Prediction of Crime Scores in Mexico City

Mario Horacio Garrido Czacki

January 2021

1 Introduction

Crime is an unfortuante byproduct of urban life, and Mexico City has a particularly high crime incidence. While locals are good at estimating a zone's risk level due to experience or anecdote, those unfamiliar with the area in most likelihood would lack a point of reference as to what measures to take in order to keep themselves safe.

Even though Mexico City has an open database of all reported crimes, quickly estimating a zone's danger is a complex task that cannot be feasibly solved because of the large volume and discrete format of the data. Even when filtering by time and geography, sometimes crimes simply don't get reported. This is why in this paper I will descibe a model of crime level estimation that may interpolate to any point within the city with a reasonable amount of confidence.

2 The Approach

Measuring crime as a single number is relatively complicated. There are temporal and spatial considerations in order to be able to properly asses the danger of a single point in space. For example, a very recent crime might reflect little danger if the trend as a whole reflects that the area is relatively safe. While simplistic, I decided to use the following definition of a crime score:

$$S_{abs}(\bar{p}) = |\{q \in Q | dist(\bar{p}, q_{loc}) < \epsilon, timedelta(q_{time}) < \delta\}|$$

Where $\bar{p}$ is a location $S_{abs}(\bar{p})$ is the crime score predicted. $q \in Q$ is a crime in the dataset with a location q_{loc} and time of occurrence q_{time}. Because of the geographic nature of the dataset, $dist(\bar{a}, \bar{b})$ should usually be a measure such as the Haversine distance. However due to the small area of interest, it will be approximated as the euclidean distance between any two points $(\bar{a}, \bar{b})$. $timedelta(t)$ are the days elapsed since a certain time t relative to the query time.

In other words, I will measure the crime score of a point $\bar{p}$ as the number of crimes occured within a certain radius ϵ in the last δ days. For scale purposes the score used in the rest of the paper, $S(\bar{p})$ will be the natural logarithm of $S_{abs}(\bar{p})$ rescaled into a $[0, 1]$ range. In the final model, the values of $\epsilon = 200m$ and $\delta = 365$ were used.

While this is a possible query for any point in the database, interpolation of scores may result difficult due to the relatively low report rate in any given area and size of the dataset. Because of this, I have decided to predict $\hat{S(\bar{p})}$ using the ten nearest crimes in the last month. I believe that a model fitted to the behavior of crime (with criminal reports as a proxy) in Mexico City should be able to identify the telltale signs of a low, normal or high danger zone by observing what are the crimes that happened nearby and how long it has been since they ocurred. This has the added benefit of being able to compare the change of danger levels in areas with different historic levels of crime. If

the *signature* of a certain location indicates that it is becoming safer or more dangerous, the model would be able to easily report this fact even if historic trends state otherwise.

Because this is in essence a K-nearest neighbors regression, I will employ Annoy in order to construct indexes of locations in addition to a matching dictionary of crimes to their occurrence time, filtering the original database down to only crimes happening in the last month. The prediction will be generated by a neural network that will outpt a score between 0 and 1 when given ten pairs of distances and times, ordered nearest to furthest.

2.1 Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah) is an utility to build tree-based indexes of items as to facilitate the search by similarty. It is built in C++ to quickly create indexes in memory that allow for spatial searching in a reduced amount of time. Note that this is an approximate nearest neighbor search, so it does not guarantee perfect results for all queries.

It works by creating random projections and building n binary trees. For every tree generated, a hyperplane is created to split a random sample of points into two subspaces, (one for each of the child nodes of the root) where the points are then assigned to. This process is then repeated, recursively subdividing the space into this binary tree representation until all nodes have at most k nodes.

After the trees are built, a point's nearest neighbors can be searched for by traversing the tree depending on the hyperplane at each node and where the point is relative to it. The probability of finding nearer points to the query rises as the tree is traversed until the node in which the point is contained is reached. While some points might be missed in this way, by building many trees it is possible to offset the random element of the search by aggregating the nearest neighbors found in each tree, approximating the true nearest neighbors and reducing time complexity. Because this algorithm transforms the problem into a tree traversal algorithm, searching for the approximate nearest neighbors is far faster to solve in comparison to a traditional nearest-neighbors search, which naively has a $O(n^2)$ time complexity.

A big plus of Annoy is its low memory footprint and high speed to load indexes into memory. Because we will make extensive use of nearest-neighbors search, we will parallelize the process and thus load one copy of the index into memory for each process in use.

In the scope of this work, Annoy changes the formulation of the crime score in the following way:

$$S'_{abs}(\bar{p}) = |\{\hat{q} \in Q_{Annoy} | dist(\bar{p}, \hat{q}_{loc}) < \epsilon, timedelta(\hat{q}_{time}) < \delta\}|$$

Where now the crimes we consider, $\hat{q}$, belong to a set Q_{Annoy} such that they are identified as nearest neighbors nondeterministically. With more indexes built, the value of $S'_{abs}(\bar{p})$ should converge to the real value of $S_{abs}(\bar{p})$, relation that also holds for $S'(\bar{p})$ and $S(\bar{p})$.

2.2 K-neighbors Regression

K-neighbors regression is similar to K-neighbors clustering, however instead of a classification problem, the features of the k nearest neighbors are used as the input into a regressor in order to predict a certain value. While finding the nearest neighbors is usually a computationally intensive task, Annoy makes this faster and more efficient. Because of the relation between time and distance features and the difference in importance between variables, the regression I will use is based on a neural network. This should - in theory - be able to model any complex non-linear relationships between variables in order to fit a function with a small avoidable bias such that:

$$\hat{S}(\bar{p}, \hat{q}_1, ..., \hat{q}_k) = \hat{S}(\bar{p}, \hat{q}_{loc\ 1}, \hat{q}_{time\ 1}, ..., \hat{q}_{loc\ k}, \hat{q}_{time\ k}) \approx S'(\bar{p}) \approx S(\bar{p})$$

Where $\hat{S}$ is the regression generated by the neural network trained to predict $S'(\bar{p})$ based on a position $\bar{p}$ and the k nearest approximate crimes $\{\hat{q}_1, ..., \hat{q}_k\}$. Because of Annoy's index and how it approximates the real set of nearest-neighbors, this should in turn become a proxy to predict $S(\bar{p})$.

Usually training a neural network requires a measure of the bayes error (human level error) in order to calculate the avoidable bias. Because I have no such estimate for this problem, I will merely build the neural network iteratively, adding complexity while both the train and development losses keep on being lowered until this is no longer the case.

2.3 Neural Networks

A neural network is, at its core, a composition of functional units that is used to calculate an output from a set of inputs. In this paper I will merely use a multilayer perceptron composed of fully connected layers. Each fully connected layer is implemented as a matrix multiplication in which an input matrix of a size $M \times N$ is fed into linked transformation matrix of a size $N \times O$. The result of this multiplication is passed through an activation function(ReLu for the hidden layers and Sigmoid for the output layer in this paper) and fed to the next "layer" of the network until the end is reached.

Activation functions are used to introduce nonlinearity to the sistem, which in turn makes it capable of approximating very complex functions. A neural network is usually trained by using the backpropagation algorithm.

2.3.1 Backpropagation

After a forward pass from a network, a loss function is computed. The gradient of the loss function with respect to its activation function can be used (if the loss function is convex) in order to change the last layer's weights by the gradient times a learning rate t. After this is done, the process can be continued on the second-to-last layer by calculating the partial derivative of its activation function and applying the derivative chain rule as to obtain another gradient.

Given an n hidden layer neural network, a prediction $\hat{y}$, a cost function C, an activation function Z_i and a linear combination hidden layer H_i, the gradient to calculate the change of the error with respect to the weights of the hidden layer H_k can be obtained via the chain rule in the following way:

$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z_{n+1}} \cdot \frac{\partial Z_{n+1}}{\partial H_n} \cdot \frac{\partial H_n}{\partial Z_{n-1}} \cdot \frac{\partial Z_{n-1}}{\partial H_{n-1}} \cdot \frac{\partial H_{n-1}}{\partial Z_{n-2}} \cdot ... \cdot \frac{\partial Z_k}{\partial H_k}$$

By applying this rule to the biases and weights of each layer, it is possible to iteratively correct the weights assigned to each layer as to reduce the convex loss function. This process is repeated until the input layers of the network is reached, after which we can say an epoch of training has passed.

3 The Data

I will be using the public database provided by Mexico City's government, *Carpetas de Investigación FGJ de la Ciudad de México*. This is a set of police reports by Mexico City's Attorney General's Office detailing the type of crime, date, location and other relevant information. Data profiling yields the following results:

- There are a total of 1,118,989 observations in the dataset as of the date of the download.

- The data is mostly clean, as it only presents some missing values in date and location.
- The only two variables of interest per register are the date and value, so these were preprocessed in order to guarantee them having the same format and no missing values.
- There were alternate dates depending on the report date and the crime ocurrence date. By default I used the later, but if it was unavailable, I took the report date as the crime ocurrence date.
- The positions were given in longitude and latitude. In order to make use of Annoy's functionalities I converted them into euclidean positions. Even though the distance between two such points should be measured using the Haversine Distance (distance over a sphere, measured by longitude and latitude), for small geographic scales the three-dimensional euclidean distance and the Haversine distance output almost indistinguishable results, so the transformation is considered valid for this scale.
- Any register with missing values that could't be fixed was dropped, as they would not work for the task at hand. There were very few such cases across the whole dataset, so this will not affect my model.
- At first I decided to use the most recent data available, however the results presented by the model suggested that the COVID-19 pandemic had transformed the danger levels of areas for which I had very different gut feelings. While this model could very well be representing the current crime rates properly, I decided to take the data in the span of February 1, 2019 to February 1, 2020 in order to create a model that would be more friendly to prior expectations of the users. This dataset has 230,822 items.
- I filtered the data set again in order to extract only the dates from January 1, 2020 to February 1, 2020. This subset of the data will be used as the predictive base for the model. It is far smaller than the yearly dataset, only containing 17,272 items.

4 The Nearest-Neighbors Model

I generated two Annoy indexes that will allow me to make the nearest neighbors queries quickly and efficiently. One will contain the location data and identifiers for the selected year, and the other one will contain the same but for the month that will serve to predict. The later index will also be deployed in production in order to generate the input features for the neural network, so a separate dictionary was constructed linking the crime identifiers to the date.

5 Feature Generation

The first step to generate the features was to generate the scores for each of the locations that presenced crimes in the last year. For this I iteratively broadened the search within the yearly Annoy index until I found all neighbors at a shorter distance than 200m from the interest point. For each one of these the score S was calculated. After this was done, the following distribution of scores was observed:

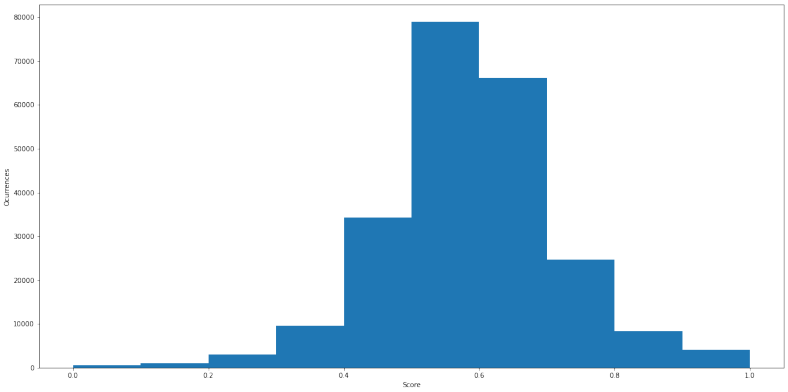

Figure 1: Histogram of the distribution of crime scores.

The scores seem to follow a normal distribution over the range of the data. While this makes sense due to the fact that most places should have an average score, it would be problematic to train the neural network using only this data. Very few samples of extreme score values exist, so there is a chance that the model would not be able to correctly differentiate these cases. Because of this, data augmentation was applied on the new dataset in the following way:

- All locations with a score below 0.3 were filtered into a low density dataset containing 4,645 items.
- All locations with a score above 0.8 were filtered into a high density dataset containing 12,434 items.
- Each sample was resampled n times, and every resample had its coordinates randomly changed by at most 50m on the x and z axis. Dates were also shifted by at most one day.
- The resampling process was done 20 times for each element of the low density dataset and 3 times for each element of the high density dataset.
- Each resampled point had its density calculated and was afterwards added to the dataset.

With the previous process, the following data distribution was achieved:

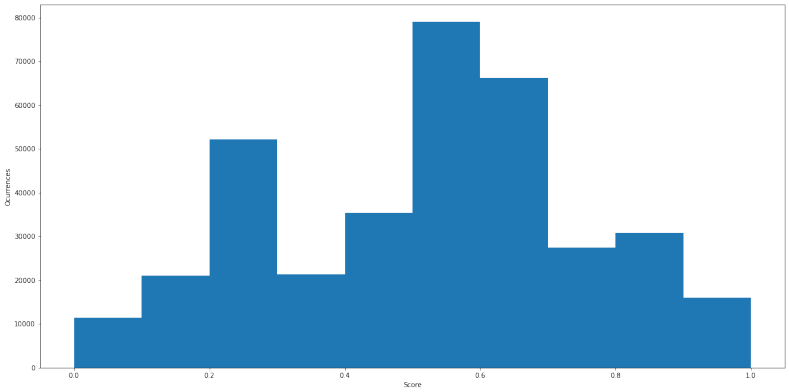

Figure 2: Histogram of the distribution of crime scores (augmented).

The higher amount of extreme values should hopefully allow the network to more easily generalize towards a less biased estimation of the crime score without the need of custom loss functions to train the network. With this enriched dataset of locations centered on crime locations and scores, I finally proceeded to generate the features for training the network by obtaining the ten nearest neighbors in the last month's index and generating the distance between centroids and each one of their neighbors, along with the normalized days between both crimes. A sample of the data obtained can be seen here:

	score	v_0_dist	v_0_tiem	v_1_dist	v_1_tiem	v_2_dist	v_2_tiem	v_3_dist	v_3_tiem	v_4_dist	...
0	0.587151	0.112671	0.700000	0.238930	0.466667	0.240804	0.233333	0.266376	0.266667	0.275522	...
1	0.733958	0.018951	0.166667	0.089886	0.433333	0.098781	0.066667	0.113784	0.066667	0.115626	...
2	0.628651	0.040814	0.033333	0.041370	0.033333	0.050244	0.833333	0.054075	0.066667	0.057232	...
3	0.501681	0.121885	0.366667	0.147557	0.400000	0.175356	0.433333	0.176455	0.100000	0.204137	...
4	0.734773	0.025289	0.733333	0.056600	0.766667	0.077926	0.100000	0.096454	0.966667	0.144502	...
...	...	...	...	...	...	...	...	...	...	...	...
361019	0.254874	0.208194	0.366667	0.230433	0.033333	0.493439	0.266667	0.576054	0.700000	0.597056	...
361020	0.254874	0.229212	0.366667	0.254242	0.033333	0.515213	0.266667	0.599310	0.700000	0.626039	...
361021	0.276802	0.184383	0.033333	0.223806	0.366667	0.446496	0.266667	0.529780	0.700000	0.569909	...
361022	0.254874	0.221931	0.366667	0.242994	0.033333	0.504732	0.266667	0.588381	0.700000	0.615068	...
361023	0.312551	0.183609	0.366667	0.208533	0.033333	0.472497	0.266667	0.552670	0.700000	0.561224	...

Figure 3: Sample of the final dataset, containing 361,024 elements.

6 Training the Neural Network

In order to generate the neural network, I split the dataset into three different sets:

- Train - A dataset used exclusively to train the network, contains 90% of the items.
- Dev - A dataset used to measure the performance of the network while training. In order to prevent overfitting, the weights of the best-performing network on the dev set were the definitive version of each network. Contains

10% of the items.

- Test - A dataset used exclusively to test the network's performance, contains 10% of the items.

The neural network was built iteratively. I started by trying out a simple logistic regression, however the results presented a large bias. Each time I grew the network, I trained it until it converged (the criteria used to determine convergence was 100 epochs without dev loss improvement). The decision as to whether continue making the network more complex was based on the development set loss.

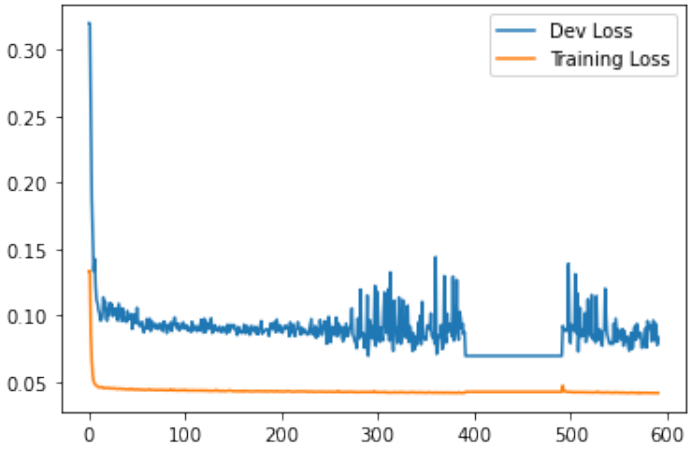

Figure 4: Training loss of the neural network (loss per epoch). Note the significant dip in dev loss between epochs 400 and 500. This was caused by a change of optimizer hyperparameters mid-training. After it apparently converged, further testing of different optimizer hyperparameters yielded no significant improvement, so convergence was assumed.

If the more complex network did better on the development set than the previous iteration of the network, that network was preserved and built upon. If a more complex network wasn't able to perform better on the dev set, I considered the previous generation to be the optimal architecture. This iterative process generated an 11 layer fully-connected network with the following architecture:

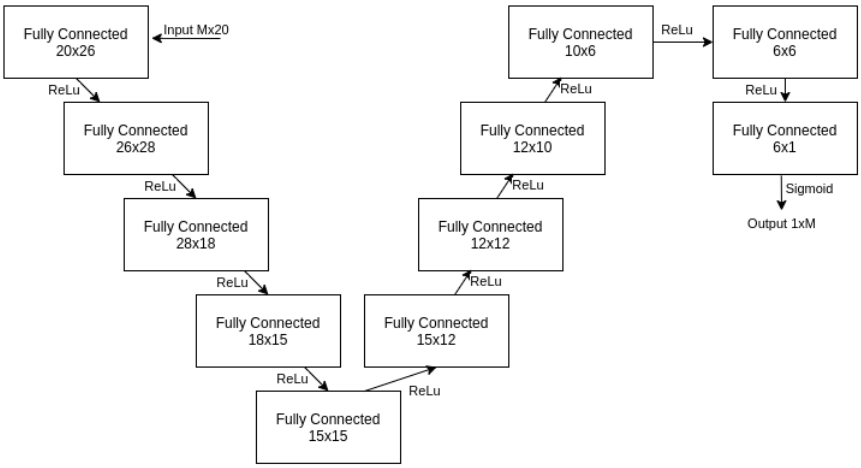

Figure 5: Neural network architecture. It contains a total of 2942 trainable parameters.

The loss function used was the L1 norm, as it is easier to evaluate intuitively than the L2 norm. The network ended up with a training loss of 0.04285, a development loss of 0.0698, and a test loss of 0.05449. Some example predictions next to the their real values are shown below:

	pred	score
0	0.197198	0.207051
1	0.228939	0.205312
2	0.228939	0.436900
3	0.559365	0.573916
4	0.670006	0.617333
5	0.650826	0.658177
6	0.528315	0.549116
7	0.843473	0.824033
8	0.528315	0.551045
9	0.254874	0.220303

Figure 6: Example predictions vs true values. Even though some predictions seem relatively different from the true values, those cases are rare enough to make the model a decent predictor based on the inputs available.

7 Deployment

In order to implement an interactive visualization for the model I used Dash, which allows you to easily deploy HTML pages with interactive Python elements. I added a simple map in which users may query their preferred location with the help of the Google Maps Geocoding API.

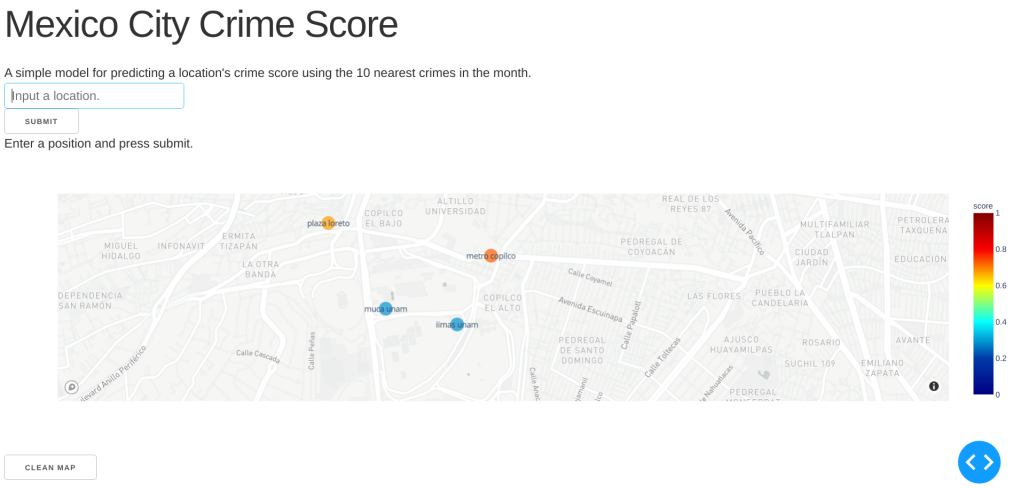

Figure 7: Example queries around UNAM's main campus.

Queries in the map are color-coded to represent their crime scores. In order to generate this prediction, each time a user submits a query to the system, the following process is followed:

- The query is encoded and passed as a request to the Google Maps Geocoding API.
- If there is a valid answer to the query, the API responds via a json file. This response contains the longitude and latitude of the place found.
- The longitude and latitude are extracted from the answer and converted into 3d euclidean coordinates.
- The last month's Annoy index is queried for the 10 nearest neighbors to the coordinates provided. It returns two ordered lists of distances and identifiers.
- Each identifier is looked up in the dates dictionary that associates a crime to its occurrence date.
- An input tensor is constructed using pairs of the distance between the query location and each one of the ten crimes. A similar process happens for dates, however these are divided over 30 (days in a month) in order to normalize the input for the network.
- The feedforward process for the network starts and a predicted score is obtained.
- The predicted score, along with the query name, latitude and longitude are added to a dataframe that is then fed as the input to the interactive map. The points' colors are assigned based on the crime score in order to have a pleasant and clear visualization.

For this example, the query date is set as if it was February 1, 2020. This is due to the precomputed Annoy index used. While I belive that the model should in theory remain valid as time passes, the network might have to be retrained periodically in order to adjust for changes in the city. The one component that should always be kept up to date should be the Annoy index and its associated date dictionary, as these are time-sensitive and they serve as the backbone of our predictive model. Because the process of building them is almost instant, doing so periodically should present no problems for a productive enviroment.

8 Conclusions and Possible Improvements

The support of the validation and test sets in addition to the performance metrics obtained lead me to believe that this is a decent model for the task at hand. Sanity checks done by querying certain locations I know well further reinforces this feeling. I believe that this model adds value to the data, and it might even have potential for the following tasks:

- Tracking the latest trends of crime in the city.
- Planning routes for anyone trying to keep themselves safe, as this allows for a data-driven and less biased outlook on the safety in the city.

However, before any serious deployment is planned, there are several steps to be taken in order to assure the system's reliability:

- Test the trained neural network with datasets created from other years. Does it still work or does its peformance decay?
- Seek expert advice as to how to overcome the low report rate problem, if it has any effect on the system.
- Extensive tuning of the network should be done, maybe combining different prediction years into one training dataset to see if the performance is improved.
- Add more real-time based features such as weather conditions, day of the week or time of day in the determination of the score.
- Try to vary the number of nearest crimes in order to predict. It is possible that the model overfits slightly due to having more features than necessary, or it may also be lacking features that could further drive down the development set loss.
- Check if the same model could also be used to predict the calculated crime score in other cities.

To conclude, I believe that the model is functional and it shows promise to become a tool that provides useful insights into the huge problem of crime within Mexico City. Either as a decision-aiding tool for the people or as a tool to track patterns of crime before they are actually noticed, it is my belief that this model could be of use to many, which leads me to the conclusion that this project as a whole was worth the time and effort.