *(@app.route('/'))* to specify the URL that should trigger the execution of the home function.
- Our *home* function simply rendered the *home.html* HTML file, which is located in the *templates* folder.
- Inside the *predict* function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.

- we used the *POST* method to transport the form data to the server in the message body. Finally, by setting the *debug=True* argument inside the app.run method, we further activated Flask's debugger.
- Lastly, we used the *run* function to only run the application on the server when this script is directly executed by the Python interpreter, which we ensured using the *if* statement with_*name*_== '_*main*_'.

4.2 Home.html

The following are the contents of the `home.html` file that will render a text form where a user can enter a message.

```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
    <!-- <link rel="stylesheet" type="text/css" href="../static/css/styles.css"> -->
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>

    <header>
        <div class="container">

        <h2>Youtube Comments Spam Detection</h2>

    </div>
    </header>

    <div class="ml-container">

        <form action="{{ url_for('predict')}}" method="POST">
        <p>Enter Your Comment Here</p>
        <!-- <input type="text" name="comment"/> -->
        <textarea name="comment" rows="4" cols="50"></textarea>
        <br/>

        <input type="submit" class="btn-info" value="predict">

    </form>

    </div>

</body>
</html>
```

Figure 3.2: Home.html

4.3 Style.css

In the header section of *home.html*, we loaded *styles.css* file. CSS is to determine how the look and feel of HTML documents. *styles.css* has to be saved in a sub-directory called *static*, which is the default directory where Flask looks for static files such as CSS.

4.4 Result.html

we create a `result.html` file that will be rendered via the *render_template('result.html', prediction=my_prediction)* line return inside the *predict* function, which we defined in the *app.py* script to display the text that a user-submitted via the text field.

From *result.html* we can see that some code using syntax not normally found in HTML files: *{% if prediction ==1%},{% elif prediction == 0%},{% endif %}*This is Jinja syntax, and it is used to access the prediction returned from our HTTP request within the HTML file.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/styles.css') }}">
</head>
<body>

    <header>
        <div class="container">

        <h2>YouTube Comments Spam Detection</h2>

    </div>
    </header>
    <p style="color:black;font-size:20;text-align: center;"><b>Results for Comment</b></p>
    <div class="results">



    {% if prediction == 1%}
    <h2 style="color:red;">Spam</h2>
    {% elif prediction == 0%}
    <h2 style="color:green;">Not a Spam (It is a Ham)</h2>
    {% endif %}

    </div>

</body>
</html>
```

Figure 3.3: Result.html

4.5 Running Procedure

Once we have done all of the above, we can start running the API by either double click `app.py`, or executing the command from the Terminal

Now we could open a web browser and navigate to http://127.0.0.1:5000/, we should see a simple website with the content like so

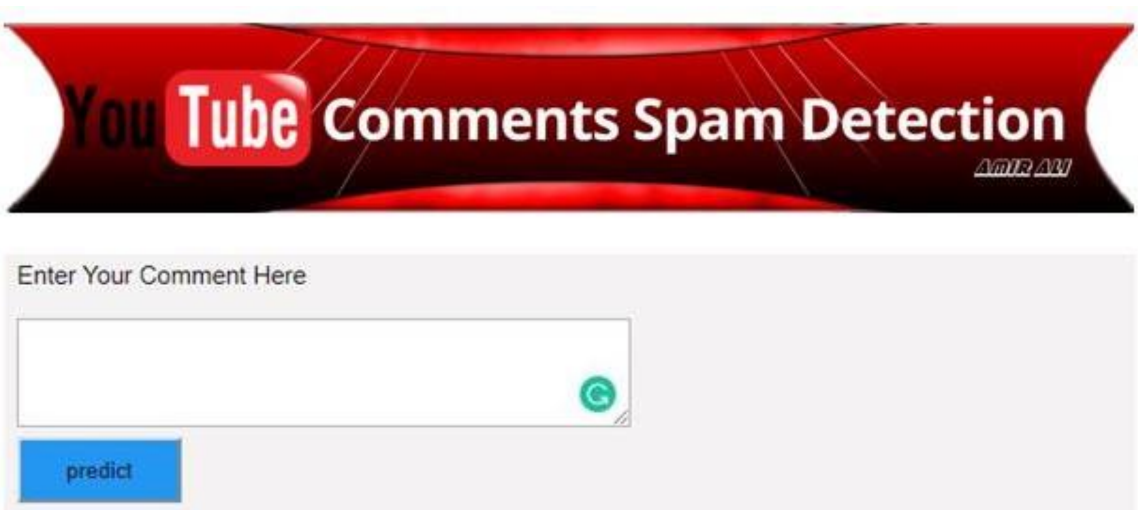


Figure 3.5: Spam Detection Website Page

Now we enter input in the comments form

Figure 3.6: Input In The Comments Form

After entering the input click the predict button now, we can the result of our input.

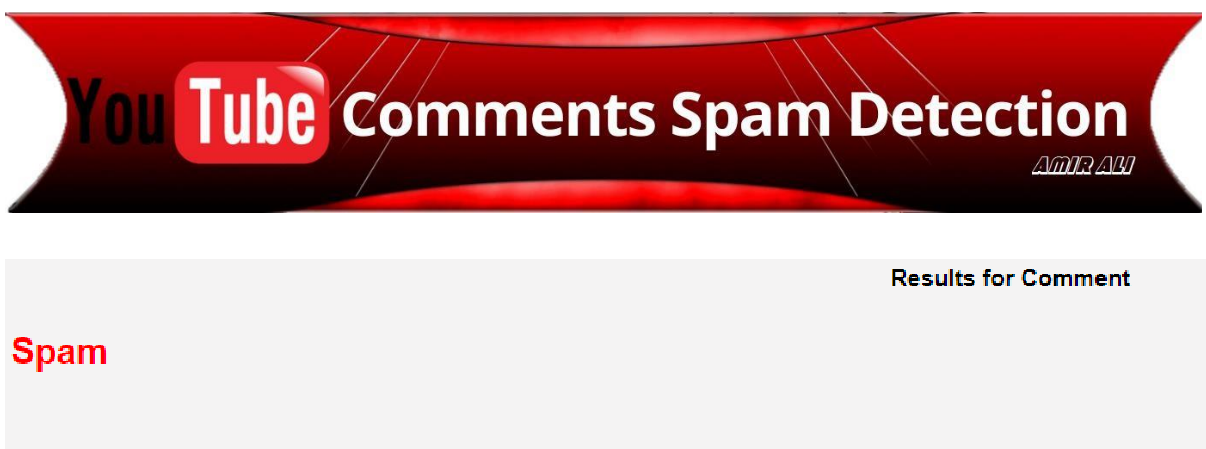


Figure 3.7: Result of Given Input

5. Model deployment using Heroku

We're ready to start our Heroku deployment now that our model has been trained, the machine learning pipeline has been set up, and the application has been tested locally. There are a few ways to upload the application source code onto Heroku. The easiest way is to link a GitHub repository to your Heroku account.

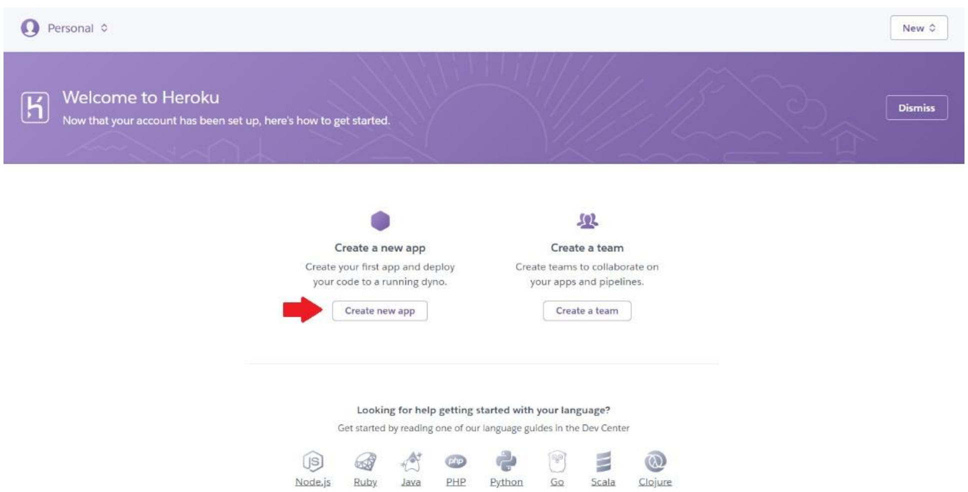
Requirement.txt

It is a text file containing the python packages required to execute the application.

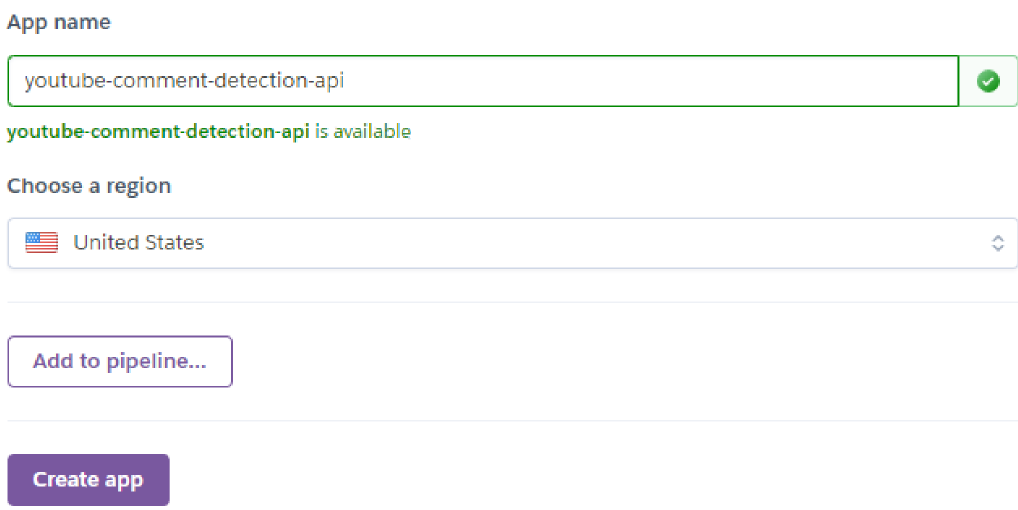
5.1 Steps for Model Deployment Using Heroku

Once we uploaded files to the GitHub repository, we are now ready to start deployment on Heroku. Follow the steps below:

1. After sign up on **heroku.com** then click on **Create new app.**



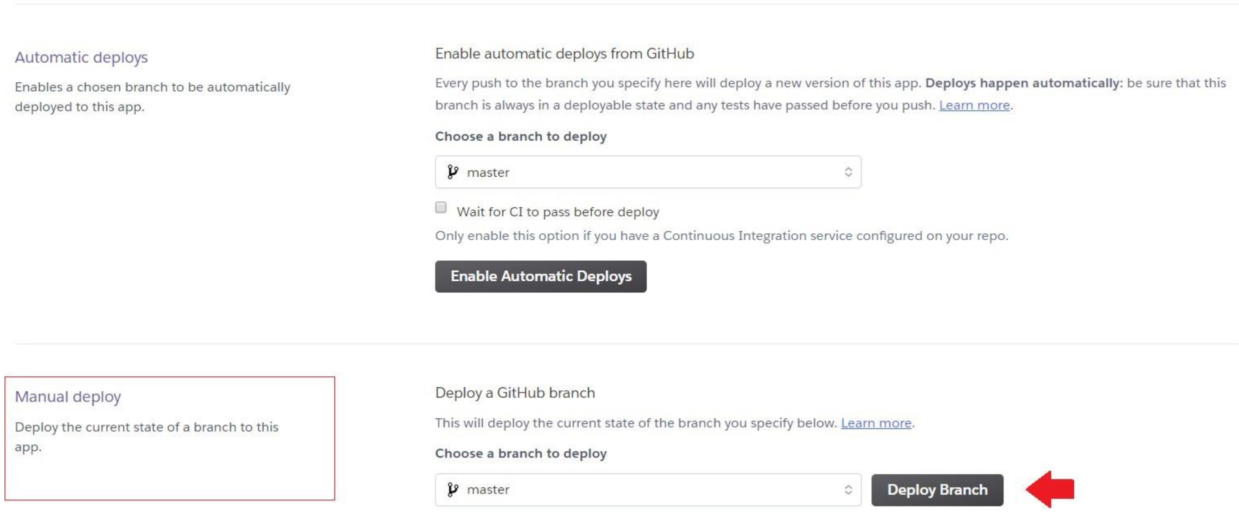
2. Enter **App name and region**



3. Connect to GitHub repository where code is I uploaded.

After that I choose the repository where I upload the code.

4. Deploy branch



5. After waiting 5 to 15 minutes our application is Ready