Break-In Monitor: Machine Learning Based Probability Predictor for Car Break-Ins

By Rene Figueroa, Alan Hsueh, Vignesh Thyagarajan, Haoran Chen

***Abstract*—Car break-ins are very common in large metropolitan areas and also prevalent in small cities around the country. By predicting car break in patterns, law enforcement agencies will be better equipped to handles these acts of vandalism. Moreover, even everyday car owners can feel safer about their parking spaces.**

Keywords—Machine Learning, Python, TensorFlow, Keras, Supervised Learning

I. Introduction

The number of car break-ins have increased tremendously in the Santa Clara county. In Milpitas alone, there have been 145 car break-ins in the first two months of 2019. (1) This problem can be addressed by dispatching more law enforcement officers to hot spot areas. We developed an application that enables users to query their own car break-in specification that suits their needs. Thus, the ML calculated result can be displayed in a map for a more user-friendly experience. We used supervised machine learning to create our model; specifically, we used classification to provide the user with a probability of a given car being broken into with the provided parameters. The data used to train the model comes directly from the Santa Clara county. (2) The time frame for the training model is from November 9th, 2017 to February 18th, 2019.

II. Project focus

The focus of this project revolves around trying to solve the problem of car break-ins in the San Jose area. The rate of crime in Santa Clara has greatly increased in the last few years, and knowing the areas in which there may be a higher chance for a car break-in would be beneficial to car owners and law enforcement individuals situated in the Santa Clara area. Since there are a lot of factors that help elevate the chance of a car break-in, such as day, time of day, as well as location, our project aims to give users a platform to predict the chances of their car being broken into on any given day, any given time, and any given location in the Santa Clara area.

III. Architecture

A. Technologies Used

- Python with Flask
- Jupyter Notebook
- Pip, pandas, tensorflow, numpy, scikit-learn, scipy, matplotlib, sodapy
- HTML/CSS
- Bootstrap (DASH.IO template)
- Google Maps API

Our team opted to implement a Python-based backend for the purpose of the Break-In Monitor. This was due to the numerous libraries that would be beneficial for our project, including the Python libraries of tensorflow, keras, scikit-learn, etc. These tools enabled the training of a Jupyter Notebook which would be the basis of our ML backend. In terms of frontend, we used a bootstrap dashboard template (DASH.IO) and modified it greatly to suit our needs for the project, which included HTML, CSS, and Google Maps API. Ultimately, our UI turned out very polished and professional looking, with the capability of working on mobile platforms.

B. Machine Learning Model

In order to create the model, we used supervised machine learning. The data we obtained for the Santa Clara contains an abundance of information that can be integrated in the machine learning model. We decided to use latitude, longitude, day of the week, and time as the features for our machine learning model. In order to create the machine learning model, we had to perform the following steps:

- Obtain raw data.
- Preprocess the data.
- Build the model.
- Train the model.

We obtained our data for only the scope of Santa Clara county. The raw data was obtained as pandas frames and contained information that was needed for the machine learning model. (Fig. 1) There was a total of 1761 samples of features and labels that have theft from vehicles. Samples when there was no car burglary was also appended to the data set. In total, there were 2761 samples. 80% of the samples were used to train the model and the remaining 20% was used to test the model.

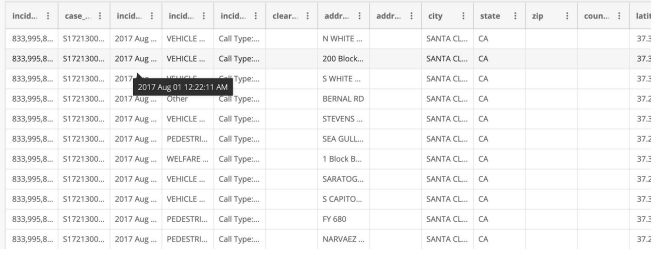

Fig. 1 Raw data for machine learning model

The first step in preprocessing the data was to map the parent incident type of "Theft from vehicle" to a numerical value of 1 and every other incident type to 0. We then mapped the days of the week to numerical values. (Fig. 2)

	latitude	longitude	hour_of_day	day_of_week	parent_incident_type
0	37.388892	-121.985716	19	3	1
1	37.258594	-122.120620	11	6	1
2	37.194711	-121.992883	5	2	1
3	37.329350	-121.898919	3	0	0
4	37.332314	-121.892714	13	2	0
5	37.332004	-122.031400	2	5	1
6	37.322955	-122.022169	7	2	0

Fig. 2 First step in preprocessing

Consequently, the next step was to transform the sets of data to numpy arrays and to normalize the data. The data is split into a labels and features with 80% being used for training and 20% being used for testing purposes, as stated before. (Fig. 3) After the data is split into features and labels, it is scaled. By scaling the data between 0 and 1, the machine learning model can process the data more efficiently. This was completed using the MinMaxScaler function from the sklearn library to scale the data. (Fig. 4)

```
training_data = sample_data[0:math.ceil(len(sample_data)*0.8)]
testing_data = sample_data[len(training_data):len(sample_data)]

#labels
training_data_labels = training_data.pop("parent_incident_type")
testing_data_labels = testing_data.pop("parent_incident_type")

#convert to numpy arrays
training_data = np.array(training_data)
testing_data = np.array(testing_data)
training_data_labels = np.array(training_data_labels)
testing_data_labels = np.array(testing_data_labels)
```

Fig. 3 Data transformation

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_training_samples = scaler.fit_transform(training_data)
scaled_testing_samples = scaler.fit_transform(testing_data)
scaled_training_samples
```

Fig. 4 MinMaxScaler

The next step was to create the machine learning model using the current data. (Fig. 5) This was done using the sequential model from Keras to create the model. (Fig. 6) The neural network has 3 total layers: the first layer consists of 16 neural sets, the second layer contains 32 sets, and the final model (which its the output) consists of 2 neural sets. The last neural set represents the probability of a car getting broken into. The input shape for the model is 4, since there are 4 features.

In order to train the model, we utilized the Adam optimizer as it is considered the best-practice optimizer. An optimizer is a specific implementation of the gradient descent algorithm, which changes the internal variables a slightly every time to gradually reduce the loss function. Thus, model has reached 70% accuracy with data from only less than 2 years ago. (Fig. 7) This model was saved to be integrated with the backend of our application.

```
array([[0.89269067, 0.20438886, 0.82608696, 0.5       ],
       [0.64769444, 0.06383257, 0.47826087, 1.        ],
       [0.52757518, 0.19692068, 0.2173913 , 0.33333333],
       ...,
       [0.78316838, 0.10657332, 0.82608696, 1.        ],
       [0.76866217, 0.14838359, 0.82608696, 1.        ],
       [0.87158157, 0.03743571, 0.52173913, 0.83333333]])
```

Fig. 5 Final preprocessed data

```
model = tf.keras.Sequential([
    #only first layer of the sequential model needs to know the shape of the input data
    tf.keras.layers.Dense(16,input_shape=(4), activation=tf.nn.relu),
    #second layer
    tf.keras.layers.Dense(32, activation=tf.nn.relu),
    #last layer is the output layer, we only have 2 units for car burglary or no car burglary
    tf.keras.layers.Dense(2, activation=tf.nn.softmax)
])

model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                80
_________________________________________________________________
dense_1 (Dense)              (None, 32)                544
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 66
=================================================================
Total params: 690
Trainable params: 690
Non-trainable params: 0
```

Fig. 6 Keras sequential model

```
model.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])

model.fit(scaled_training_samples, training_data_labels, batch_size = 10, epochs = 10, shuffle = True, verbose =2)
```

```
Epoch 1/10
 - 0s - loss: 0.5381 - acc: 0.7162
Epoch 2/10
 - 0s - loss: 0.5389 - acc: 0.7184
Epoch 3/10
 - 0s - loss: 0.5366 - acc: 0.7184
Epoch 4/10
 - 0s - loss: 0.5353 - acc: 0.7230
Epoch 5/10
 - 0s - loss: 0.5366 - acc: 0.7239
Epoch 6/10
 - 0s - loss: 0.5353 - acc: 0.7198
Epoch 7/10
 - 0s - loss: 0.5349 - acc: 0.7220
Epoch 8/10
 - 0s - loss: 0.5339 - acc: 0.7202
Epoch 9/10
 - 0s - loss: 0.5333 - acc: 0.7193
Epoch 10/10
 - 0s - loss: 0.5333 - acc: 0.7239
<tensorflow.python.keras.callbacks.History at 0x1437707f0>
```

```
test_loss, test_acc = model.evaluate(scaled_testing_samples, testing_data_labels)

print('Test accuracy:', test_acc)
```

```
552/552 [==============================] - 0s 24us/sample - loss: 0.5554 - acc: 0.7047
Test accuracy: 0.7047101
```

Fig. 7 Adam optimizer and model accuracy

C. *Backend*

For the backend of our application, Flask was used. We created different API endpoints to obtain different types of information. The most important endpoint was the probability endpoint. (Fig. 8)

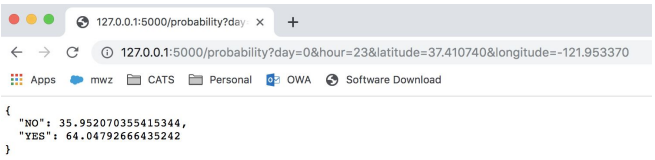

Fig. 8 Calling the probability endpoint

We also stored all the queries requested by the user in a database. These can be requested with the following API call: "current_locations." (Fig. 9)

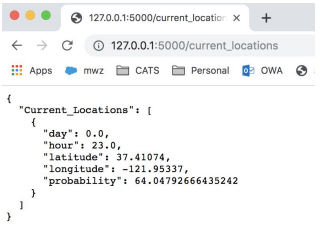

Fig. 9 Calling current_locations

D. *Frontend*

For the frontend of our application, we used a bootstrap HTML/CSS template (DASH.IO) as a base and modified it to suit our needs. This provided a fluid and polished UI that could support both mobile and tablet views as well. The UI will be demonstrated in the following sections.

IV. Process Flow

A. *Python-based Machine Learning Models*

Break-in data was obtained from the Santa Clara county sheriff, then parsed and analyzed with three different machine learning models to calculate break-in probability:

- Load_Data Model
- Training Model_V1
- Training Model_V2

Initially, the Load_Data_Model is used to load data from ‘moto.data.socrata.com.’ (Fig. 10)

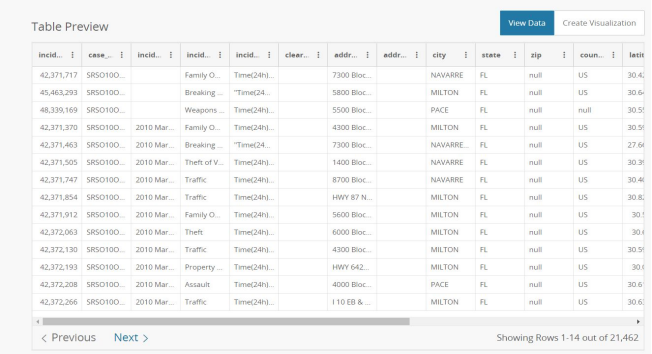

Fig. 10 Crime_data_set

Both Training model Model_V1 and Model_V2 were thus used to train the data set. the training model calculates the probability for a car break-in using get_current_location_probability by taking in each table’s parameters. (Fig. 11)

```
def get_current_location_probability(model, current_location, testing_data):
    scaler = MinMaxScaler(feature_range=(0,1))
    current_testing_data = np.append(testing_data, [current_location], axis=0)

    #print(current_testing_data) <-- uncomment to see the location added to the testing data set
#scaled the testing data
    current_testing_data_scaled = scaler.fit_transform(current_testing_data)
#now we have normalized the data and to access the current location we use [-1]

    current_prediction = model.predict(current_testing_data_scaled, batch_size = 10, verbose = 0)
    return list(current_prediction[-1])
```

Fig. 11 get_current_location_probability()

B. *Frontend: Main Page and Query Page*

The main page serves as a splash page for users to become acquainted with the application’s purpose, as well as direct them to subsequent pages. (Fig. 12) These include the table and map pages.

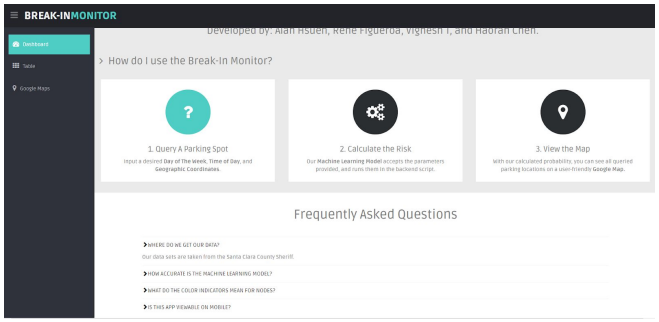

Fig. 12 Home page

By utilizing HTTP requests, the table page (Fig. 13) can allow entry for several input forms that can be submitted which in turn will update the table of parking locations. The query form requires the day of the week, time of day, latitude, and longitude values. (Fig. 14)

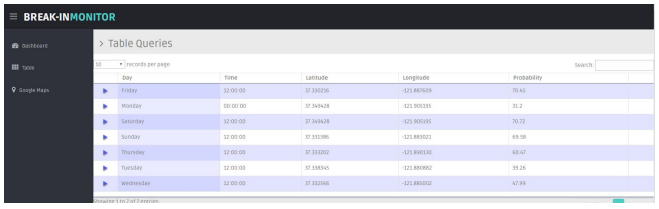

Fig. 13 Table page

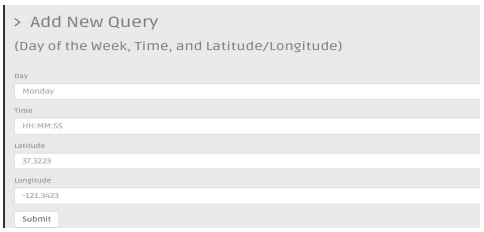

Fig. 14 Query form

The back-end server will calculate the probability of the given parameters and respond using a JSON form. (Fig. 15)

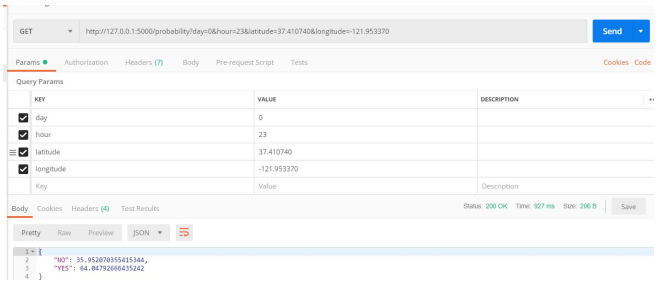

Fig. 15 JSON backend form

C. *Frontend: Google Maps API*

The maps page utilizes Google Maps API to plot points defined in the locations table, which is stored as an array of tuples. (Fig. 16) Each respective point displays the index of the location, and supports the ability to click on a node for detailed information concerning the parking location. Nodes are color coded by risk, with red being the highest risk chance and green being the lowest. Upon clicking a node, it will display the probability of break-in in a readable format, along with the rest of its parameters. (Fig. 17)

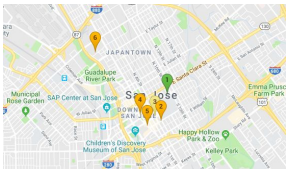

Fig. 16 Map page

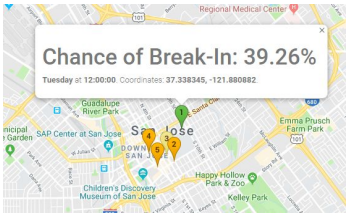

Fig. 17 Map node details

V. Future Enhancements

There are many future enhancements that can be made to this application to help it scale out for a much wider user base. These enhancements, along with feedback from Professor Ranjan are listed as following:

A. *Expanded Geographical Base*

The first enhancement would be to expand the geographical base of the application as a first priority. As of now, the application is situated in Santa Clara county and the model is trained off of data collected only for this county. By collecting more data from different areas and training the ML model accordingly, we can expand the reach of the application from just Santa Clara county to possibly all of California, and eventually even the entire USA.

B. *Expanding Beyond Car Break-ins*

In the future, it would be desirable to remove the limitation of predicting car break-ins to predicting many more types of crime, including non-vehicular theft or assault. While this may be harder to categorize and predict, the data exists, and predictions can be made using machine learning in similar fashion to how car break-ins were calculated. This would not only help more people, it would help direct law enforcement to monitor situations in high probability areas.

C. *Professor Ranjan's Feedback*

Professor Ranjan was gracious enough to provide feedback for our application, which helped us better understand the scope of the project and how we can expand it in the future if given an opportunity. His feedback included automating the querying through the use of location features on smartphones and the Google Assistant API. The automation of queries would substantially improve the user experience of this application. Another piece of feedback by Professor Ranjan was to look into options of licensing this technology to other companies rather than starting a product from scratch. This would also expand the horizons of this technology as it could see more widespread use with different enterprise companies acquiring the product and implementing it.

Conclusion

Using machine learning to predict car break-ins is an idea that can be incredibly useful for car owners and law enforcement officials alike. As crime rises in the Santa Clara county area, the Break-In Monitor may be necessary in order to curb the rate of crime that occurs and better inform officials to take action.

Acknowledgment

We'd like to acknowledge and thank Professor Ranjan for approving our project idea and giving us positive feedback on the scope of our idea. Thank you for your support and willingness to give an opportunity for us to learn through a practical project such as this one.

Repository link

https://github.com/SJSU272Spring2019/Project-Group-12/tree/master

References

[1] https://abc7news.com/car-break-ins-on-the-rise-in-milpitas/5170600/

[2] https://moto.data.socrata.com/dataset/Santa-Clara-County-Sheriff-s-Office/wrmr-tdyp