San Francisco Crime Data Analysis

Renz Pascual

Contents

Problem

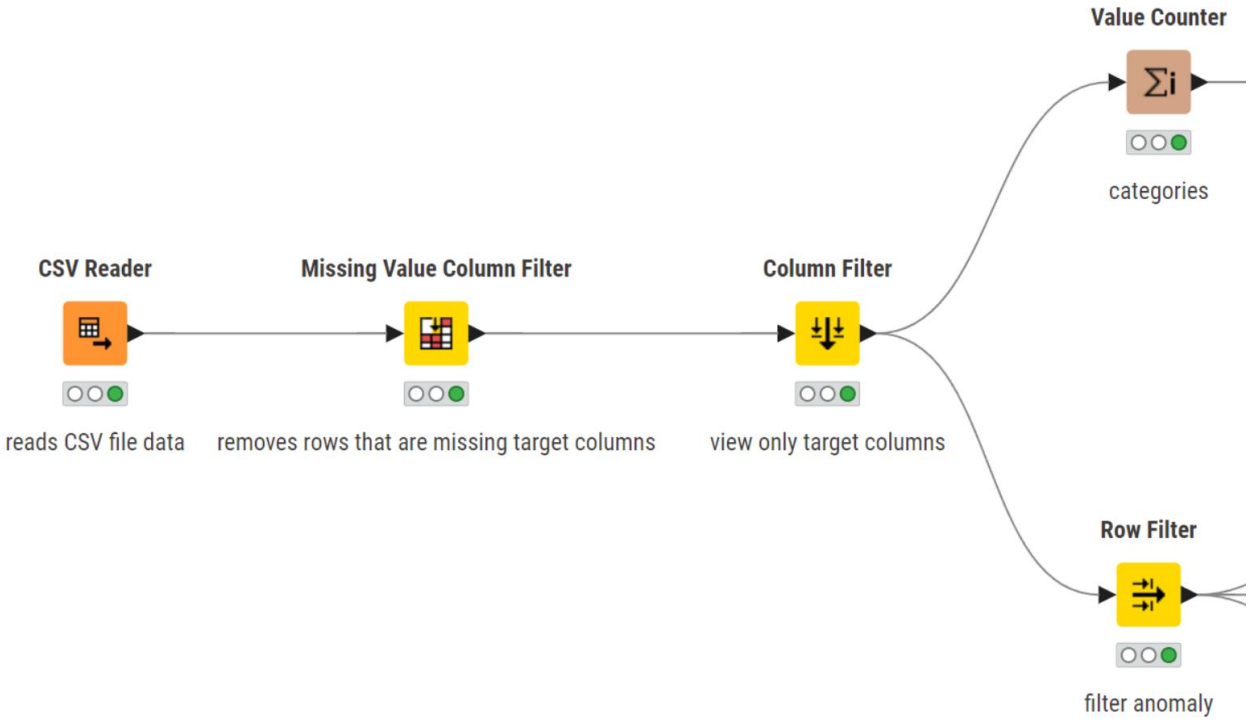
I wanted to identify the crimes documented in the San Francisco area and thought it would be a good idea to analyze the type of crime, the frequency of crimes, and possibly find if there were any patterns in crime events. I found a giant data set consisting of over two-million data points with over a dozen meaningful columns of data. https://data.sfgov.org/Public-Safety/SF-Crime-Heat-Map/q6gg-sa2p

The data was collected from incident reports from 2003 to 2018. It was discontinued in 2018 because it was prone to issues and caused delays in data accessibility.

Pre-processing

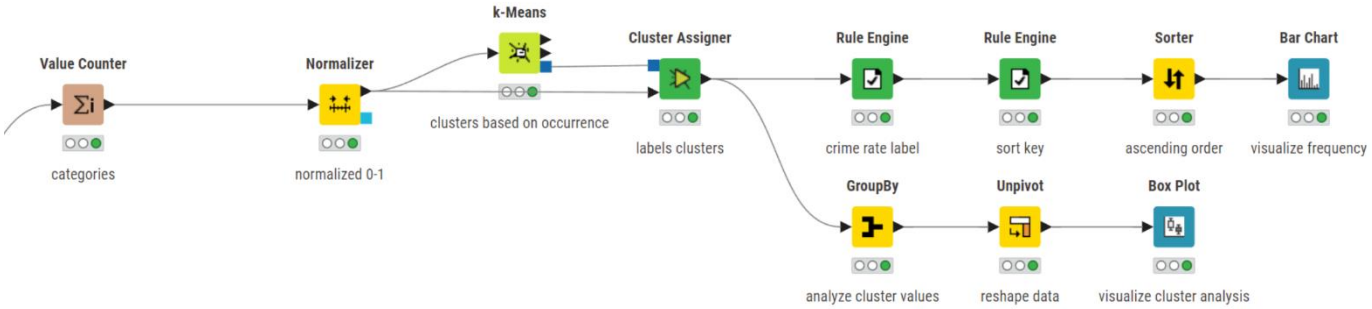
I used the KNIME Analytic Platform to process the data. There are three main sections

Filtering



1. I downloaded the dataset (CSV file) and read them into the CSV Reader node.
2. I removed any rows that were missing data in any columns I needed.
    a. Category, DayOfWeek, PdDistrict, and Time
3. Then I filtered the columns, showing only what I needed to see (2a).
4. The output is then pushed into a Value Counter node and a Row Filter node.
    a. Value Counter counts the # of times a "Category" appears in the dataset.
    b. Row Filter removes any "NA" values in the PdDistrict column.
        i. Looking at all the different PdDistricts, only one row had a value of "NA" which was an anomaly.

Cluster Analysis



1. The normalizer node reads the "count" column from Value Counter which then scales that column using decimal scaling.
    a. Main reason why we use it is because the next node, k-means, is sensitive to magnitude, normalization ensures that a single category with a large count doesn't dominate the clustering.
2. k-Means node clusters the normalized frequency data, grouping the crime categories into clusters based on how often they occur.
    a. You can see which crimes are similar in frequency.
3. Cluster assigner reads the cluster model generated by k-Means and takes each row and assigns it a cluster based on the cluster model results.
    a. cluster_0, cluster_1, cluster_2, cluster_3, cluster_4, and cluster_5
4. The first rule engine assigns labels to the clusters and appends that to a new column "Crime Rate Label".

```
$Cluster$ = "cluster_2" => "Very Common"
$Cluster$ = "cluster_1" => "Common"
$Cluster$ = "cluster_4" => "Average"
$Cluster$ = "cluster_3" => "Uncommon"
$Cluster$ = "cluster_0" => "Rare"
```

    a. GroupBy node aggregates the clustered data, computing the mean/min/max counts for every cluster.
    b. Unpivot node reshapes the columns so that mean/min/max are not longer separate columns.
        i. This makes it so that when visualizing the data, it isn't use mean/min/max as columns.
    c. Box plot node visualizes the data and lets me determine the appropriate labels for the clusters. (fig. 1)
5. The second rule engine node converts the categorical "Crime Rate Label" into ordinal labels and appends that to a new column "Sort Key". (e.g. 1,2,3,4,5)

```
$Crime Rate Label$ = "Very Common" => 1
$Crime Rate Label$ = "Common" => 2
$Crime Rate Label$ = "Average" => 3
$Crime Rate Label$ = "Uncommon" => 4
$Crime Rate Label$ = "Rare" => 5
```

6. The sorter node sorts the rows based on the "Sort Key" column in ascending order.

a. The main purpose of this is when it is visualized via bar graph, it goes from very common to rare in ascending order.

7. The bar chart node creates a bar chart comparing the number of unique crimes present in each cluster. (fig. 2)

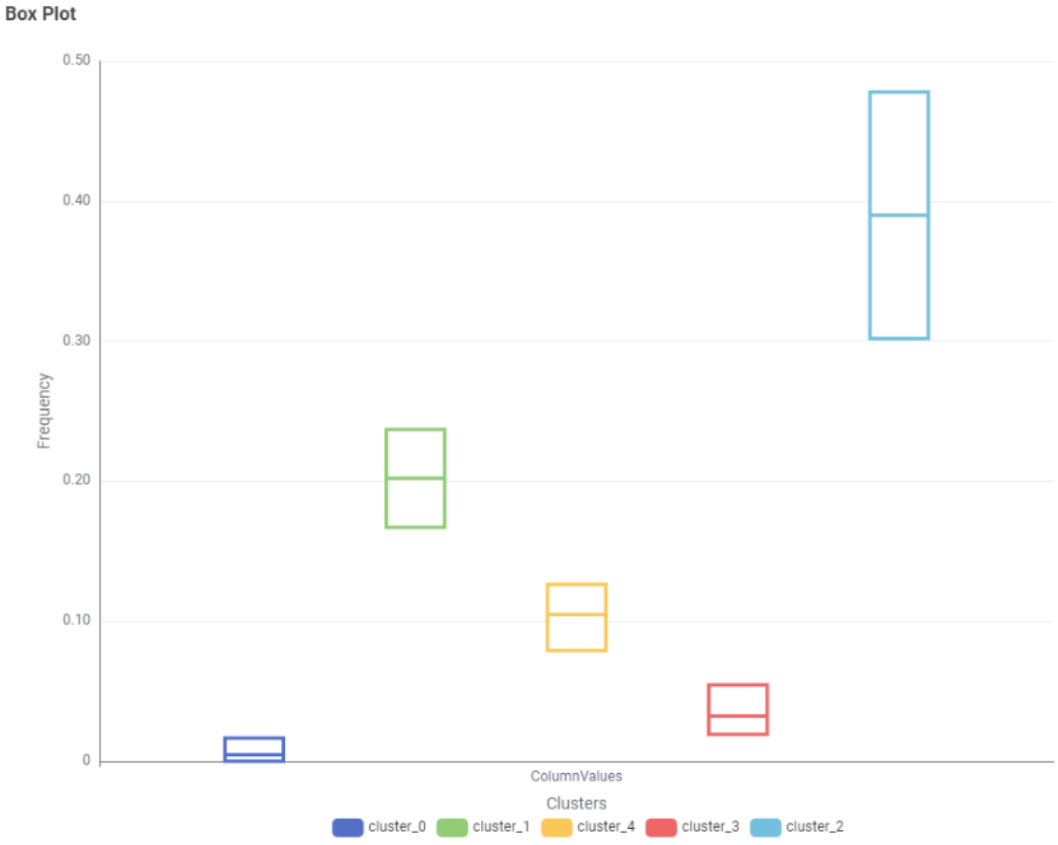


Figure 1 (top)

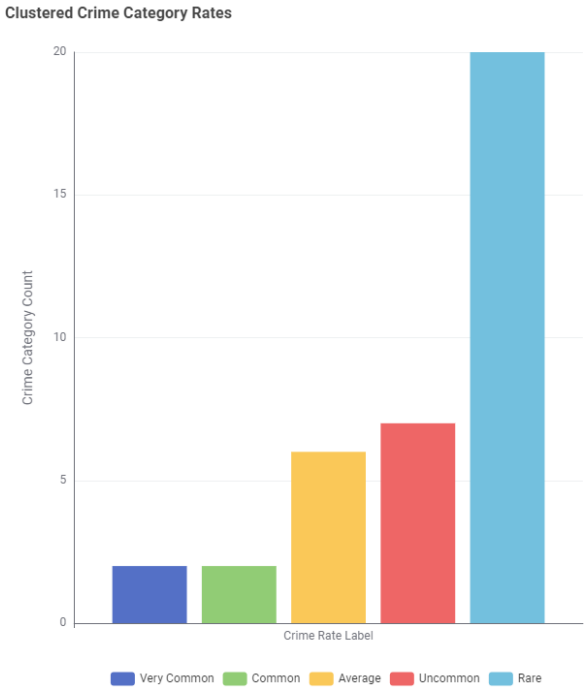
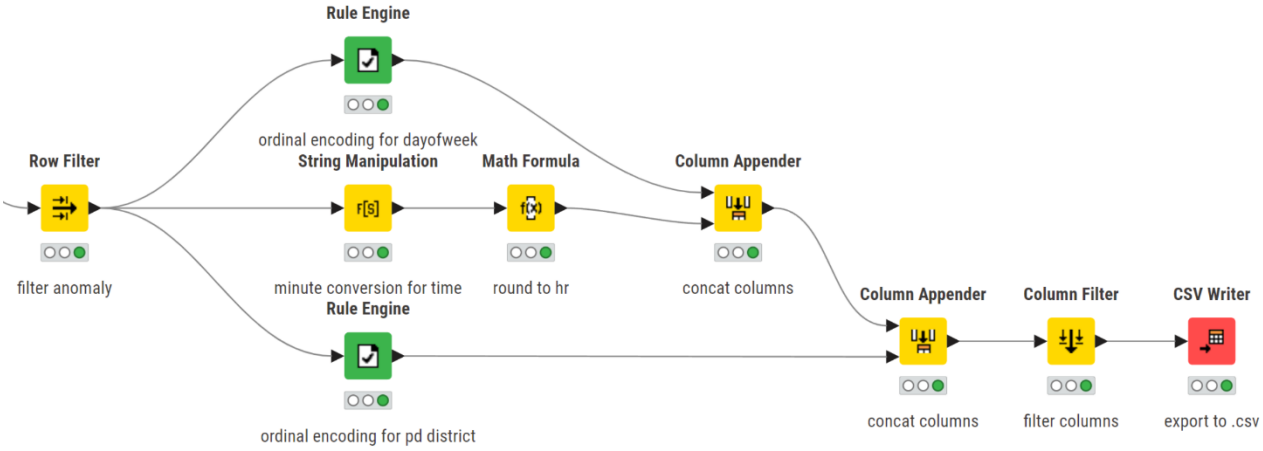


Figure 2 (right)

Predictive Modeling



1. The first topmost rule engine node converts the DayOfWeek column into an ordinal column which is appended as “ordinalDOW”.
2. The string manipulation node converts time from a date format into an integer format in minutes, this is appended to a new column “Time_Minutes”.
    a. The values are then rounded to the nearest hour for simplicity’s sake.
3. The bottom rule engine node converts the PdDistrict column into and ordinal column which is appended as “ordinalDistrict”.
4. The column appender concatenates the outputs from the top rule engine node and the out from the math formula node.
5. The second column appender concatenates the output of the first column appender and the output of the bottom rule engine.
6. The column filter takes an input that has all the new columns and filters the necessary columns.
    a. (e.g. Category, ordinalDOW, Time_Minutes, and ordinalDistrict)
7. The tables are then written into an excel csv file “crimeTime” using CSV Writer.

The csv file “crimeTime” is then loaded into mlp.py. **Further explanation** is present in the Jupyter Notebook (.ipynb) on the GitHub.

There are two versions of the predictive model.

Version 1 = mlp.py + crimeTime.csv.gz **for local machine**

Version 1 = mlp.py + crimeTime.csv **for Jupyter Notebook (unzip the crimeTime.csv.gz file)**

Version 2 = mlp2.py + crimeTime2.csv.gz for **local machine**

Evaluation

Cluster Analysis

LABEL	Crime Types	Frequency
Average	6	0.162
Uncommon	7	0.189
Very Uncommon	2	0.054
Common	2	0.054
Rare	20	0.541

Figure 3

LABEL	MIN	MAX	MEAN
Average	0.07909	0.12623	0.10479
Uncommon	0.01919	0.05447	0.03224
Very Uncommon	0.30187	0.47798	0.38992
Common	0.16704	0.23693	0.20199
Rare	0.00001	0.01650	0.00457

Figure 4 (in millions of counts)

From the bar chart (fig. 3), we can see that most crimes ended up in the "Rare" cluster, indicating that the largest number of distinct crime types occur infrequently. Only a couple crimes were labelled as "Common" and "Very Common" which means that those crimes happen a lot more often than most.

The distribution is very skewed. A small number of crime types account for a large portion of occurrences in San Francisco, while a vast majority happen relatively rarely. It's also a good way to look for crime-prevention. "Common" and "Very Common" tell you which crimes you should focus on to reduce the most accounts.

For example, LARCENY/THEFT was the most common crime with over 470,000 accounts but being only 1/37 different crime types. Loitering was one of the rarest crimes with only 2,402 accounts. If we were to try and reduce crime in San Francisco, we would focus on anti-theft rather than anti-loitering.

From the box plot (fig. 4), we can see the ranges and mean for each cluster. You can read it as

Average → 6 types of crime → represents 16.2% of the crimes in San Francisco
Lowest Crime Occurrence = 79.09 thousand accounts
Highest Crime Occurrence = 126.23 thousand accounts
Average Crime Occurrence = 104.79 thousand accounts

Predictive Model Analysis

Version 1

For Version 1 (mlp.py), my features were the pd district that handled the crime (ordinalDistrict), the day of the week the crime occurred (ordinalDOW), and the time the crime happened rounded to the nearest hour (Time_Minutes). The target was the type of crime (Category). I measured results using accuracy_score which tells you the overall percentage of correct predictions made by the model.

Figure 5

The model achieves a final accuracy of 24.3% with a validation accuracy of 24.2%. The loss is consistently decreasing, which indicates that the model is learning but not really improving it's predictions. The fact that the validation and training accuracy are very similar means that it's a pretty good fit but lacks learning power.

As a baseline, if we assume a random guess is 1/37 (since there are 37 crime categories), it would be around 2.7%. The model is doing better than a random guess but still performs poorly.

Version 2

For Version 2 (mlp2.py), my features were the same as Version 1, but I added a new feature (CrimeWeight) which weighs crimes to help the imbalance in crime counts. Rare crimes were mostly ignored in Version 1 which resulted in bad accuracy. I also added a precision, recall, and f1-score metric which gives a detailed breakdown of each crime.

Figure 6

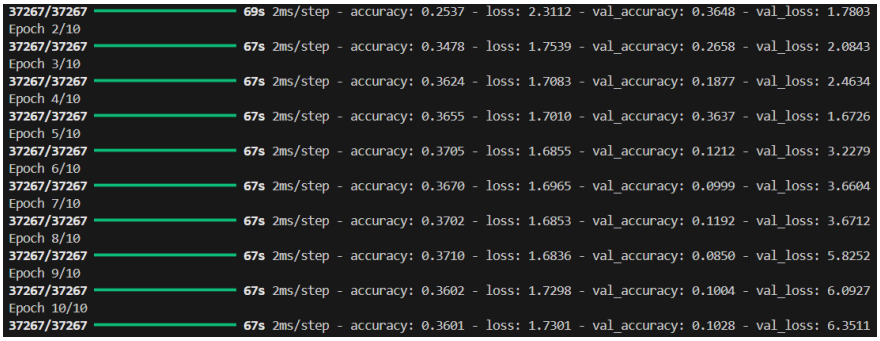


Figure 7

Crime Type	Precision	Recall	F1-Score	Support
ARSON	0.75	0.56	0.64	1179
ASSAULT	0	0	0	49862
BAD CHECKS	0	0	0	273
BRIBERY	0	0	0	234
BURGLARY	0	0	0	27396
DISORDERLY CONDUCT	0	0	0	3007
DRIVING UNDER THE INFLUENCE	0.88	0.46	0.6	1758
DRUG/NARCOTIC	0	0	0	35380
DRUNKENNESS	0.28	0.82	0.42	2928
EMBEZZLEMENT	0.32	0.8	0.45	906
EXTORTION	0.62	0.75	0.68	202
FORGERY/COUNTERFEITING	0	0	0	6871
FRAUD	0	0	0	12422
GAMBLING	0	0	0	92
KIDNAPPING	0.69	0.92	0.79	1280
LARCENY/THEFT	0	0	0	143819
LIQUOR LAWS	0	0	0	853
LOITERING	0	0	0	740
MISSING PERSON	0.51	0.98	0.67	13153
NON-CRIMINAL	0	0	0	71002
OTHER OFFENSES	0	0	0	90911
PORNOGRAPHY/OBSCENE MAT	0	0	0	16
PROSTITUTION	0	0	0	4927
RECOVERED VEHICLE	0	0	0	2523
ROBBERY	0.97	1	0.99	16461
SECONDARY CODES	0	0	0	6752
SEX OFFENSES, FORCIBLE	0.18	0.13	0.15	2640
SEX OFFENSES, NON FORCIBLE	0.41	1	0.58	14
STOLEN PROPERTY	0	0	0	3366
SUICIDE	0.04	0.06	0.04	375
SUSPICIOUS OCC	0.04	1	0.08	23738
TREA	0	0	0	5
TRESPASS	0	0	0	5766
VANDALISM	0	0	0	34207
VEHICLE THEFT	0	0	0	37585
WARRANTS	0	0	0	29925
WEAPON LAWS	0.24	1	0.39	6290

The model is experiencing severe overfitting and there is clearly an imbalance in classes. The training accuracy starts at 25% and peaks at around 37%, while the validation accuracy fluctuates significantly, starting at 36% and dropping to 10%. We can also see that while validation loss increases, training loss decreases. This tells us that the model is memorizing the

training data instead of learning generalizable patterns. The model performs well on the training set, but it fails to make predictions for unseen data. (fig. 6)

Robbery, kidnapping, and DUIs were predicted accurately; especially robbery with a precision of 0.97, recall of 1.00, and F1-score of 0.99. Most crimes (e.g. larceny, burglary, narcotics) have a precision/recall/f1 of 0. The model is effectively ignoring these crime categories and over-relying on select ones. The model is biased for more frequent categories and is struggling due to the lack of quality feature representation. (fig. 7)

Conclusion

Version 1 had a stable but poor learning performance. Training and validation accuracy hovered around 24% throughout the training. Lack of overfitting suggests that the model wasn't learning meaningful patterns between the crime categories. There weren't enough features. The model struggled to generalize and performed only marginally better than random guessing.

Version 2 introduced the CrimeWeight feature which increased training accuracy to around 37%. Validation accuracy dropped to 10%. This is a sign of overfitting. More crimes were correctly classified, but some crimes were ignored. The classification report helped us understand this.

Training the models was also very time consuming. On average, a pass through would take over a minute long. The scores could improve by increasing the # of epochs, but analyzing the data we got from 10 epochs tells us that it wouldn't have mattered since the accuracy was consistent.