Location Analytics Project: New-York Crimes Prediction

Alaeddine DHAOUI & Mohamed CHIBOUB

Recognizing the patterns of criminal activity in a place is essential to help authorities prevent it or to alert and advise citizens during their travels. Law enforcement agencies can work effectively and respond faster if they have a better understanding of crime patterns in different geographies of a city. The objective of this project is to set up a web solution to fight and prevent crime. We were able to achieve a very high accuracy using an ANN model and predict which type of crime which will occur, depending on the position of a potential victim, and his/her personal information.

Additional Key Words and Phrases: Location Analytics, Open Layers, Artificial Neural Networks, Geolocation

1 INTRODUCTION

Crime rates in New York city have seen massive change since the 1980s, after officials starting adapting new ways to minimize the crimes that were spiking in rates at the time. So the aim of this paper is to propose a way of detecting what kind of crime that can occur to a potential victim based on multiple input information. For this matter, we will be using an artificial intelligence model using Artificial Neural Networks. Various methods have been used to generate a very high accuracy of the model used. Once the prediction model was tuned, we were able to create an intuitive interface, that allows a victim to select his/her position on the city of New York, enter his/her personal information and then predict the crime that can occur to him/her, in that location, at any particular time.

2 DATA SET

The data used to train the artifical intelligence model was brought from the NYC Open Data website. It is free public data published by New York City agencies and other partners. In our case, we used a database of more than 6 million different crime records dating back to 1915.

3 DATA PREPARATION

3.1 Data Cleaning:

Data cleaning is the process of adding missing data and correcting, repairing, or removing incorrect or irrelevant data from a data set.

- Missing Data: To deal with missing data in our data set we opted for 2 steps as follow:
  - Step 1: First we will work with only features that can be inserted by a user of our application. Indeed, there are many features in our data-set that are filled based on the target value so we will not use them in our prediction model (such as: PD_CD, Jurisdiction_code, . . .). So, in this step we will only work with these features:
    - ∗ **PATROL_BORO**: The name of the patrol borough in which the incident occurred
    - ∗ **BORO_NM**: The name of the borough in which the incident occurred
    - ∗ **PREM_TYP_DESC**: Specific description of premises; grocery store, residence, street, etc
    - ∗ **VIC_AGE_GROUP**: The age of the victim
    - ∗ **VIC_SEX**: The sex of the victim
    - ∗ **VIC_RACE**: The race of the victim
    - ∗ **ADDR_PCT_CD**: The address of the precinct where the crime has occurred
    - ∗ **DateTim**e: A timestamp value that refers to the exact date and time of the occurence of the crime
    - ∗ **Longitude**: Longitude from the place where the crime has occurred
    - ∗ **Latitude**: Latitude from the place where the crime has occurred
  - Step 2: Remove rows that shows at least one missing value in any of our features.
- Noisy Data: Data cleaning also includes fixing "noisy" data. This is data that includes unnecessary data points, irrelevant data, and data that's more difficult to group together.
  Some features present noisy data. Let's take as example the feature that refers to the age of the victim. This one, presents values higher than 100 and sometimes an observation has as victim age value higher than 900. That's why we used a technique called Binning that groups similar observations values (of one feature or more) into smaller groups hence we can eliminate groups that are below some logic thresholds.

3.2 Data Transformation

With data cleaning, we have already begun to modify our data, but data transformation will begin the process of turning the data into the proper format(s) you'll need for analysis and other downstream processes.

- **Feature selection**: After selecting some features in the previous phase (Data cleaning phase) we will use also some statistical techniques to choose the best combination of features that are the most likely to give the best results. So that, we used the "Correlation Matrix" to perform this task.
- **Normalization**: Normalization scales our data into a regularized range [0 , 1] so that they can be compared more accurately (especially when using binary encoding or/ and one-hot encoding with the categorical features).
- **Data Encoding**: Encoding is the transformation of categorical features to binary (or numerical) counterparts.
  In our case, we opted for the binary encoding as it does not create an ordering problem between values of the same variable. In addition, the advantage of using binary encoding Instead of one-hot encoding is that the last one generates from a feature that have for example 128 classes 128 new binary features and the first one generates only 8 binary features. Hence, this allows us to reduce the probability of getting a problem of "curse of dimensionality".

3.3 Data Splitting

Data splitting is the act of partitioning available data into two parts (at least). One portion is used to develop a predictive model (training data) and the other to evaluate the model's performance (test data). We used the function sklearn.model_selection.train_test_split().

4 PREDICTION MODEL

Model Selection: The Choice of the model depends on many variables of the problem:

- The quantity of the data
- The quality of the data
- the dimensionality of the space
- etc

We decided to use an Artificial Neural Network model, as its hidden layers, and connections give the most accurate results, given the number of classes and the complexity of the data.

5 RESULTS

After training the ANN model, we were able to get the following accuracy result, reaching an accuracy value of 81%.

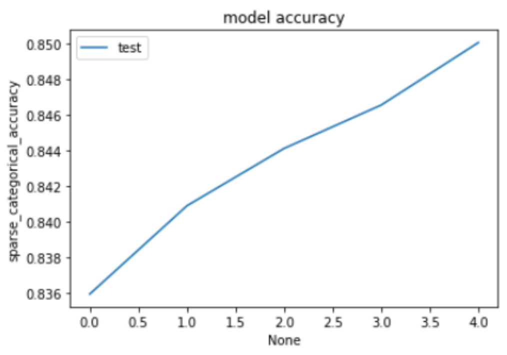
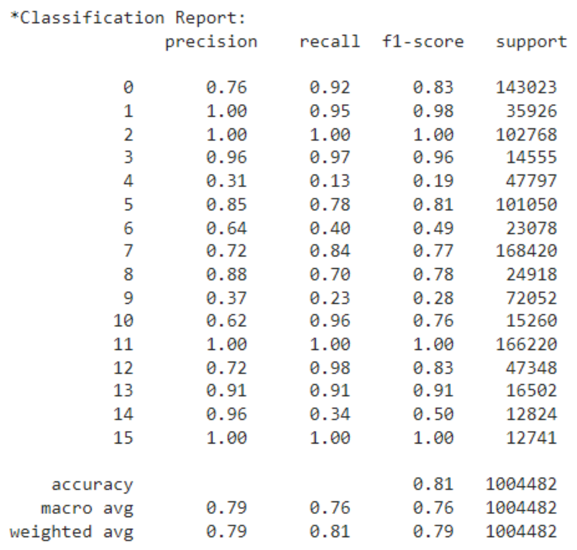


Fig. 1. Model Accuracy

As for the classification report, we got very promising results, which are the following:



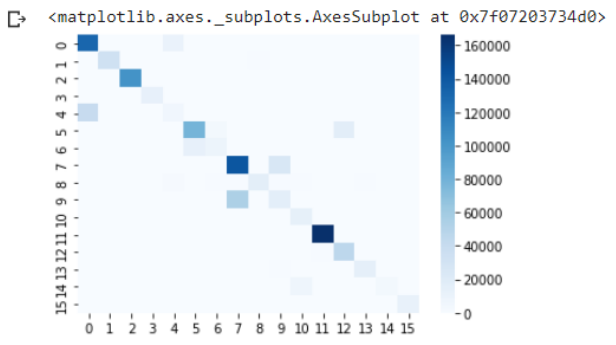
```
*Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.92      0.83    143023
           1       1.00      0.95      0.98     35926
           2       1.00      1.00      1.00    102768
           3       0.96      0.97      0.96     14555
           4       0.31      0.13      0.19     47797
           5       0.85      0.78      0.81    101050
           6       0.64      0.40      0.49     23078
           7       0.72      0.84      0.77    168420
           8       0.88      0.70      0.78     24918
           9       0.37      0.23      0.28     72052
          10       0.62      0.96      0.76     15260
          11       1.00      1.00      1.00    166220
          12       0.72      0.98      0.83     47348
          13       0.91      0.91      0.91     16502
          14       0.96      0.34      0.50     12824
          15       1.00      1.00      1.00     12741

    accuracy                           0.81   1004482
   macro avg       0.79      0.76      0.76   1004482
weighted avg       0.79      0.81      0.79   1004482
```

Fig. 2. Classification Report

And finally, the following confusion matrix was generated to gather an idea about the ratio of false-positive and false-negative results.



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f07203734d0>
```

Fig. 3. Confusion Matrix

6 WEB APPLICATION

In order to properly use our prediction model, we have to incorporate it within a web application. In this case, we used FastAPI and Vanilla Javascript in order to make it.

6.1 Frontend

The frontend was made using vanilla javascript and mainly used OpenLayers to use geolocation capabilities, such as displaying the map of New York, adding layers on top of it, etc.
In our case, the frontend consisted of two parts: the first part was mainly the map, where the user can click anywhere on the map. If the map lies within the surface of New York, then that point will be marked as the user location.

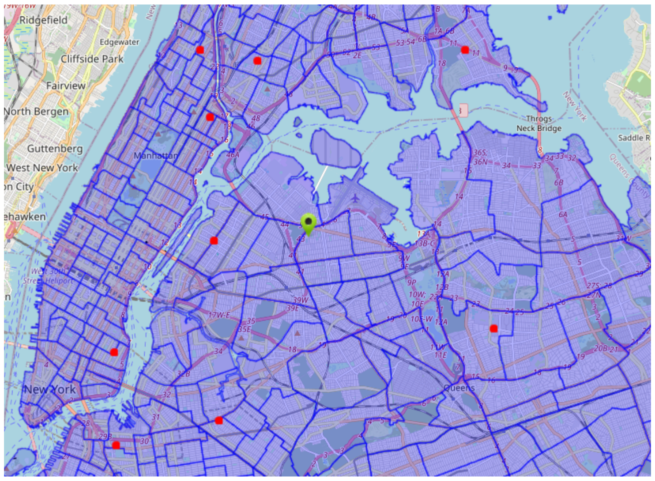


Fig. 4. Selecting User Location using Open Layers Map

Users can activate different layers. The first layer is the different boroughs of New york (marked with the blue color) and the second layers is the different locations of main police stations in New York.

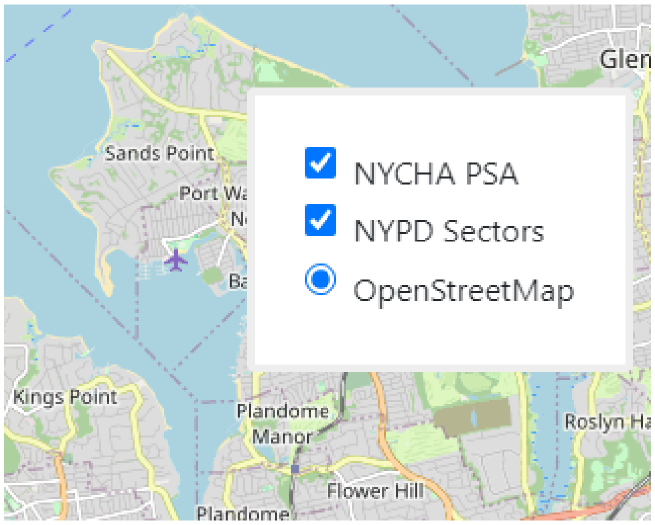
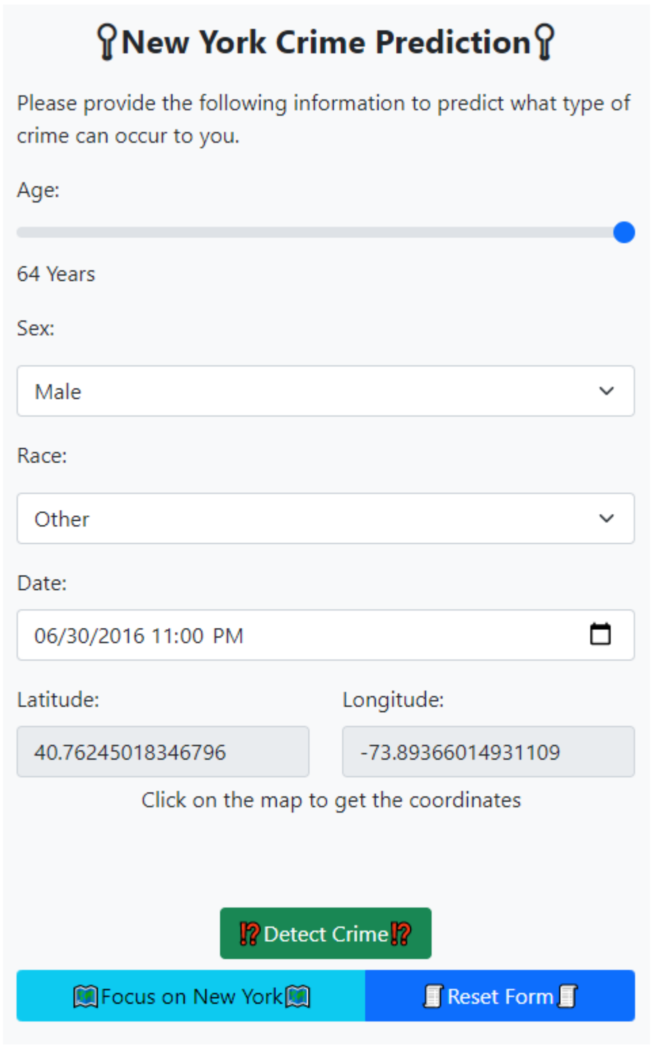


Fig. 5. Layer selection inside the map

Right after clicking on the map, we also save current location information, such as the borough code of the selected area, the precinct, etc. This information will be sent later when we want to predict the crime.
As for the second main part of the frontend, we have the prediction form, which must be filled in order to be able to send the prediction request to the backend. Several input fields are required, such as the age of the victim, its race and sex, the date of the supposed crime, as well as the coordinates of the crime. But note that the latter is automatically filled, as long as the user chooses a location on the map.


New York Crime Prediction

Please provide the following information to predict what type of crime can occur to you.

Age:

64 Years

Sex:

Male

Race:

Other

Date:

06/30/2016 11:00 PM

Latitude: 40.76245018346796

Longitude: -73.89366014931109

Click on the map to get the coordinates

Fig. 6. Filling out the crime prediction form

After filling out all the information, the "Detect Crime" button will become clickable. So once it is clicked, a request will be sent to the backend with all the information of the victim. The result is then returned and we show it via a toast popup.

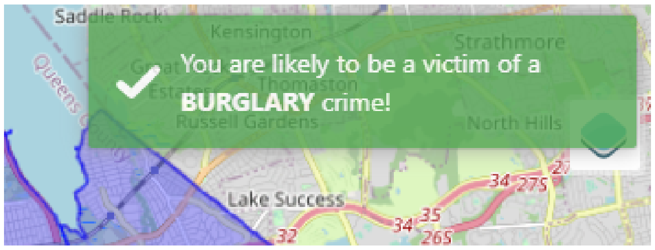
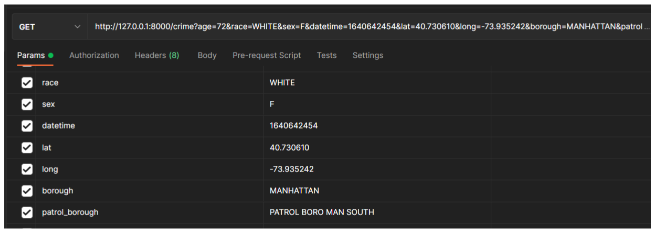

You are likely to be a victim of a **BURGLARY** crime!

Fig. 7. Showing the Result Toast

6.2 Backend

As for the backend, we used FastAPI to create an CPI that will receive all the information from the frontend, call the prediction model using that info and finally get and return the predicted crime. For this, we loaded all the dictionaries necessary to convert the data that the model needs. We also enabled CORS. The API works correctly and the response time is very minimal. Testing using Postman was done in order to make sure the the API works correctly.

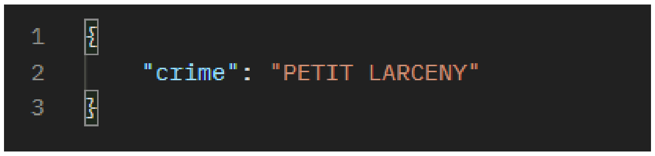


```
http://127.0.0.1:8000/crime?age=72&race=WHITE&sex=F&datetime=1640642454&lat=40.730610&long=-73.935242&borough=MANHATTAN&patrol ...
```

✓ race	WHITE
✓ sex	F
✓ datetime	1640642454
✓ lat	40.730610
✓ long	-73.935242
✓ borough	MANHATTAN
✓ patrol_borough	PATROL BORO MAN SOUTH

Fig. 8. Example of a request made using Postman

The result is sent as JSON and is parsed correctly in the frontend. It only contains one attribute which is the crime that is committed.



```
{
    "crime": "PETIT LARCENY"
}
```

Fig. 9. The JSON Response from the prediction API

As an optimization, we added a FastAPI caching plugin, which uses redis in-memory database, in order to cache requests, this makes the prediction much faster for repetitive requests.