Lorenzo De Nisi s276545 *09/12/2019*

Report Data Science Lab – Laboratory 9

The purpose of the laboratory 9 is to analyze a dataset of **AirBnB listings** and build a **regression model** capable of predicting the price per night for new listings.
The dataset is divided into

- Nearly 40000 labeled listings used for training (NYC_AirBnB/development.csv)
- Nearly 10000 unlabeled listings used for final evaluation (NYC_AirBnB/evaluation.csv)

Firstly we have to discuss whether or not every given information should be used.
These are assumptions coming from the context knowledge, not from effective correlation analysis.

• id: a unique identifier of the listing	**IDENTIFIER**
• name	**NOT RELEVANT**
• host_id: a unique identifier of the host	**NOT RELEVANT**
• host_name	**NOT RELEVANT**
• neighbourhood_group: neighborhood location in the city	**VERY RELEVANT**
• neighbourhood: name of the neighborhood	**VERY RELEVANT**
• latitude: coordinate expressed as floating point number	**RELEVANT**
• longitude: coordinate expressed as floating point number	**RELEVANT**
• room_type	**VERY RELEVANT**
• price: price per night expressed in dollars	**LABEL**
• minimum_nights: minimum nights requested by the host	**RELEVANT**
• number_of_reviews	**RELEVANT**
• last_review: date of the last review expressed as YYYY-MM-DD	**NOT RELEVANT**
• reviews_per_month: average number of reviews per month	**NOT RELEVANT**
• calculated_host_listings_count: amount of listing of the host	**RELEVANT**
• availability_365: number of days when the listing is available for booking	**RELEVANT**

The given data is then integrated with **criminality info** taken from data.cityofnewyork.us for the year 2018.
In particular the informations used for the regression were the **crimes/citizen ratio*** for every neighborhood group and the **number of crimes committed** in 2018 in a radius of 500m from the listing coordinates.

*Population data is taken from citypopulation.de

The first step is reading the files:

```python
def load_data(path):
    data = {}
    with codecs.open(path, "r", 'utf-8') as file:
        header = True
        headers = []
        for row in csv.reader(file):
            if header:
                for field in row:
                    data[field] = []
                headers = row.copy()
                header = False
                continue

            skip = False
            for n, row_value in enumerate(row):  # check
                if headers[n] == "price" and row_value == "0":
                    skip = True

            if skip: continue

            for n, row_value in enumerate(row):
                if headers[n] == "reviews_per_month" and row_value == '':
                    row_value = "0"

                data[headers[n]].append(row_value)

    return data
```

Here I filter the rows with price **0** and I fill the missing reviews per month with 0 values.

Then the categorical data needs to be converted with **one-hot encoding**.
In this case we have **neighborhood** and **neighborhood group**.
(**room type** is also categorical but we can label-encode it, assuming an order: *shared room, private room, Entire home/apt*).
(A shared room is more likely to be cheaper than an entire home)

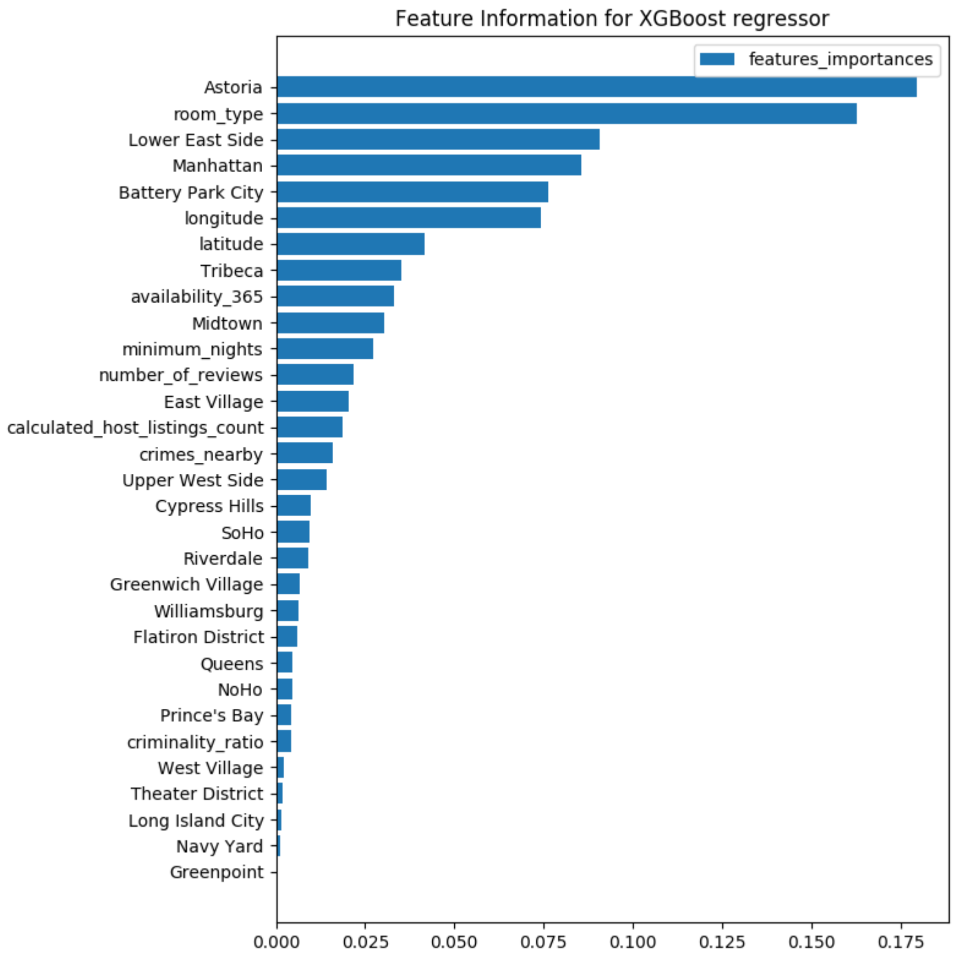
```python
def one_hot_encode_data(data, uniques=None):
    if uniques is None:
        uniques = {}
        i = 0
        for value in data:
            if value not in uniques.keys():
                uniques[value] = i
                i += 1

    one_hots = []
    for values in data:
        code = [0] * len(uniques)
        code[uniques[values]] = 1
        one_hots.append(code)

    return one_hots, uniques
```

Numerical values are converted into **float** for uniformity.

The training step is done with an holdout with 0.8 ratio while the used regressor is **XGBoost**.

```python
def regression(X_train, X_test, y_train, y_test):

    reg = xgb.XGBRegressor()
    reg.fit(X_train, y_train)
    print_score(reg, X_test, y_test)

    return reg
```

From the trained regressor we can extract features informations.
This can be useful for a possible grouping of categorical attributes:



We can see that as predicted, the **room type** is highly significant for the regression, followed by **coordinates**, **availability** and **minimum nights**.

The most significant neighborhood group is **Manhattan** (that usually hosts the higher price listings)

From the bar graph we can observe that only some neighborhoods are effectively useful for regression, so for dimensionality reduction we can group the others in a single category.
In this case the relevant neighborhoods (feature importance > 0) were:

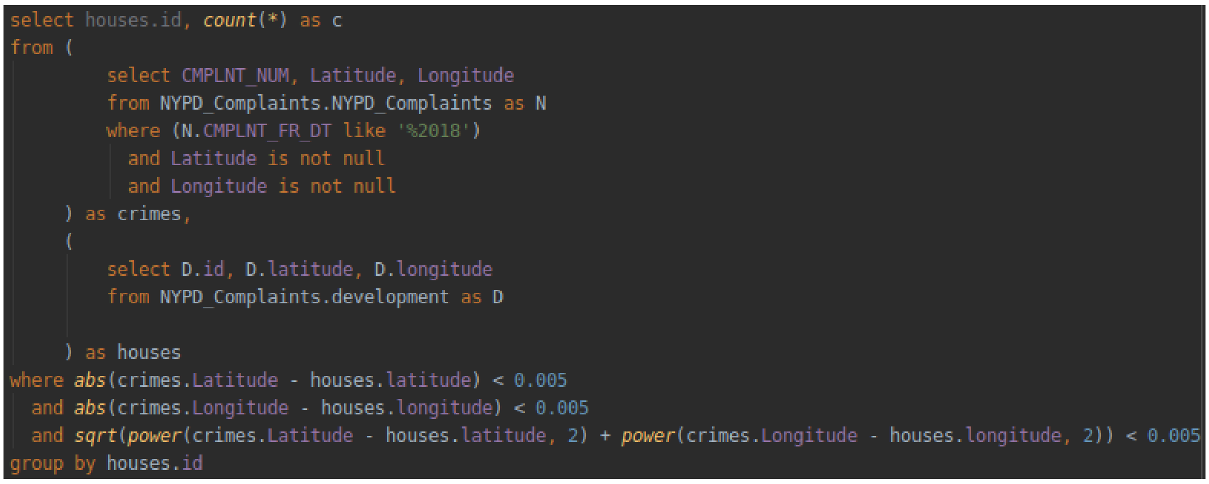
```
relevant_neighbourhoods = ['Greenwich Village', 'Williamsburg', 'Greenpoint', 'East Village', 'Upper
West Side', 'Lower East Side', 'Midtown', 'Astoria', 'Cypress Hills', 'Tribeca', 'SoHo', 'Flatiron
District', 'Theater District', 'NoHo', 'Navy Yard', 'Battery Park City', "Prince's Bay", 'Riverdale',
'West Village', 'Long Island City']
```

```
def custom_one_hot(data, relevants):
    dic = {}
    i = 0
    for r in relevants:
        dic[r] = i
        i += 1

    one_hots = []
    for value in data:
        code = [0] * len(dic)
        if value in dic.keys():
            code[dic[value]] = 1
        one_hots.append(code)

    return one_hots
```

In this case, neighbourhoods labeled as '**others'** will correspond to a all-zero one-hot code.

Criminality Analysis

The criminality data comes from the data.cityofnewyork.us website.
There are a whole bunch of informations that, in this case, are not relevant, and we only need 2018 data, so, for extracting the aggregate data that we need, the csvs are loaded into a *MySql* database.
Then by querying jointly on **AirBnB listing** data and **crime records** we can extract the number of crimes in a radius of approximately **500m**. (considering 1 degree as 100km with an heavy approximation)



```
select houses.id, count(*) as c
from (
         select CMPLNT_NUM, Latitude, Longitude
         from NYPD_Complaints.NYPD_Complaints as N
         where (N.CMPLNT_FR_DT like '%2018')
           and Latitude is not null
           and Longitude is not null
     ) as crimes,
     (
         select D.id, D.latitude, D.longitude
         from NYPD_Complaints.development as D

     ) as houses
where abs(crimes.Latitude - houses.latitude) < 0.005
  and abs(crimes.Longitude - houses.longitude) < 0.005
  and sqrt(power(crimes.Latitude - houses.latitude, 2) + power(crimes.Longitude - houses.longitude, 2)) < 0.005
group by houses.id
```

The extracted **csv** tells us the number of crimes related to the listing ids.
We can see the distribution of number of crimes with a **boxplot**.

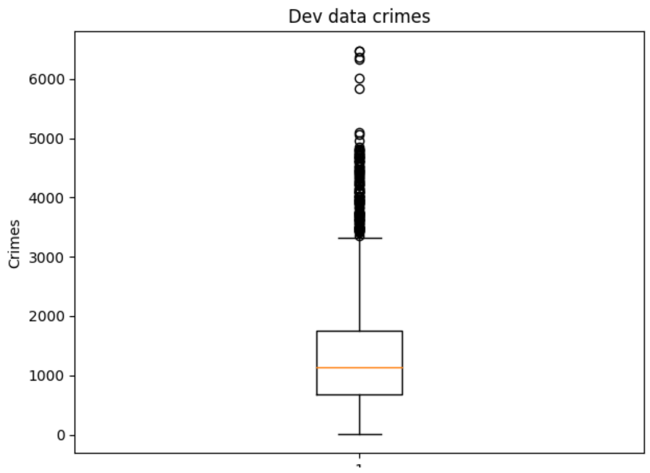


Figure 1: 500m radius

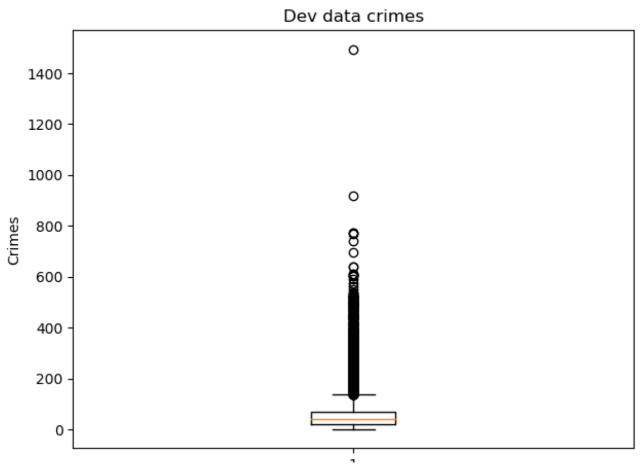
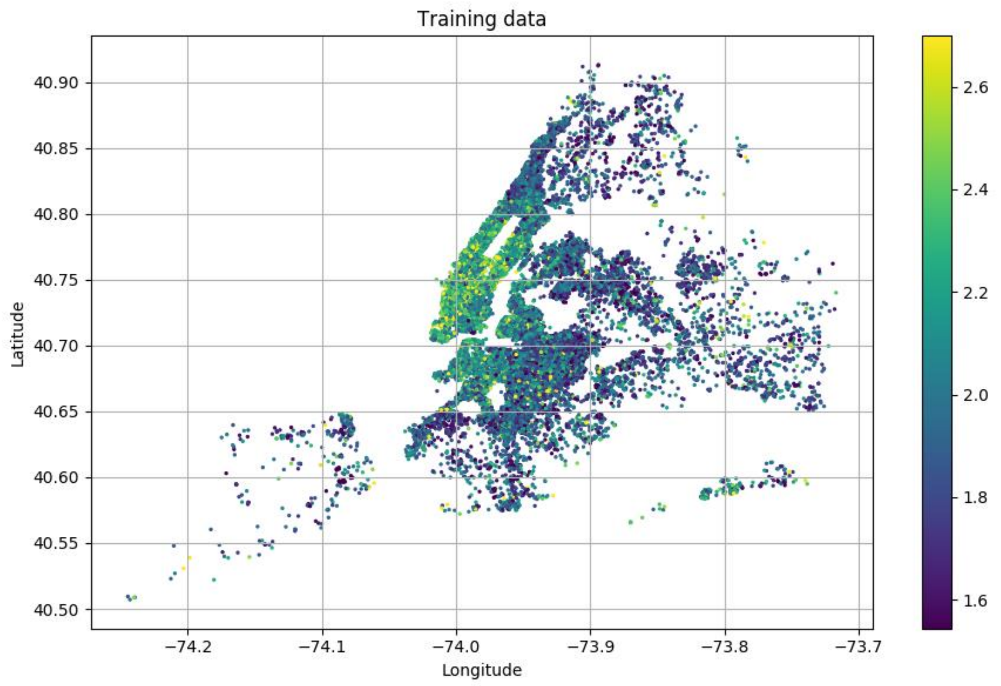


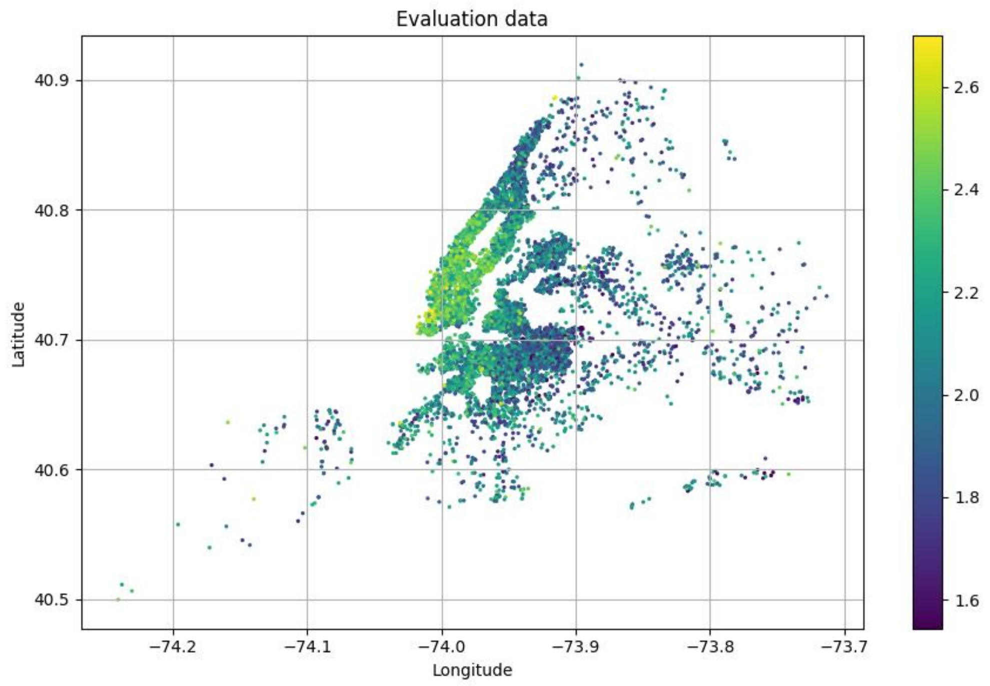
Figure 2: 100m radius

As we can see from the boxplot comparison, using a wider radius gives us data distributed over a wider range.
A too small radius is too much subjected to casuality while a too big radius may consider other neighborhoods and a lot of listings would share the same crimes, causing lower variance.

Result

The regression results can be displayed by heatmapping development prices with respect to predicted prices on evaluation data.
Here the color represents the **log10 of the listings price**, in that way the color range is better covered.
Moreover, the tails of the price distributions are “cut”.
To do so, the color range is calculated between the *2.5th and the 97.5th percentiles*.





We can see that the general trend on the evaluation data is respected while it has some problems with the more eterogeneous areas (Brooklyn for instance, where there are some scattered expensive listings)

Appendix

Here I report the other pieces of code that I consider relevant:

Function used to plot the heatmaps of prices:

```
def plot_prediction(train_lon, train_lat, y_train, test_lon, test_lat, y_pred):
    fig, ax = plt.subplots(1, 2, figsize=(18, 6))
    ax[0].grid()
    ax[0].set_title("Training data")
    ax[0].set_xlabel("Longitude")
    ax[0].set_ylabel("Latitude")
    p1 = ax[0].scatter(train_lat, train_lon, c=y_train, s=2, vmin=np.percentile(y_train, 2.5),
                       vmax=np.percentile(y_train, 97.5))
    plt.colorbar(p1, ax=ax[0])

    ax[1].grid()
    ax[1].set_title("Evaluation data")
    ax[1].set_xlabel("Longitude")
    ax[1].set_ylabel("Latitude")
    p2 = ax[1].scatter(test_lat, test_lon, c=y_pred, s=2, vmin=np.percentile(y_train, 2.5),
                       vmax=np.percentile(y_train, 97.5))
    plt.colorbar(p2, ax=ax[1])
    plt.show()
```

Function used to plot feature importance bar graph.

```
def plot_features_importance(reg, features_dict):
    fi = reg.feature_importances_
    sorted = np.argsort(fi)

    fi = [fi[i] for i in sorted if fi[i] > 0]
    sorted = sorted[-len(fi):]

    x = np.arange(0, len(sorted))
    width = 0.8

    fig = plt.figure(figsize=(8, 8))
    plt.barh(x, fi, width, label='features_importances')
    plt.yticks(x, [features_dict[i] for i in sorted])
    plt.legend(loc='best')
    plt.title("Feature Information for XGBoost regressor")
    plt.show()
```

features_dict is a dictionary with the index of a feature as a key, and the name of the feature as the value.

Label encoder used for **room_type**

```
def label_encoding(data, dict=None):
    if dict is None:
        dict = {}
        i = 0
        for value in data:
            if value not in dict.keys():
                dict[value] = i
                i += 1

    encoded = []
    for value in data:
        encoded.append(dict[value])

    return encoded
```

used with this encoding dictionary:

```
encoding_dict = {"Shared room": 0, "Private room": 1, "Entire home/apt": 2}
```