Using google map API grabs features to implement robbery prediction

Abstract

I wonder know that relation between a feature of location between crime risk. Therefore, I use a crime data and street lights from SD city database, google map API, and Yolo to collect my data object. I focus on the robberies which happened in years 2010 to current, 626 events, and random picked 600 locations which don’t have robberies in 2010 to current. After that, I choice how many cars, how many windows, how many street lights, and distances to the closest SDPD, convenience store, gas station, and lamp to be features. Last but not least, I choice k-nearest neighbor and AdaBoost to train models and predict result.

Introduction

The area where has high probability of crime occurred might have similar features. For example, the place where doesn’t have street lights and people rarely walked along is easier to be robbed. Usually, robbers might try to target a victim who doesn’t close to a police station and might not dare to rob around polices. Moreover, robbers would try to reduce a chance to be seen when they rob, so distances to stores, convenience store or gas station, which open 24 hours might affect a probability of robbery. According to above assumptions, I used google map api, “nearbysearch”, “distancematrix” and “streetview”, to get street view images and distance to stores. Using dataset of crime gets information of locations where robberies happened.

The tools which I introduce into, not only documents and google map API, but also the YOLO object detection. It a detector which can assist to detect labels in an image. I used Yolo to detect features which aren’t provided by documents. For instance, how many cars or how many windows around a location. In this research, I used 237 images to train YOLO.

After data collected from 626 coordinates of robberies and 600 coordinates of safeties, I used K nearest neighbor and Gaussian Naive Bayes to train and predict. If the accuracy is good enough, then I am able to predict a probability of robbery of an unseen location.

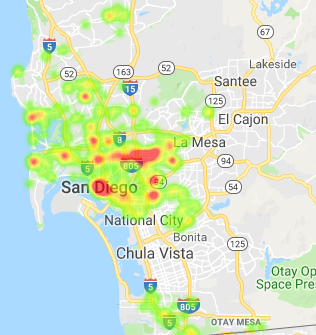


Figure 1 A histogram of crime incident

Data processing

Collect data from database.:

At first, I found a document of crime in 2007 to 2013 from San Diego regional data library. It contains different type of crimes and each of incidents has year, a distance to the closest lamp, and a coordinate. Also, there is a document which contains 61K street lights locations, from

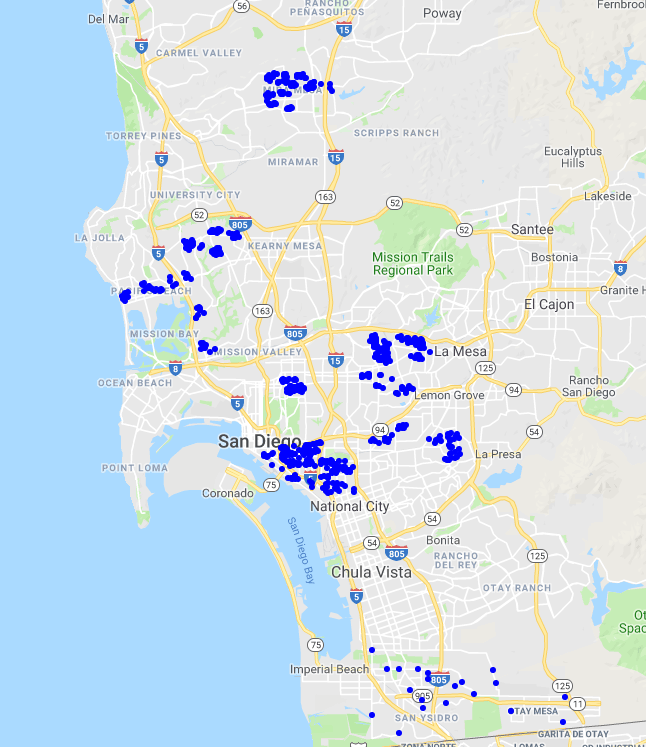
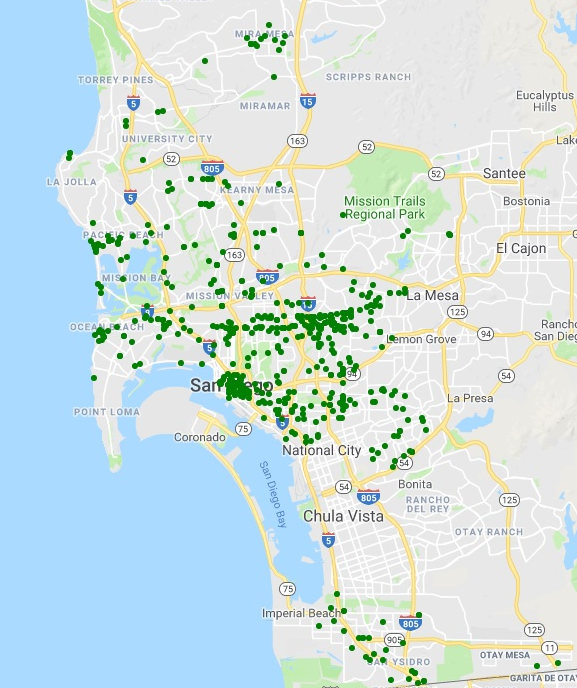
Data San Diego. According to those two files, I focused on the robbery which happened in 2010 to current, because a place where incidents happened in the past might have different street views. For example, the area might open a new convenience store or place a new street light in 2011 but the incident happened in 2009. Then the data collected by documents or google api don’t correspond to the real situation. After that, I followed the robbery incidents map in San Diego city to randomly pick 600 safe points to be safe locations data.

Collect data from Google API:

Secondly, I used google api to grab street view for each coordinate of safe and crime. Every coordinate got 4 images for different heading. Then I could get 4 images around a coordinate. These images would be saved for later steps which detect features in images.

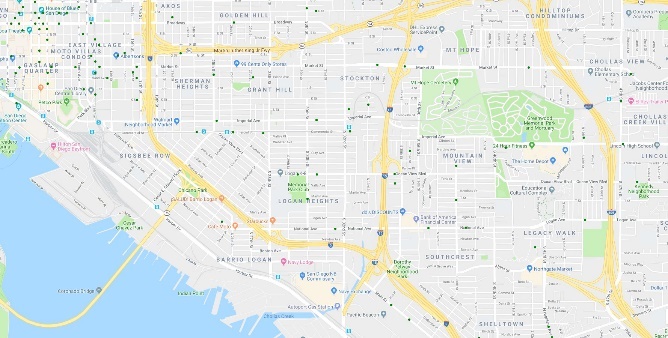
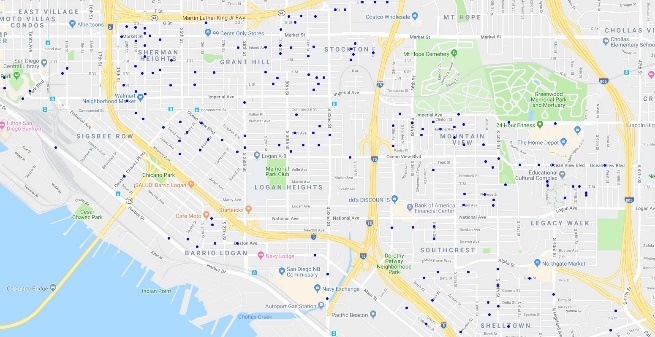
Four different heading at a location.

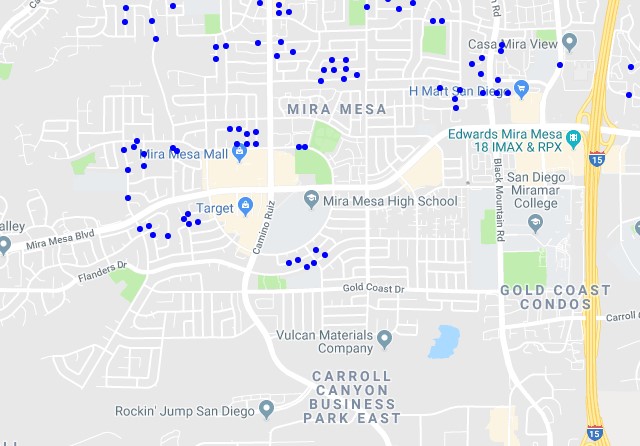
Safe sample points map Robbery incidents map

Robbery incidents map (South) Safe sample points map (South)

Robbery incidents map (Downtown) Safe sample points map (Downtown)

Robbery incidents map (North) Safe sample points map (North)

Then using “nearbysearch” to get coordinates of SDPD, gas station, or convenience store could find the closest one. The google had returned a json data, so I would read data and extracted the closest for calculating distance. After extracting a coordinate, insert two points into “distancematrix” can get a distance between two points.

Collect data from street view by YOLO

Third step, it’s the most complicatedly part, since I want to detect a feature an image. The way is introduced by me is YOLO. It’s a source which uses a pretrain data to train a label which I want to detect. For every image I had to select an area and give it a label name at beginning. Then I run the training function to train my model by those data which I created. I used 237 images with features of street light and window and took 8 hours to train my model. The model can detect most of street light and windows in an image. Also, the pre-trained model can detect cars, so I didn’t train my model to detect cars. After completely trained my model, use the model to scan every street to get a result. In training step, I didn’t use the result about street lights, since number of lights have already been assigned by using the document. It saves time to check the result about street lights. For example, in the below there are two car and one window.



Methods

Grab images and Label detection:

1. Google API (streetview, nearbysearch, distancematrix)

2. YOLO (Neural network)

Classification

1. K nearest neighbor

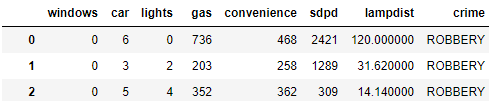
2. AdaBoost

3. Normalization

Analysis

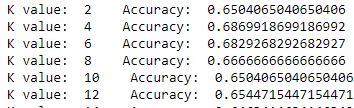
The dataset which contains columns, windows, car, lights, gas, convenience, sdpd, lampdist and crime. The X values are windows, car, lights, gas, convenience, sdpd and lampdist. The Y value is crime.

Without normalizing



K nearest neighbor

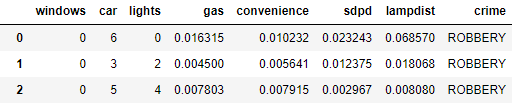
Tried different K and found K = 4 has highest accuracy. It’s 68.69%



AdaBoost

Using AdaBoost with decision tree, I get an accuracy 80.8%

With normalizing

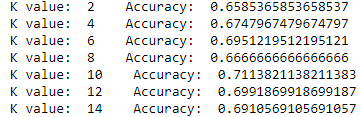


That is because values of distances are larger than numbers, so the distances influence much more than numbers. Therefore, it should normalize those columns which record distance. The above table shows the data which have been normalized.

K nearest neighbor

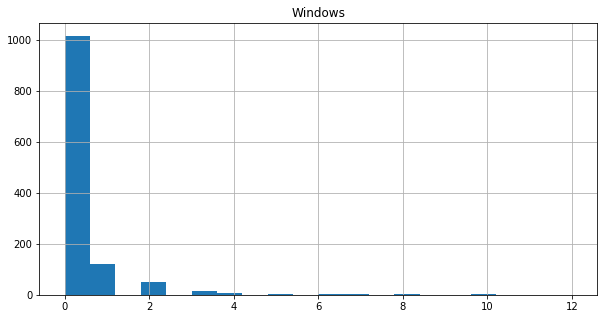
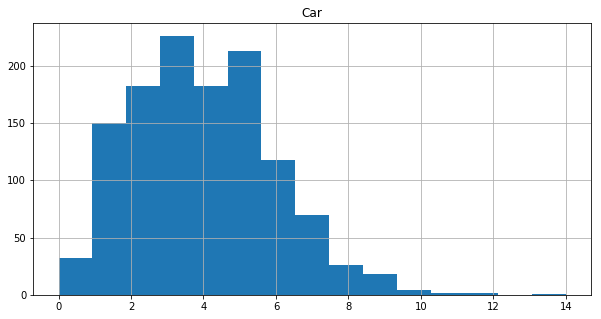
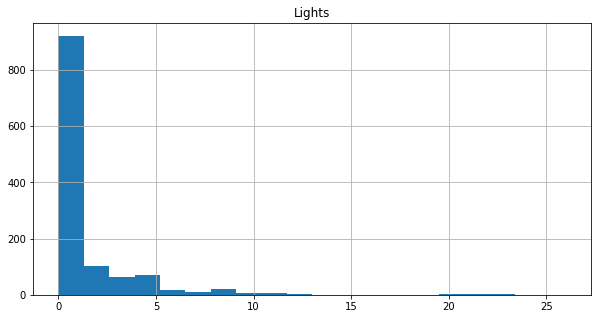
The accuracy improved after normalizing four columns.

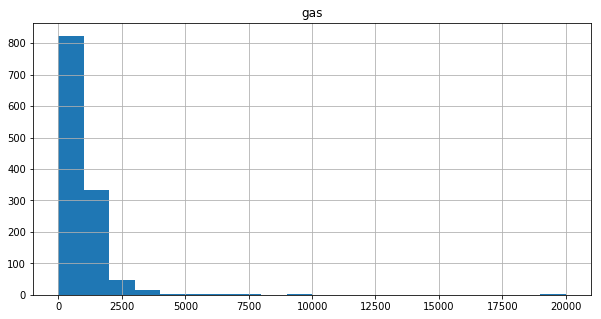
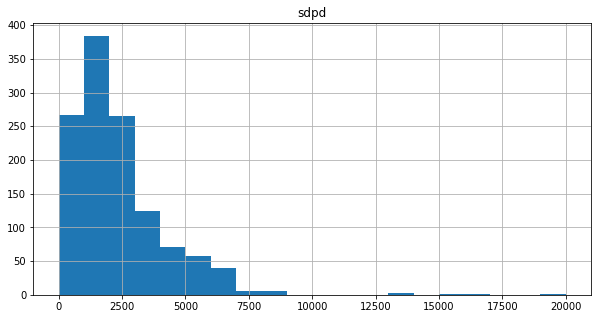
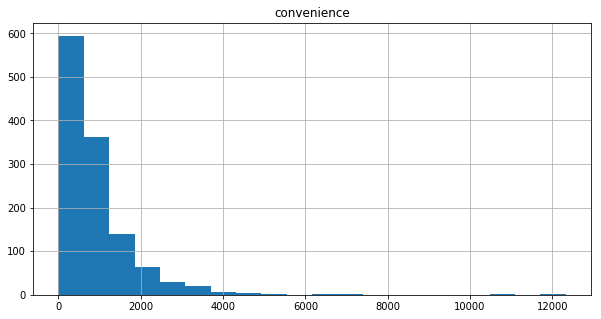
Tried different K and found K = 10 has highest accuracy. It’s 71.13%

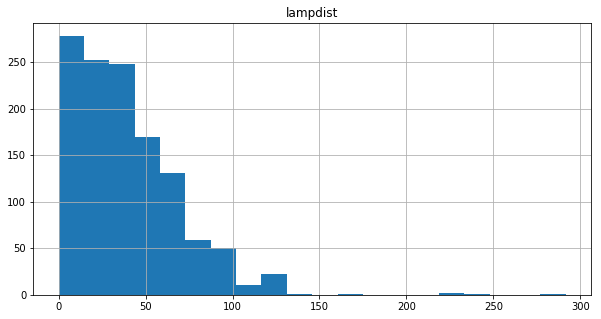


Histogram of each features

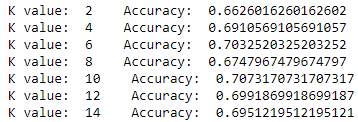
I found that most of windows are zero by histogram.



Consequently, I ignore the windows feature, and the result is similar to original one. It means windows in my dataset doesn’t contribute to model training.



**Conclusion and future works**

The result shows that the highest accuracy is only 71%. It isn’t good enough to support the assumption and the prediction. Additionally, my YOLO detection doesn’t provide high confidence in detecting windows or lights. It causes I cannot get enough information about windows and lights. I need more images and different sides of windows to train my model. The YOLO pretrain model demonstrates it can have very high accuracy to detect objects. If I have various data, then my YOLO will detect accurately. The more complete, the higher accuracy.

Reference