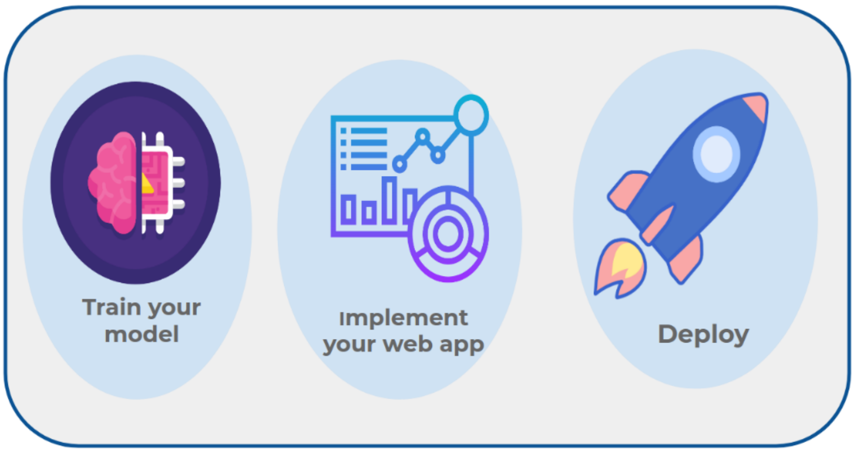
Lab Artificial intelligence

Project: Machine learning model deployment for web App



Established by: Houcem Ben Makhlouf

Mentored by: Ph.D. Erick Elejalde

Professor: Prof. Dr. techn. Nejdl Wolfgang

Table of Contents

Table of figures:

1. Problem definition

The deployment of machine learning models is the process of making models available in production where web applications, enterprise software and APIs can consume the trained model by providing new data points and generating predictions. And our main task in this project consists of integrating a tweets classification model into an existing production environment which can take in an input and return an output that can be used in making practical decisions to help the user figure out if the tweets are coming from a robot. Our project come with the importance of giving people more insight about the tweeted data.

2. Requirements and installations

After getting the repository of the project, we should create a python environment so that we can install all packages we need for this project which are going to be installed as dependencies in our app, more precisely in package.json. To create our virtual environment, we can execute the command line after specifying the path where we want to create our environment.

```
$ python3 -m venv /path/to/new/virtual/environment
```

The packages we need for this project are described in the file requirements.txt. before my work began, we had these requirements.

```
pymongo==3.11.3
scholarmetrics==0.2.1
scikit-learn==0.21.3
```

After my work this list was updated to be.

```
joblib==1.0.1
numpy==1.19.5
pbr==5.6.0
pymongo==3.11.3
scholarmetrics==0.2.1
scikit-learn==0.21.3
scipy==1.6.3
```

Figure 1: requirements to install

To install these requirements, we need to pass the command line

```
$ pip install -r requirements.txt
```

To launch the project, we should:

- Activate the environment via: $ venv\scripts\activate
- Launch the server via: $ python .\main.py
- Go under tests via: $ cd tests

- Begin sending request via: $ python .\send_requests.py

3. Project overview

a. General insight

In our project there are mainly two big parts:

- The model part: where the algorithm of our model is defined with its different features and parameters. In this project we will not be working on this part. The model is already trained over a set of data and we can use it directly. The Algorithm of this part is found under the file classify_account.py
- The deployment part: This part is main purpose of this project and it consists of building a REST API using Flask. This python framework will allow us to create a deployment component that will interact with our machine learning model and the user. It will take that input data given as an external file from the user, process it and then pass it to model to get the predicted data as an output which will be visualized to the user. This part is developed in the file “main.py”.

The figure 2 shows the general circulation of the information in our project.

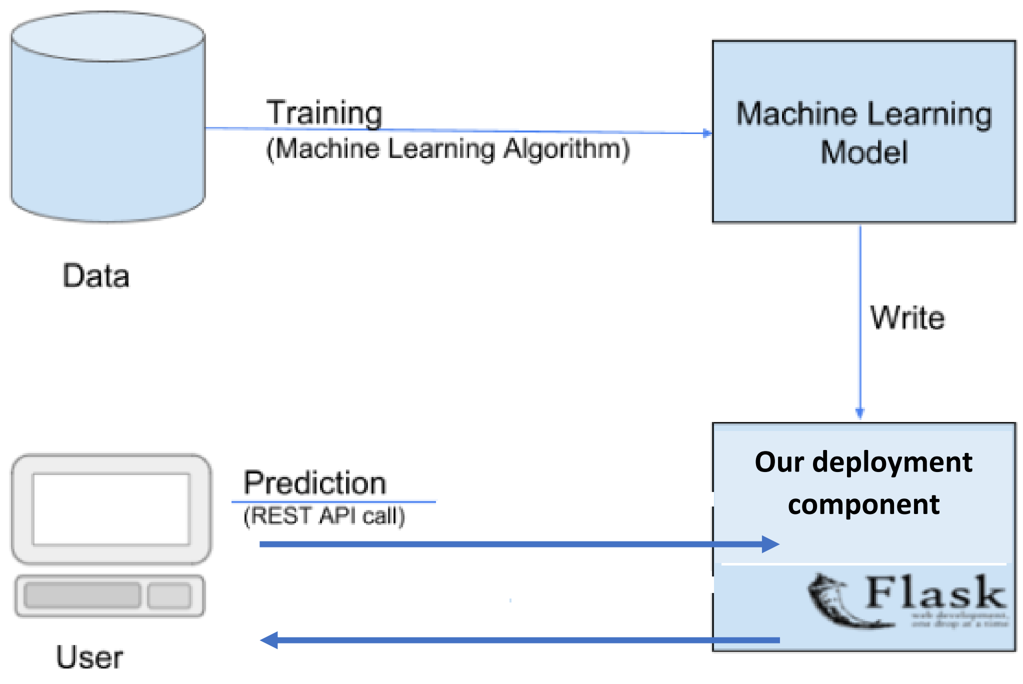


Figure 2:General insight

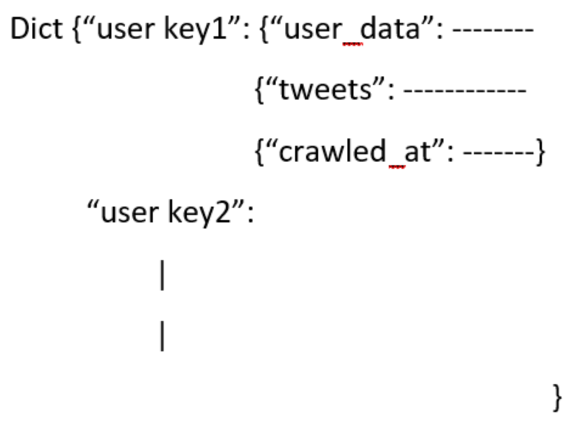
b. Input format

Data type and formats are one of the essential parts when working on a machine learning algorithm. The Data should be clean as possible to minimize error; missing values should be filled out if there is any. Data should be reduced for proper data handling. The data type is very important to precise so that we don't get any errors of compatibilities with the model.

In our case the data set was provided in pickle file. It is a dictionary with the keys ['94c383089f0dd9993020276bd01113ecb5935ad860bfa61e6079e7d548577f76', '57e2082d64baa89d01d3c03b4beaf9e6708ab209378f7dcd849cb2c9034ee362', 'BarackObama', 'Mietinotizie', 'MadeleinOpperma']

Under each dictionary we find another dictionary with the keys: ['user_data', 'tweets', 'crawled_at']

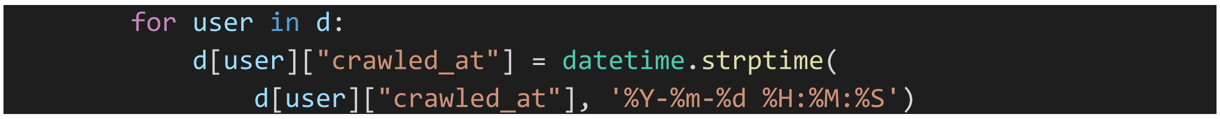
So, the general structure of our data is:



```
Dict {“user key1”: {“user_data”: --------
                    {“tweets”: ------------
                    {“crawled_at”: -------}
     “user key2”:
           |
           |
                                         }
```

Figure 3: General structure of the data set

To make the data set compatible with what we get in a real case we have converted the data set into a json version. But if we transmit directly this data set to the classify algorithm, we will face problems of serialization such “Object of type datetime is not JSON serializable”. An example to overcome such was to create a datetime object from the given string given in the json file. Under the file “main.py” we find all the modification we did to overcome this error. A code example is as follow:

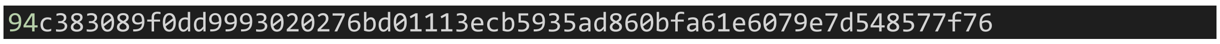


```
        for user in d:
            d[user]["crawled_at"] = datetime.strptime(
                d[user]["crawled_at"], '%Y-%m-%d %H:%M:%S')
```

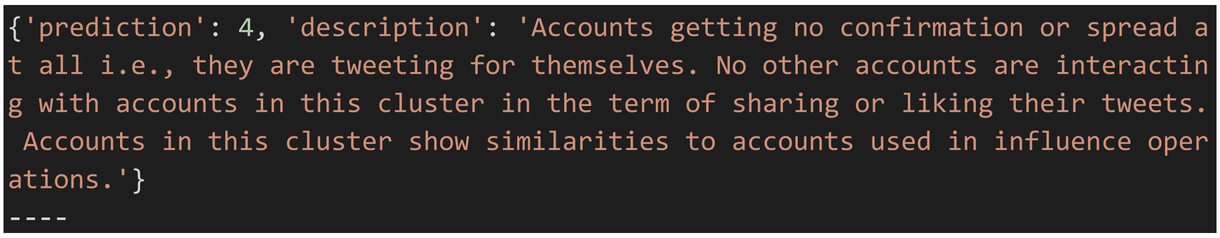
Figure 4: Resolving the sterilization problem

c. Output

After passing the data, the model is going to classify our users into one of 6 classes. The figure below shows an example of classification for the user with the key: “94c383089f0dd9993020276bd01113ecb5935ad860bfa61e6079e7d548577f76”. For each user we get predicted class and the description of that class.



```
94c383089f0dd9993020276bd01113ecb5935ad860bfa61e6079e7d548577f76
```



```
{'prediction': 4, 'description': 'Accounts getting no confirmation or spread a
t all i.e., they are tweeting for themselves. No other accounts are interactin
g with accounts in this cluster in the term of sharing or liking their tweets.
 Accounts in this cluster show similarities to accounts used in influence oper
ations.'}
----
```

Figure 5: Final result overview

d. Testing

This provided data set “example_data.p” is a reduced version of the entire data to make it easier testing the algorithm. After the testing part, the data set was multiplied into 1000 and 2000 users to test further the performance. To simplify the work, we took the first instance of data set with the following user key: '94c383089f0dd9993020276bd01113ecb5935ad860bfa61e6079e7d548577f76' and we applied the multiplication process. The table below shows the time needed to get the predicted results taking into consideration the data size:

Data set size (number of users)	5	1000	2000
Time needed to get results (Seconds)	0.04590916	7.29847478	16.06111311

Table 1: Time needed for different data set sizes

To further test the asynchronous behaviour another file “send_requests.py” is added where we can pass a predefined number of requests in the same time. These requests will be treated one by one. The table below shows the time needed to get all the predicted results when passing a predefined number of requests using the data set that was provided “example_data.json” and the data set with 2000 users named “example_users_extended.json”.

Number of requests	5	10	15
Original data	5.44167089	10.82522678	17.76469802
Extended data	125.926930	279.5530014	386.4041688

Table 2: Time needed for different data sets and different numbers of requests

Thinking about how the user will pass the command two ways were cited:

- Using Curl which is used in command lines or scripts to transfer data. Here below an example of command is presented. To make this command line work I used “BufferedReader” which reads data from the object and kept in an internal buffer.

```
$ curl -i -X POST -H 'Content-Type: application/json' -
F 'data=@/example_data.p' http://localhost:5000/process
```

- Using the terminal of the project when compiling the “send_requests.py” file. This method is more reliable in our case because more details and tests can be done in the future. Also, we can modify certain parameters like in our case, we wanted to test the time needed if we pass the request many times like shown in table 2.

4. Perspectives and conclusions

The project dealt with the subject of deployment of a machine learning model which classifies tweets based on their credibility. I have worked on data types, especially how to pre-process the data and make it ready for the processing phase. I understood better the asynchronous behaviour and I knew better how a flask Rest API works. As a perspective, I thought of building a user interface which I integrated a reduced version of it in the “main.py” component because I think it is always more user-friendly.