**Quarter 1 Machine Learning Project: LAPD Victim Crime Classification Model**

**Team Member(s):**

**Connor Friedman**

**10/12/2024**

# 

# 

# 

# 

# 

# 

# 

# 

# **Part 1 – Statement/Project Goal**

The Los Angeles Police department records every arrest it makes, meticulously recording every detail about every crime that occurs within Los Angeles. The LAPD keeps this data stored and updated in a dataset available to the public, providing crucial information regarding crime statistics and profile descriptions to whoever desires to learn more about them. The information being tracked includes:

1. Date reported
2. Date ooc (of occurrence)
3. Time ooc
4. Area
5. Area name
6. Victim's Age
7. Victim’s Sex
8. Victim’s Race
9. The type of structure, vehicle, or location where the crime took place
10. Weapon Used
11. Weapon Description
12. Case Status

The following data gives us an insight on the changing crime rates within Los Angeles as its precision and detail can be very useful for analyzing trends in crime and victim profiling. Los Angeles' position as a major city, and an extremely popular one at that, makes it a hot spot for crime, with victims spanning across all sexes, ages, and races. Using the LAPD’s data, we can create a classification model that can accurately predict the category and severity of a given crime based on any attributes more correlated with either class. Therefore, for our project we can find which features are most influential in determining the result of any given crime and create a model that can reveal which type of crime a given person is susceptible towards within Los Angeles, allowing them to be prepared both physically and mentally ahead of time before visiting the city.

# **Part 2 – Description of Dataset**

Link to the dataset: https://catalog.data.gov/dataset/crime-data-from-2020-to-present

My project consists of a single dataset provided by the LA police department which is publicly known as Crime Data from 2020 to Present. The full downloaded dataset is nearly 250 megabytes large and contains every arrest made within LA from January 2020 to October 2024 (the month in which I downloaded the dataset). The dataset is flooded with attributes, consisting of nearly 28 unique values each indicating a separate piece of information about a single arrest. However, nearly half of this data is either irrelevant to the classification model or used as internal filing by the LAPD, containing no useful information for the general public. The following attributes that I decided to remove were:

DR\_NO

Rptd Dist No (reported district number)

Part 1-2

Crm Cd 1

Crm Cd 2

Crm Cd 3

Crm Cd 4

DR\_NO was an internal filing code used to sort the dataset, but is irrelevant to the classification model and is more accurately replaced by the date of occurrence of the crime. In addition to this, Date Reported, Reported District Number, and Part 1-2 have nothing to do with the crime itself, allowing for an easy removal from the dataset. Finally we have another batch of internal filing codes that provide no useful information on their own, also allowing themselves to be removed from the total dataset. The class of my model is Crm Cd Desc, a label that consists of the category and severity of a crime committed against the recorded victim (if there is one) and will be what we’re measuring the correlation of each feature with. Overall, the dataset used in our project consists of nearly 101 thousand instances and 19 attributes (including the class label).

# **Part 3 – Preprocessing**

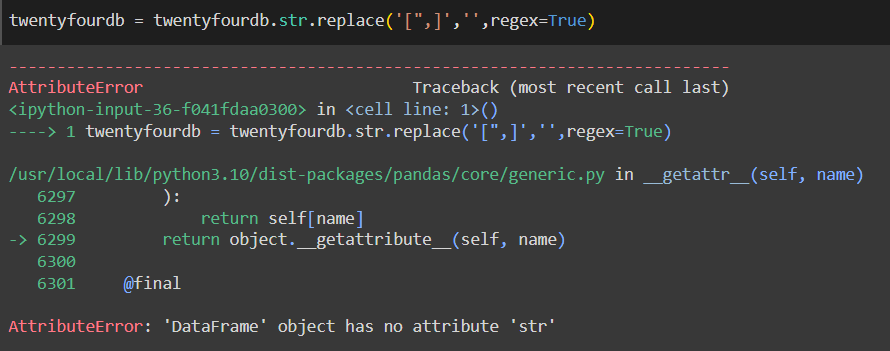
Pre-Processing was done in two Google Collab files with Python.

**Part 3.1 – Cutting the dataset to 2024**

The dataset that I decided to analyze with my Quarter One project is extremely insightful, but also extremely troublesome to work with. The preprocessing stage of my project took much longer than I had anticipated it would as a result of the complex steps I had to make to prepare my dataset to be loaded into WEKA. Firstly, in order to more effectively work with my chosen dataset, I had to drastically reduce any excess or unnecessary information found within the dataset, as its total file size came around to about 255 mb’s (which is much larger than necessary for the given project). To eliminate this problem I decided to simply erase any crime recorded before 2024, meaning that my project would no longer work with data from 2020 to 2023. Uploading my original dataset to google colab, I utilized pandas to transform it into a database for further manipulation. Using the LA Crime data database, called db in this case, I created yet another database by extracting each value with the year “2024” in its date value and finally downloading that new database as a csv file to my computer.

**Part 3.2 – Eliminating quotes and commas between them**

Originally I believed that this was all that was necessary to complete further analysis of the dataset within WEKA, but the formatting of the csv file made it ineligible to be explored from within WEKA for reasons that I didn’t quite understand at the time. After completing about 20 minutes of research I managed to discover the reason why importing the dataset was failing so much. Unfortunately, uploading a csv file to WEKA means that there can be no single quotation or double quotation characters within the entirety of the file. What makes this problem even more difficult is the fact that even if I decided to simply remove every ‘ and “ character from within the file, some instances have commas within the quotation, disrupting the structure of the file and making a simple replace all command impossible.

After consulting stackoverflow, I discovered a line of code (shown below) that would replace every unnecessary character throughout the dataset with an empty character, finally allowing the csv file to be read into WEKA for deeper analysis.



**Part 3.3 – Eliminating disguised missing data**

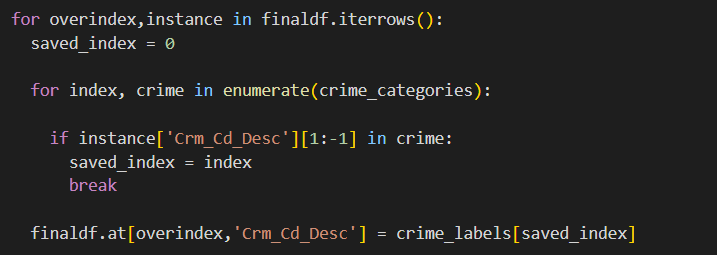
Crimes reported in the 2024 LAPD dataset do not always have victims, making them essentially useless when training the classification model. Getting rid of these values is a rather simple process, as we simply need to eliminate every instance that doesn’t contain information regarding the profile of the victim of the crime (meaning the committed crime didn’t have a victim).

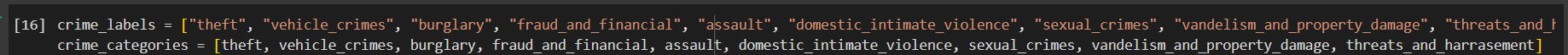


**Part 3.4 – Categorizing Types of Crime**

The class being used in our dataset is extremely varied in its values, with it containing approximately 140 unique values. In order to reduce the spread of data required to create an accurate model around such wide data, I categorized each crime into 1 of 9 categories:

1. **Vehicle\_crimes**
2. **Assault**
3. **Burglary**
4. **Theft**
5. **Vandalism\_and\_property\_damage (Spelled this wrong in the data ignore that**
6. **Domestic\_intimate\_violence**
7. **Fruad\_and\_financial**
8. **Threats\_and\_harrasment**
9. **Sexual\_crimes**





**Part 3.5 – Reducing size of dataset for Weka processing**

While having a large dataset can be helpful when maximizing the accuracy of your classification model, my dataset was literally too large to function properly. After hours and hours of preprocessing and exploring methods of shortening its size, I was only able to reduce the size of the dataset to around 60k values, forcing me to find some other method of reduction. In order to meet the size requirements of Weka, I was forced to reduce the size of my dataset to about a tenth of its original size (which is still a shocking amount of data, about 6k), which was just the right amount of data to create an effective classification model.



**Part 3.6 – Split final dataset into training and test dataset**

I did a 85%/15% split for the train and test datasets, where 15% of the dataset will be used to test the model’s predicted class after using 85% to train the model. This split resulted in 5,327 reported crimes in the training set and 940 reported crimes in the test set.

# **Part 4 – Attribute Selection Algorithms & Model Classifiers Used**

After a significant amount of preprocessing, my dataset was left with around 20 attributes. Before any processing could be performed however, I first needed to set the Crm\_Code\_Desc as the class attribute, allowing both attribute selection algorithms and model classifiers to be used. Below are the attributes that can be found in the attached final1029crime.csv, testing\_crime.csv, and training\_crime.csv:

**Class Attribute:** Category of crime (known as Crm\_Code\_Desc within the dataset itself)

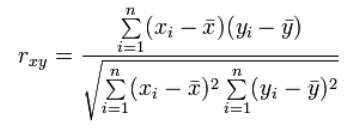
**Features:** DATE\_OCC, TIME\_OCC, AREA, AREA\_NAME, Crm\_Cd, Mocodes, Vict\_Age, Vict\_Sex, Vict\_Descent, Premis\_Cd, Premis\_Desc, Weapon\_Used\_Cd, Weapon\_Desc, Status, Status\_Desc, LOCATION, Cross\_Street, LAT, LON

One thing to note before posting the results of my attribute selection and classification is the fact that the descriptions of a select amount of attributes are directly correlated to its associated code, meaning that attributes such as Premis\_Cd and Premis\_Desc will pretty much be directly correlated with one another (with a couple of differences).

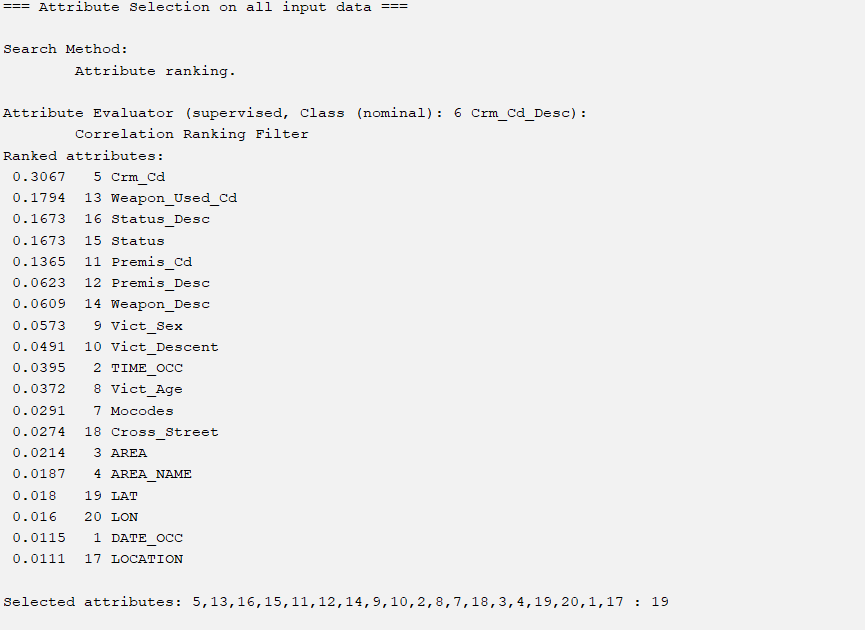
**Part 4.1 Attribute Selection Algorithms:**

1. **CorrelationAttributeEval**

The CorrelationAttributeEval attribute selection method can be used to measure how much impact each attribute has on the outcome of the class attribute, which can be measured using the Pearson's correlation formula:

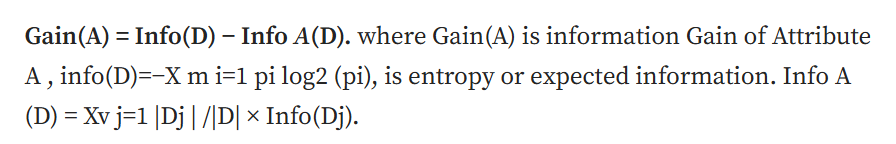


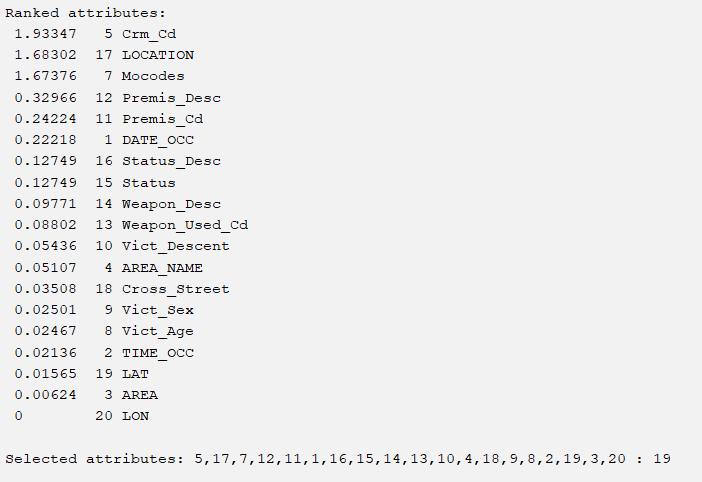
I utilized Weka for this approach because its integrated attribute selection made it extremely easy to calculate and compare the correlation of each attribute shown below.



For the threshold of the correlation, I decided to use every attribute with a correlation of 0.0491 or above, as I felt that Vict\_Sex and Vict\_Descent are some of the two most important factors when it comes to determining how safe one is in any given part of LA, regardless of how correlated the Pearson correlation formula deems them as.

1. **InfoGainAttributeEval**

Similar to the correlation evaluator, the InfoGainAttribute evaluator utilizes a complex formula to measure the worth of an attribute by measuring its information gain with respect to the class attribute. Information gain is calculated as the reduction in entropy that results from partitioning the data based on a given attribute.

The following is the calculated info gain of my dataset (in WEKA).

When creating a dataset based off of the results of the InfoGain evaluator, I decided to use every attribute with an info gain of 0.12749 or above (not including status description).

1. **OneRAttributeEval**

The OneR attribute evaluator is a simple yet effective attribute selection method. OneR is a selection algorithm that generates one ‘rule’ for each predictor on the data, then selects the rule with the smallest total error as its ‘one rule’. Despite the supposed basicness of the evaluator, OneR has been shown to have extreme accuracies that cut just short of state-of-the-art classification algorithms, making it extremely effective when looking at our data. OneR uses the following Pseudocode:

*For each attribute*

*For each value of the attribute*

*count frequency of each class*

*find the most frequent class*

*make rule: assign that class to this attribute-value*

*Compute the error rate of the rules (of this attribute)*

*Choose the rules with the smallest error rate*

The following data is WEKAs OneR algorithm being run on my dataset:



Utilizing a threshold of 56.5592 I am given 10 attributes to work with in my classification algorithm.

1. **ReliefFAttributeEval**

I used WEKA for this approach. This attribute selection algorithm evaluates the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class. To put this in layman's terms, for each training instance a vector of attribute values and the class value, the ReliefF attribute selector produces the vector W of estimations of the qualities of attributes. The core idea of this algorithm is to estimate the quality of an attribute by how well the attribute can distinguish between two instances that are near to one another. The following is the pseudo code of the selection algorithm:

*1.set all weights W[A] := 0.0;*

*2. for i := 1 to m do begin*

*3. randomly select an instance Rᵢ;*

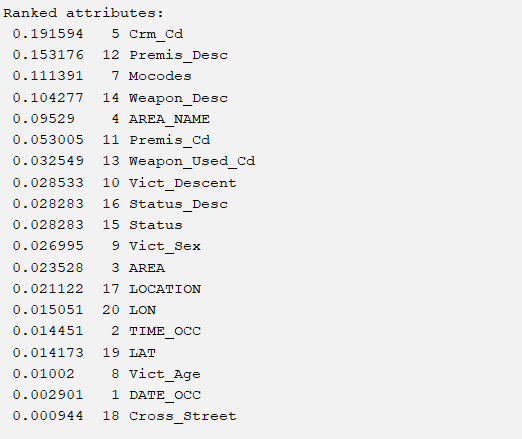
*4. find nearest hit H and nearest miss M;*

*5. for A := 1 to a do*

*6. W[A] := W[A]-diff(A,Rᵢ,H)/m + diff(A,Rᵢ,M)/m;*

*7. End;*

The following data is the ReliefF selection algorithm being run on the aforementioned dataset:



The threshold of attributes I used for Relief’s associated classifiers is 0.026995.

1. **Custom Attribute Selection**

Rather than follow a strict formula, algorithm, or pattern, I decided to create a final selection of attributes that simply made the most sense in the context of the dataset, outwardly apparent factors that would most likely contribute to a crime of a certain category being committed against you. The following is the list of attributes I decided properly fit this description:

**Time\_OCC**

**Date\_OCC**

**Vict\_Age**

**Vict\_Sex**

**Vict\_Descent**

**Area**

**Area\_Name**

I chose the following attributes because when watching out for crime within LA, the following factors are perceived by the public as the most important when attempting to avoid crime within a given area. For example, within a sketchy downtown part of LA, a 19 year old asian woman walking alone at night on a weekend is oftentimes most susceptible to a robbery, a situation clearly described by the attributes I have chosen.

**Part 4.2 Classifier Models**

**J48 -** A decision tree algorithm that utilizes entropy to both gather information and construct decision trees. The algorithm calculates the information gain for each attribute, and attributes with the highest yield of information are selected to be nodes for the tree. This process will repeat from node to node until no more information can be obtained from the dataset, completing the classification model.

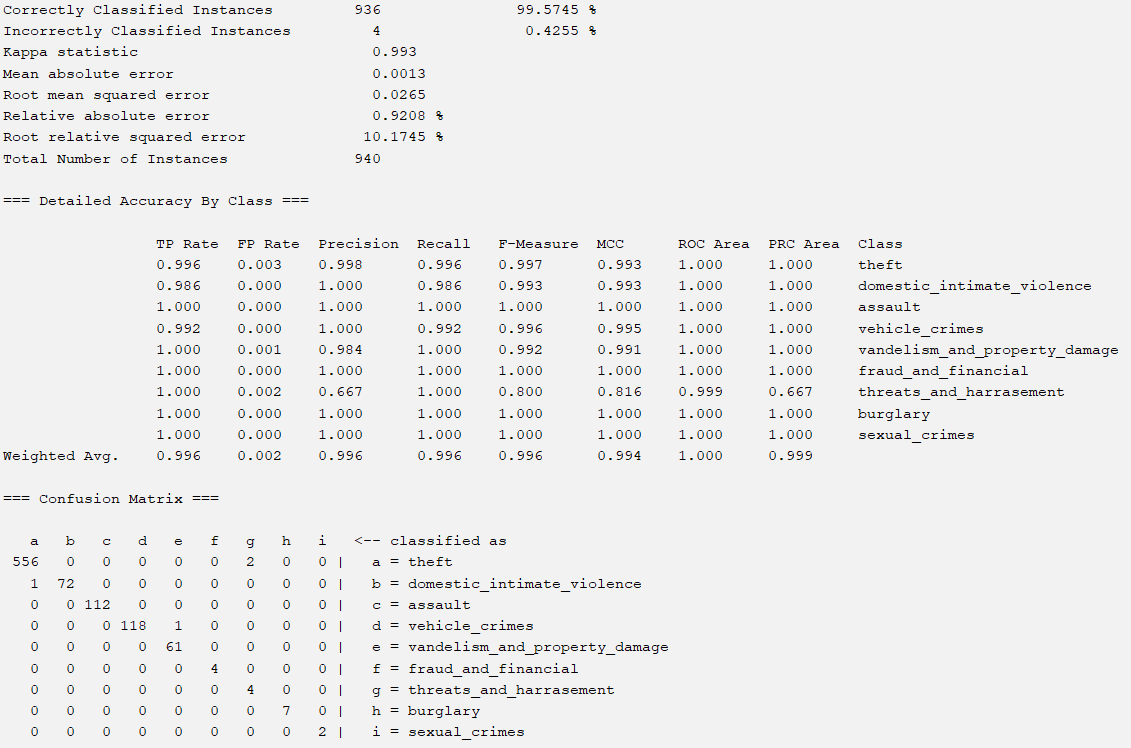
**Naive Bayes -** A probabilistic classification model that operates under the principles of the Bayes theorem. This model assumes that all attributes in the class are independent of each other given a class label, calculating the probability of each class according to a set of input attributes, then predicts the class with the highest resulting probability.

**Random Forest -** Utilizing a large set of independent and evenly weighted classification trees, Random Forest utilizes the output of each instantiated tree to determine a classification for the entirety of the model.

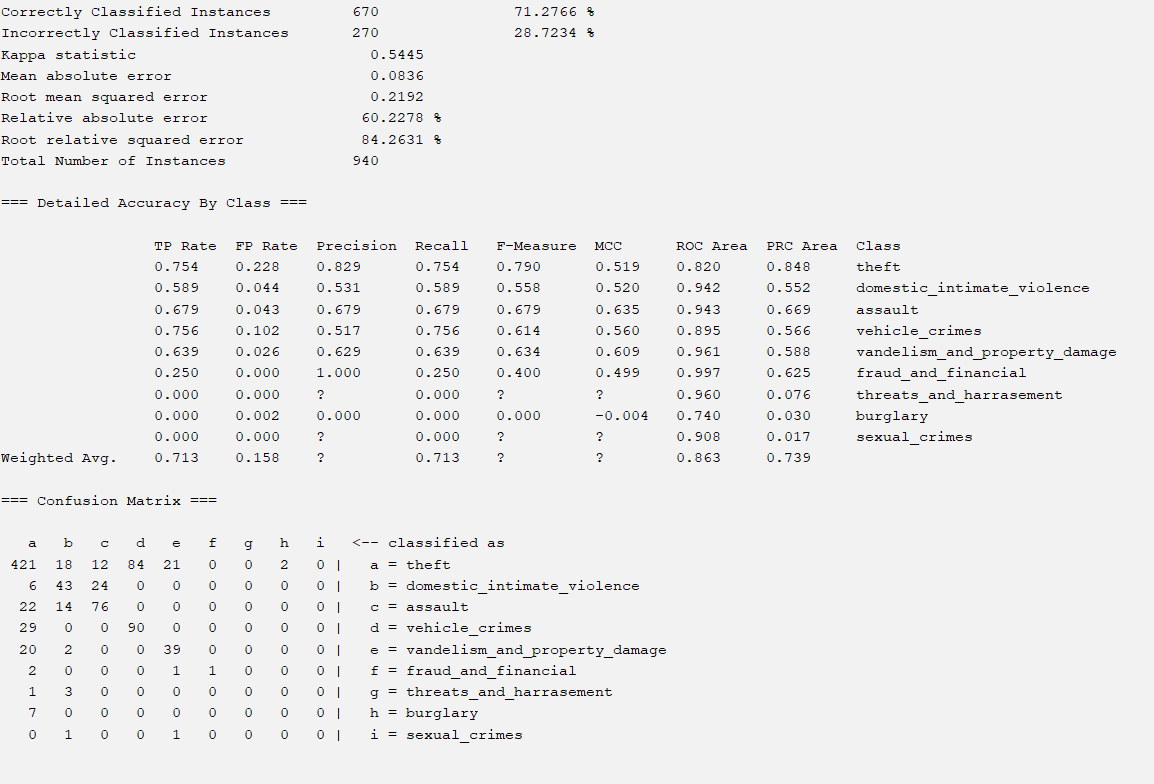
**OneR -** Just like its attribute selection counterpart, The OneR classification model creates a single rule based on the feature that best classifies the data, selecting the feature with the fewest classification errors. It then assigns classes based on the most frequent class for each value of that feature.

**Part 4.31 Classifier Results - Correlation Attribute Evaluation**

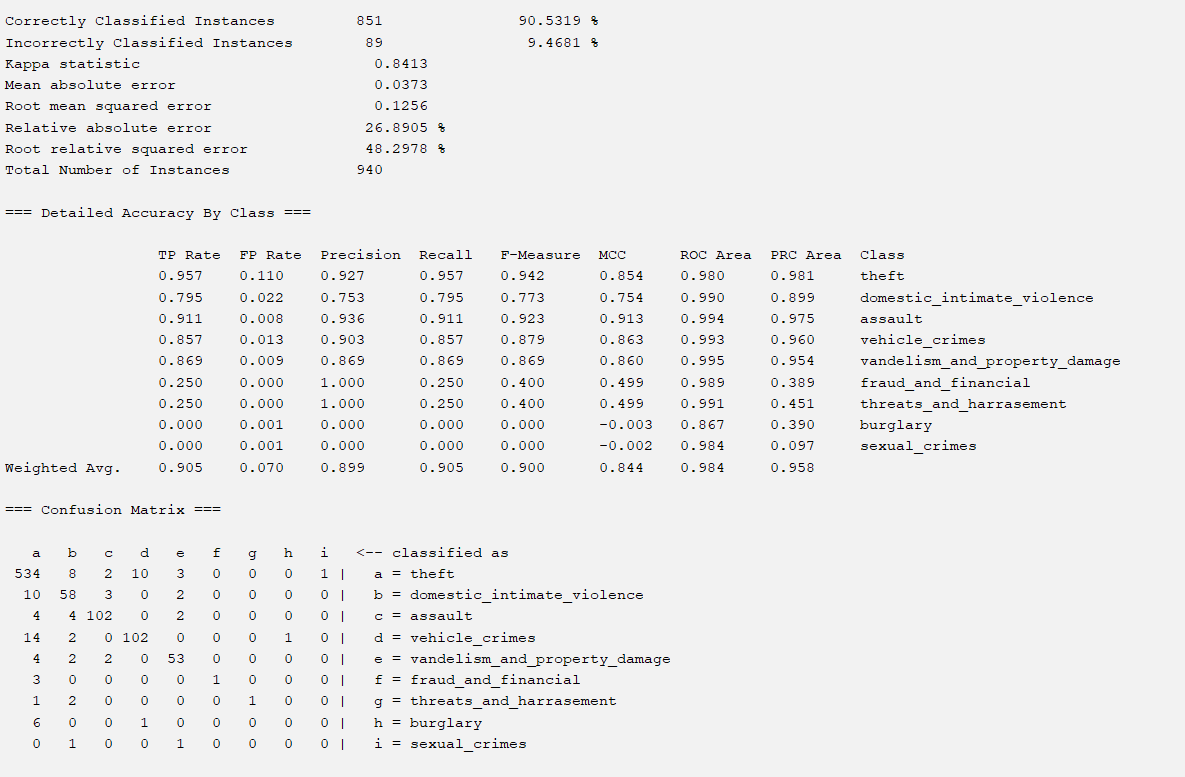
**CorrelationAttributeEval x J48**

****

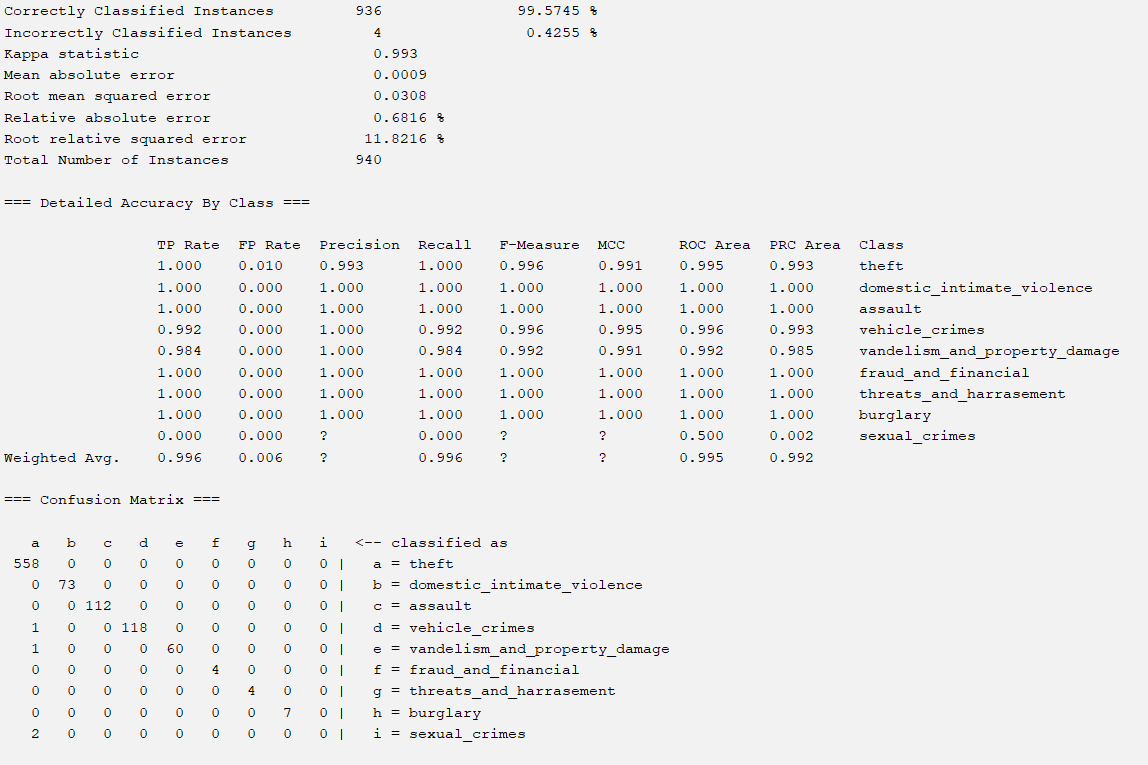
**CorrelationAttributeEval x Naive Bayes**

****

**CorrelationAttributeEval x Random Forest**

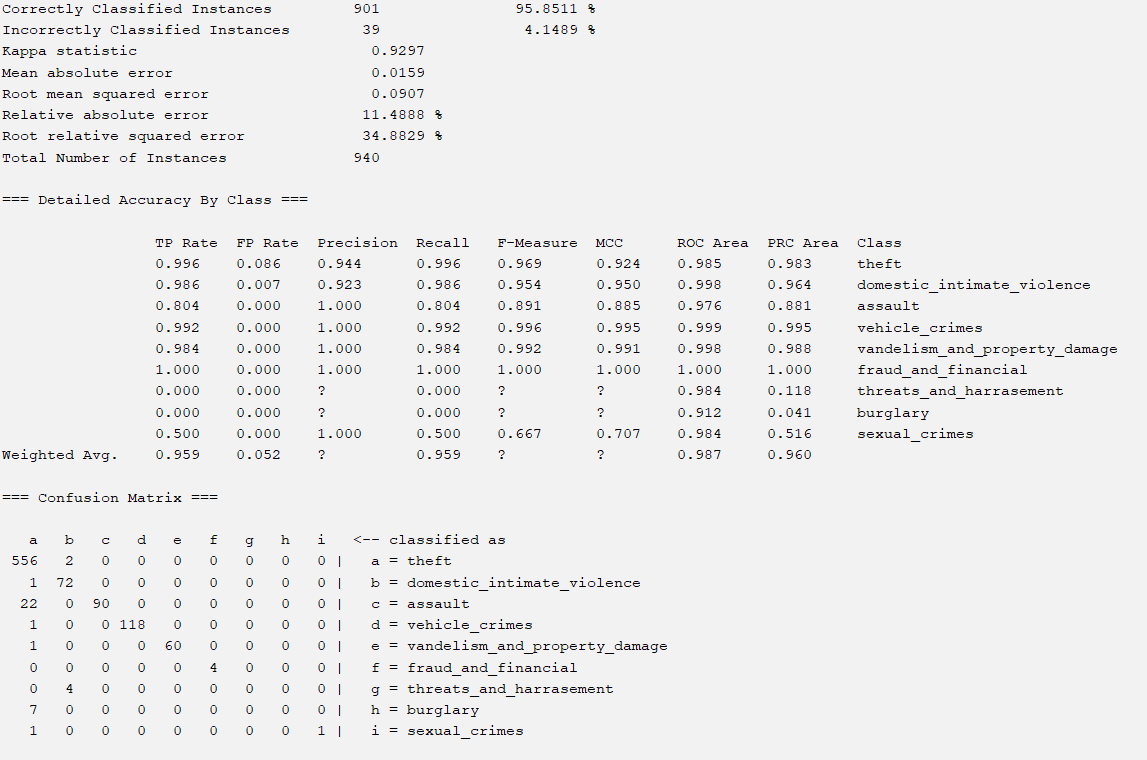
****

**CorrelationAttributeEval x OneR (classifier)**

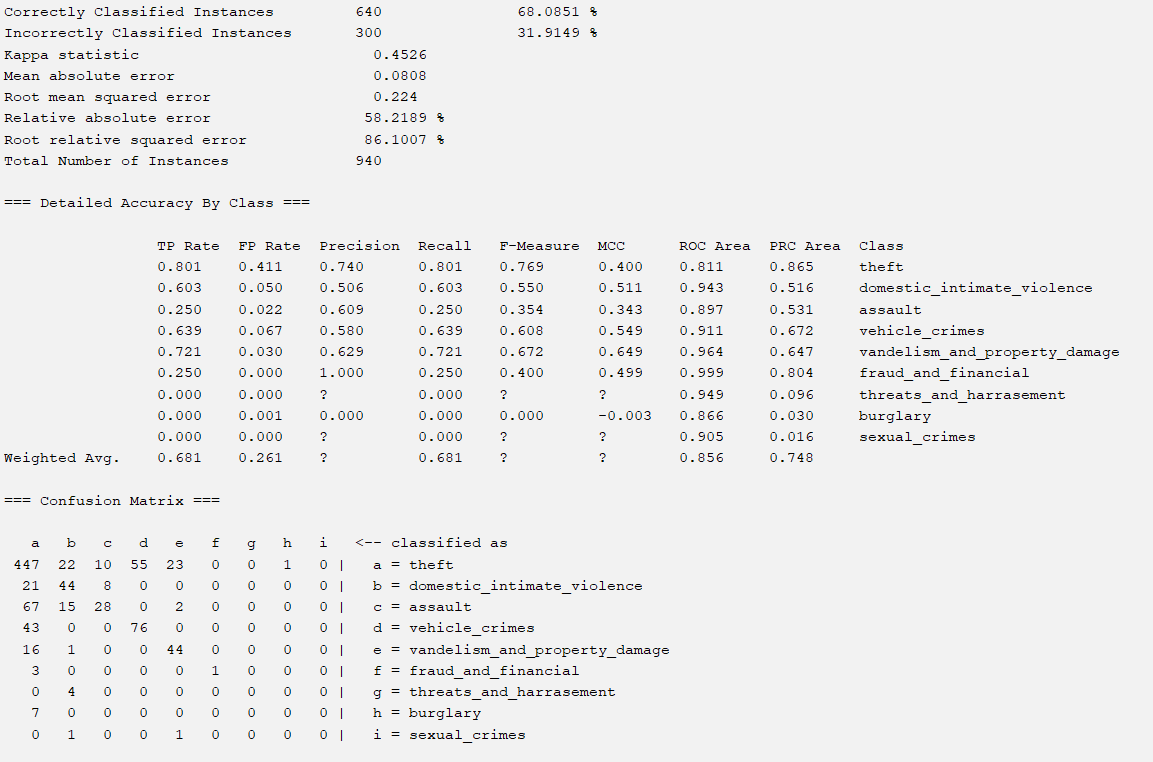
****

**Part 4.32 Classifier Results - Info Gain Attribute Evaluation**

**InfoGainAttributeEval x J48**

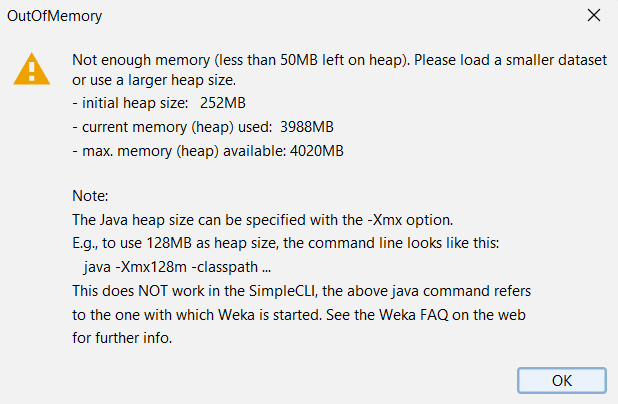
****

**InfoGainAttributeEval x Naive Bayes**

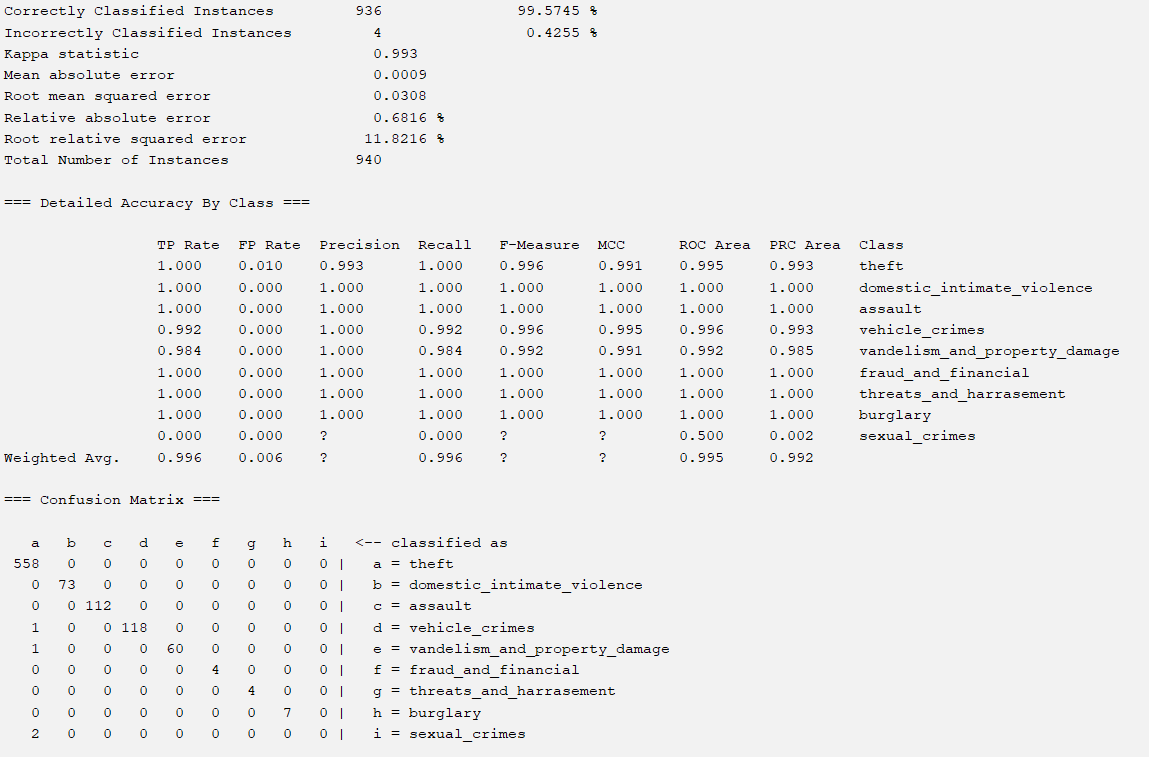
****

**InfoGainAttributeEval x Random Forest**

Surprisingly, this classification model actually wasn’t able to be created, as whenever I ran the classification model within WEKA I would get an error that forced me to restart the app.

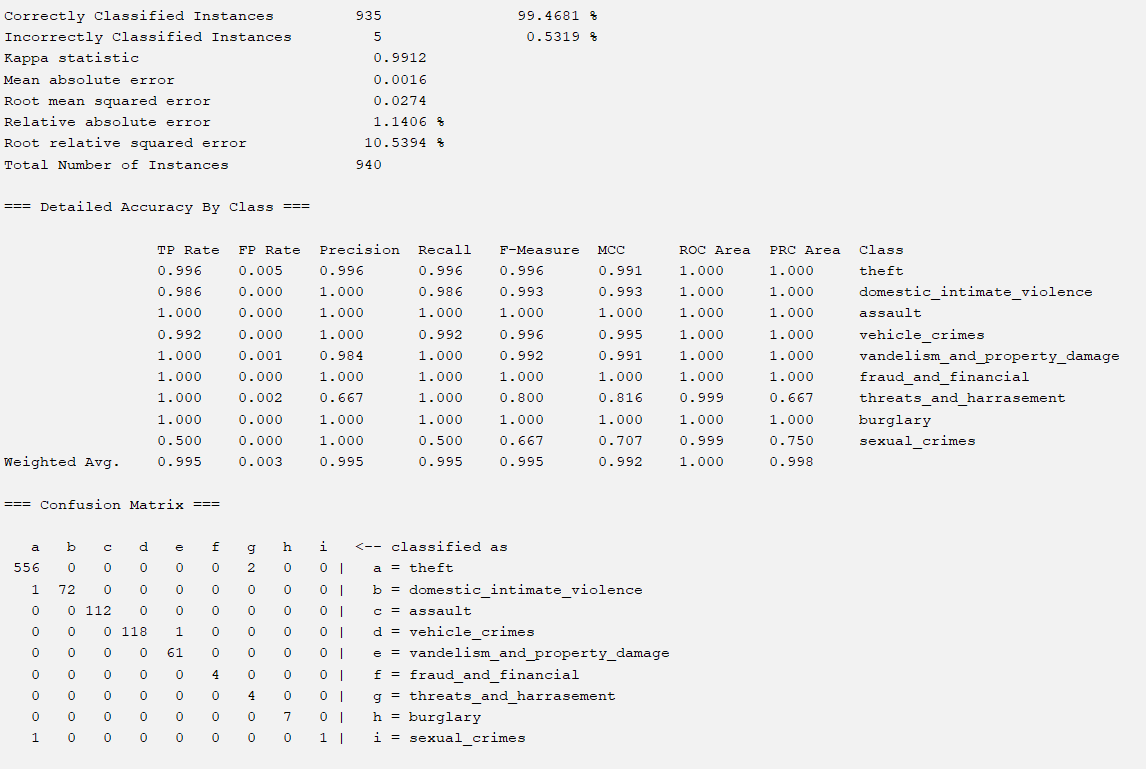
****

**InfoGainAttributeEval x OneR (classifier)**

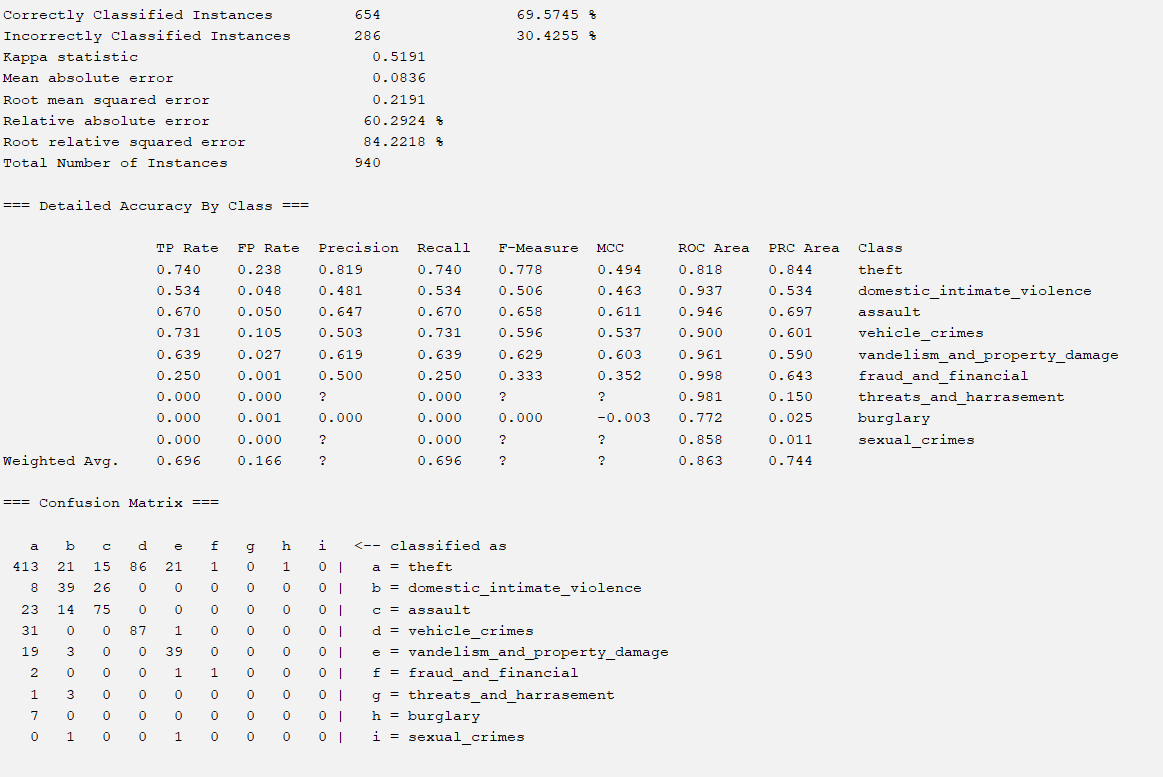
****

**Part 4.33 Classifier Results - One R Attribute Evaluation**

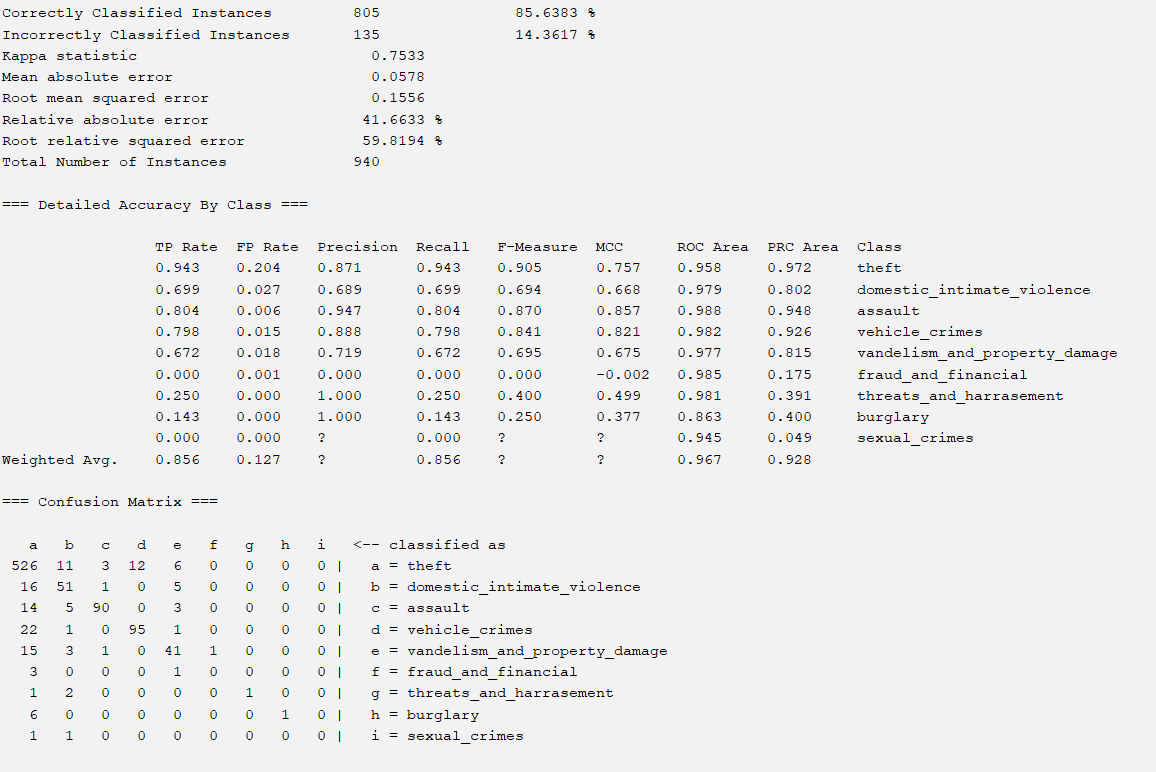
**OneR x J48**

****

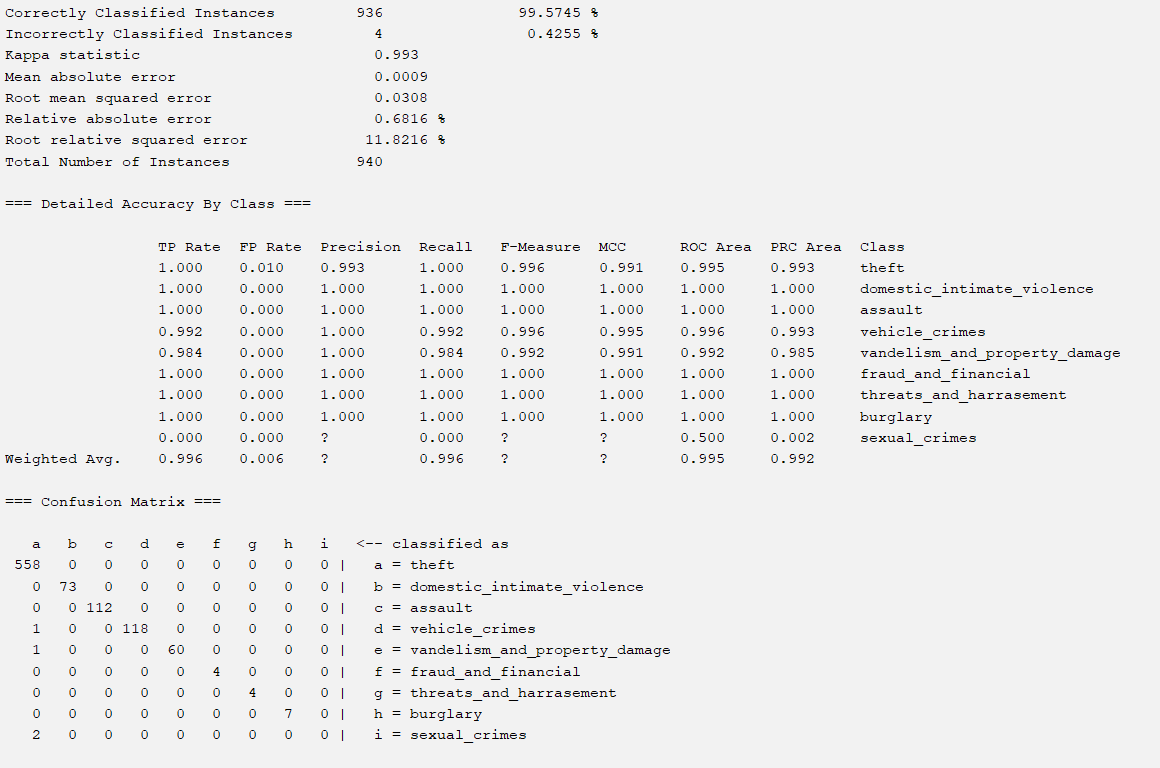
**OneR x Naive Bayes**

****

**OneR x Random Forest**

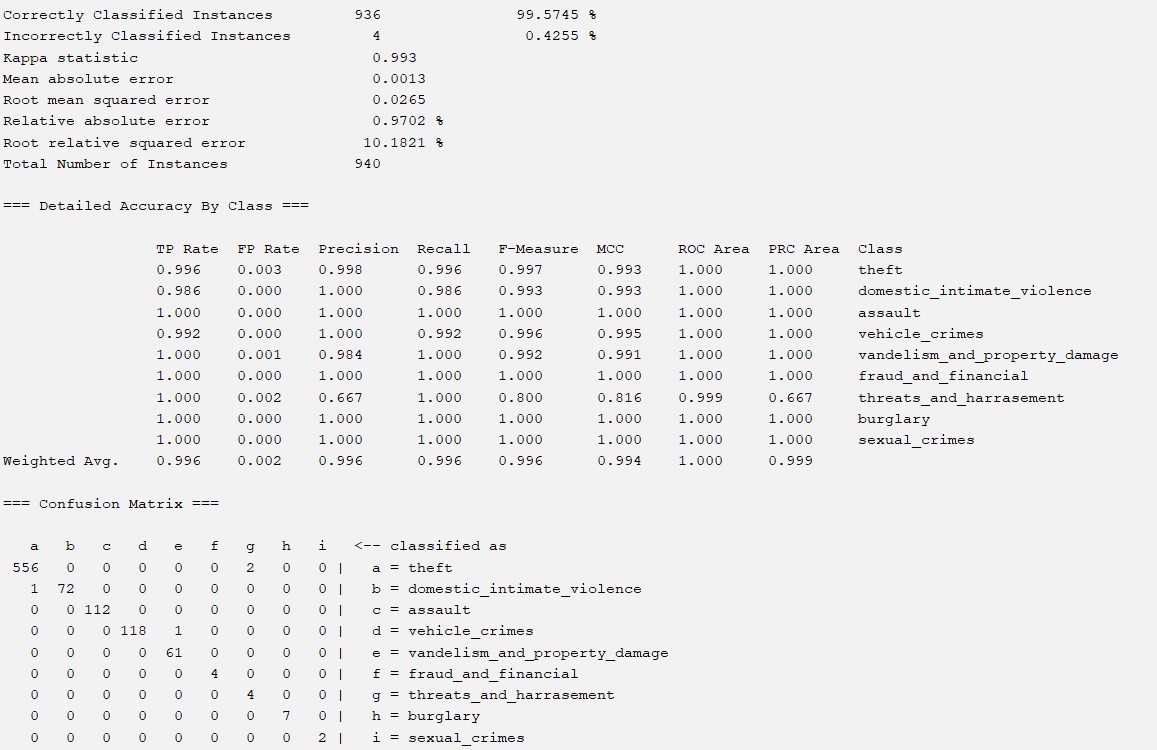
****

**OneR x OneR (classifier)**

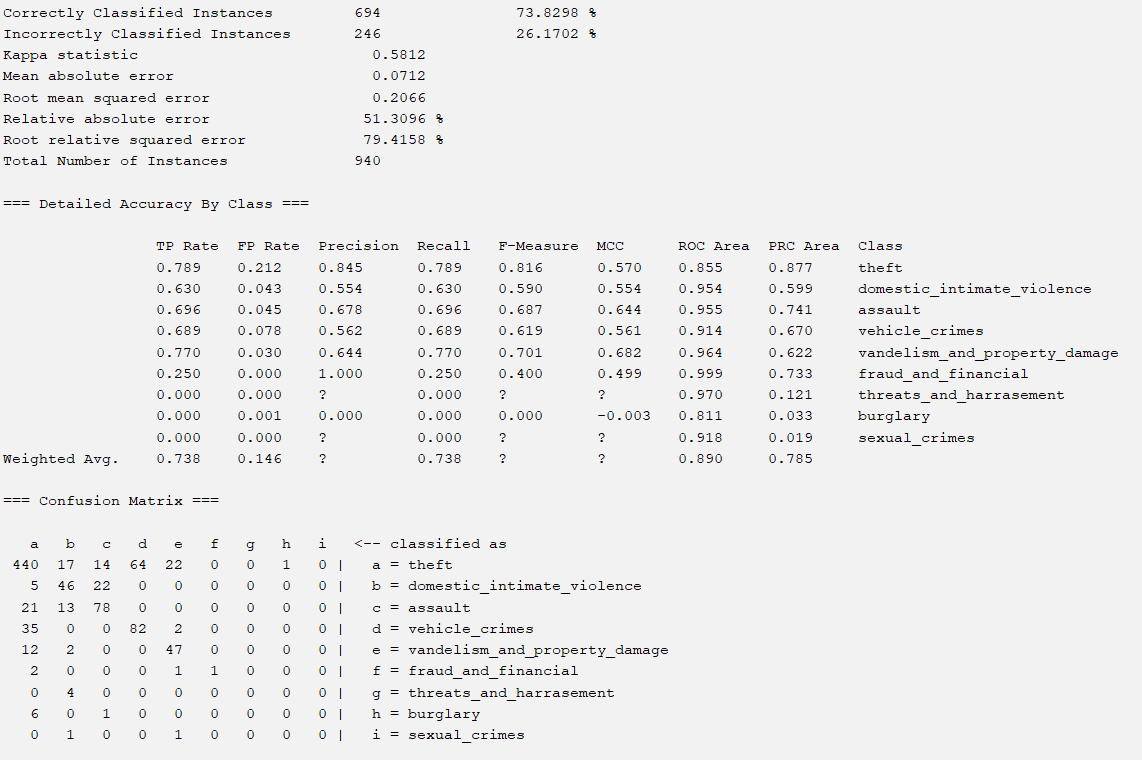
****

**Part 4.34 Classifier Results - Relief F Attribute Evaluator**

**ReliefF x J48**

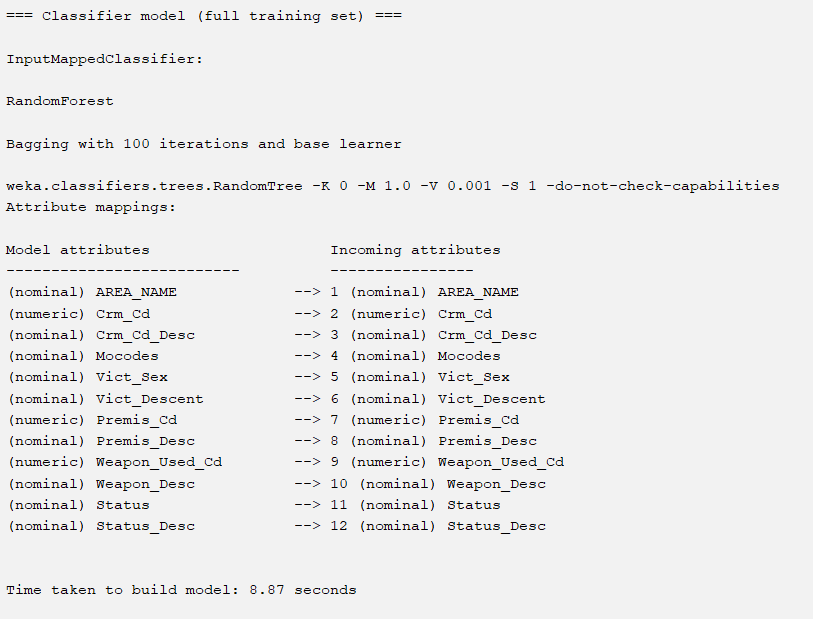
****

**ReliefF x Naive Bayes**

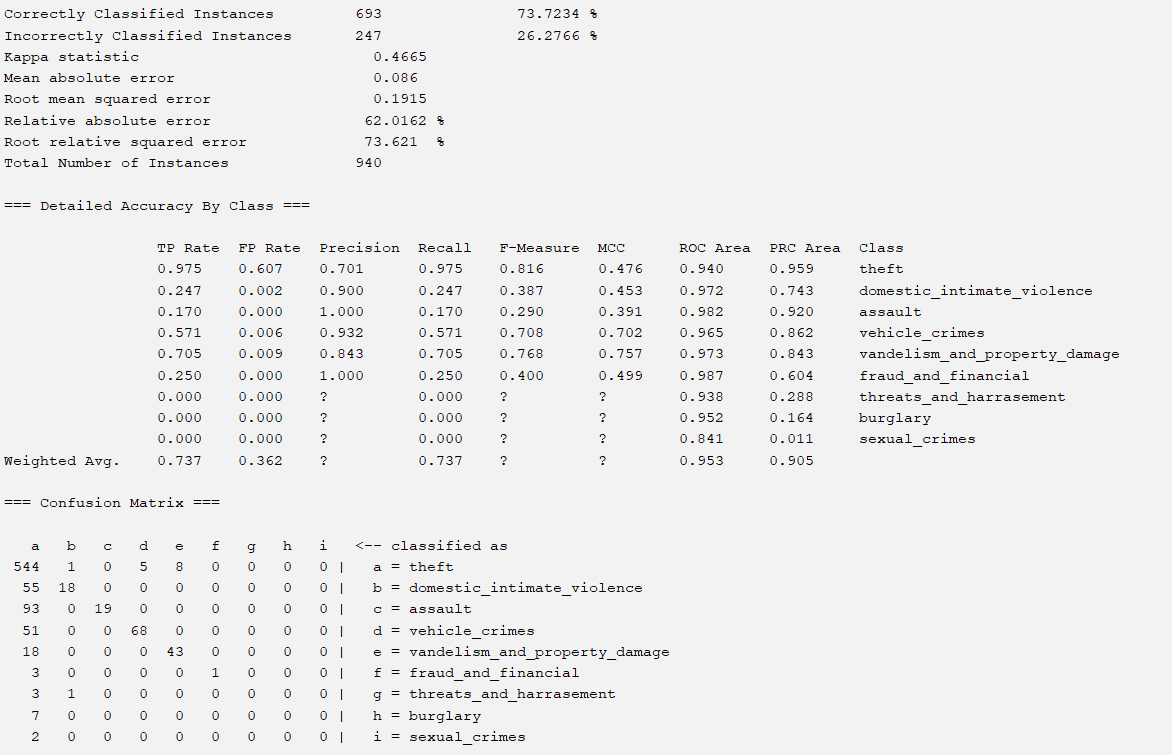
****

**ReliefF x Random Forest**

Weka ran into an error when I tried to train my ReliefF dataset using Random Forest, displaying this cautionary sign prior to building the model, which didn’t put any data regarding the model in the provided screen, leaving me with no information and much less space on my computer.

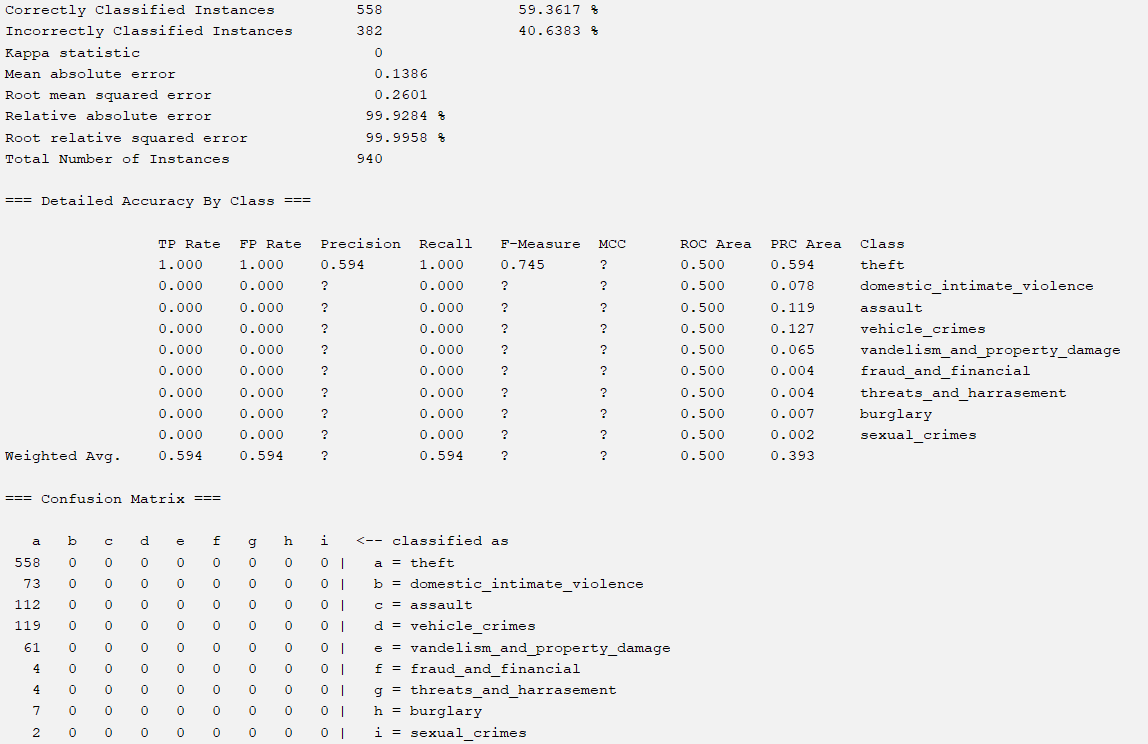
****

**ReliefF x OneR**

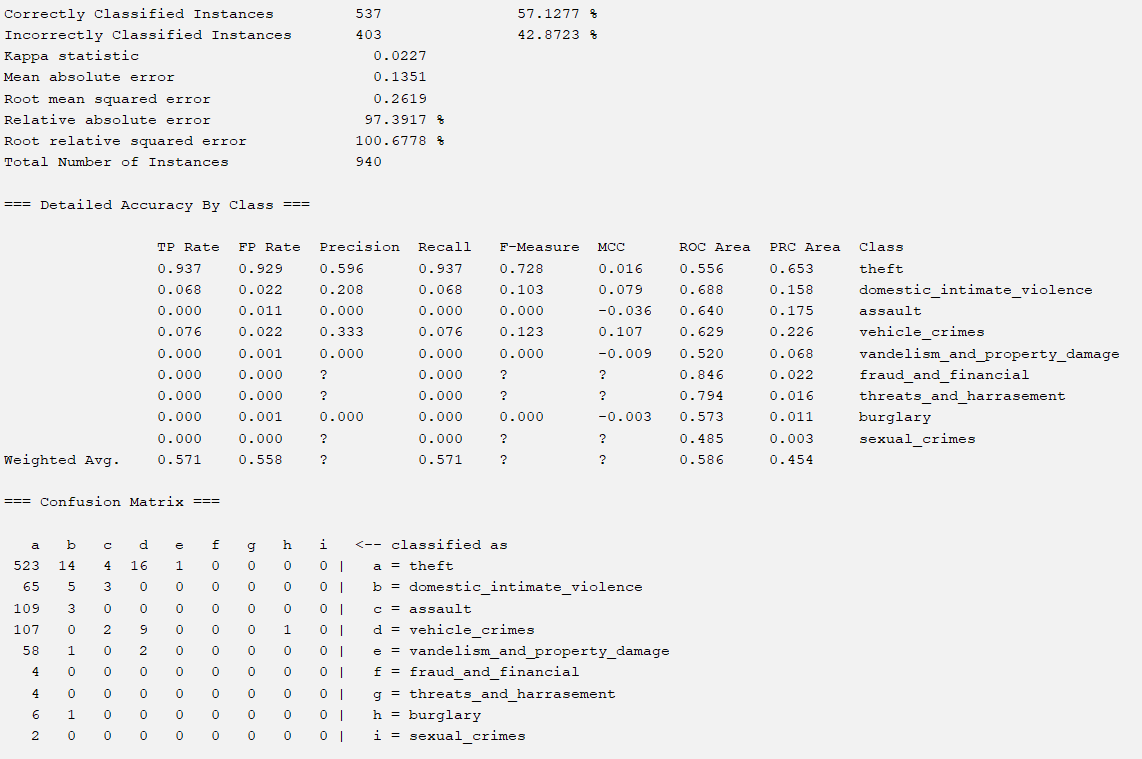
****

**Part 4.35 Classifier Results - Intuition**

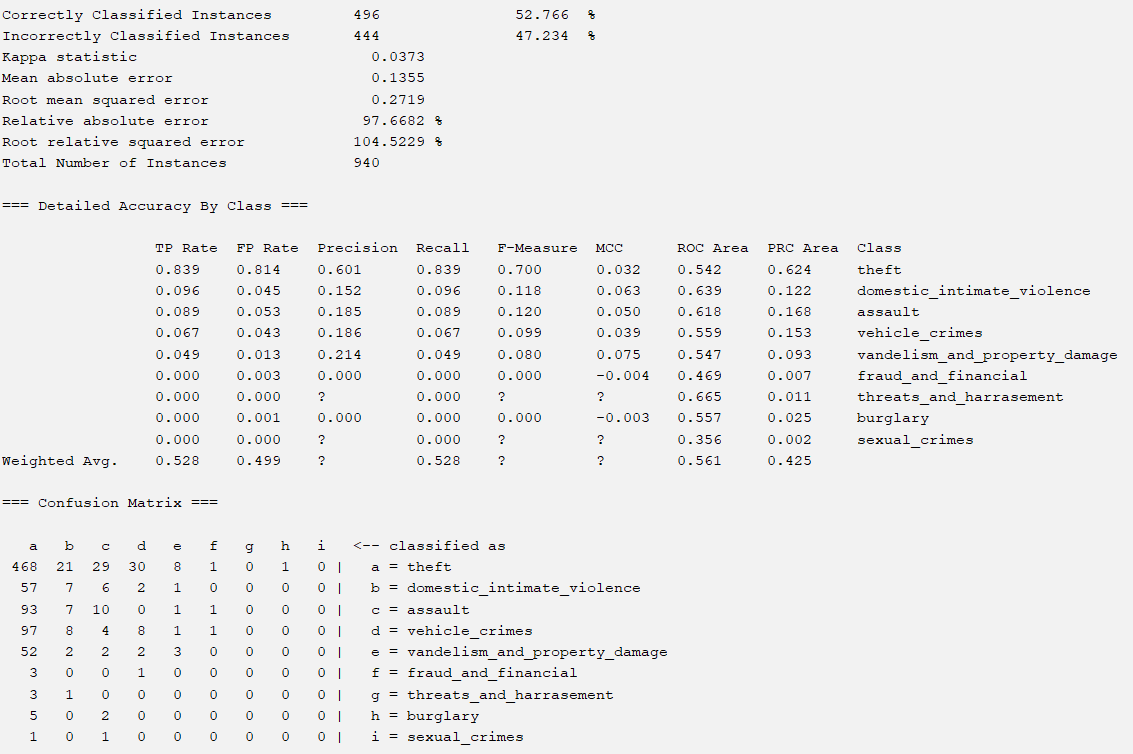
**Intuition x J48**

****

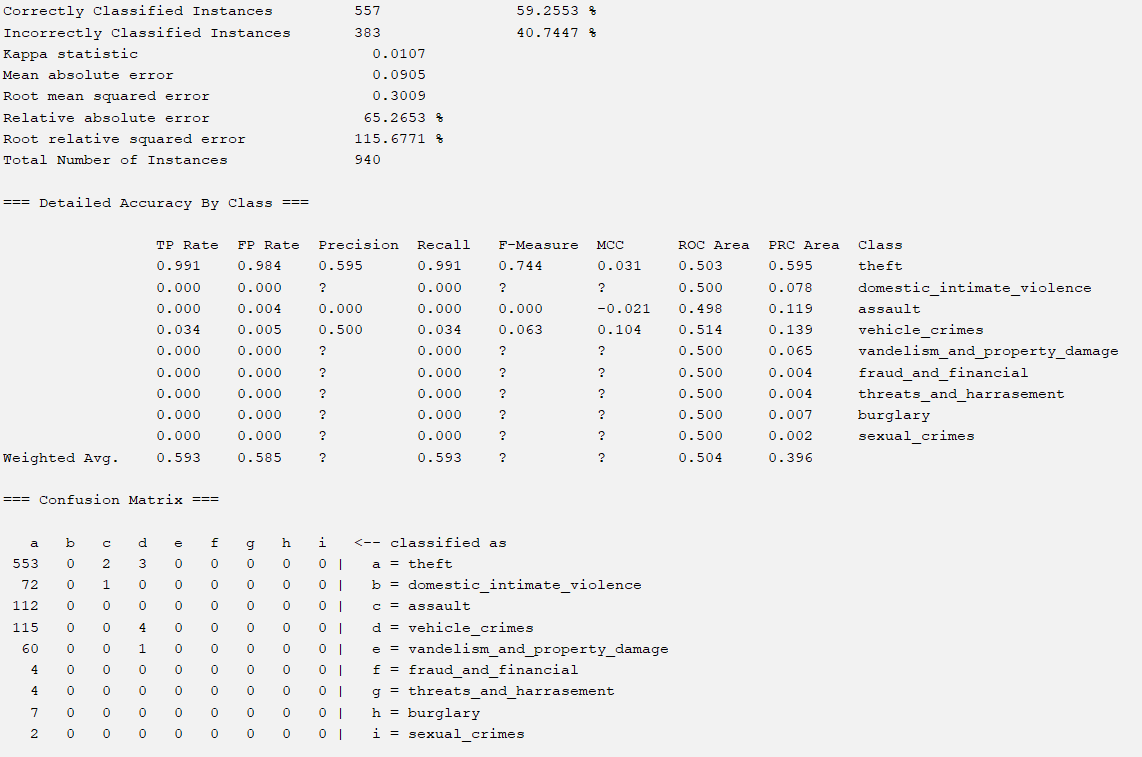
**Intuition x Naive Bayes**

****

**Intuition x Random Forest**

****

**Intuition x OneR**

****

**Part 5.1 General Analysis**

After running 20 models (with 2 of those models being unable to be processed through WEKA), I found that the best performing model(s) in terms of accuracy per attribute selection algorithm were:

1. **CorrelationAttributeEval x J48 and CorrelationAttributeEval x OneR (classifier)**
2. **InfoGainAttributeEval x OneR (classifier)**
3. **OneR x J48**
4. **ReliefF x OneR**
5. **Intuition x J48**

Overall the models I created using the attribute selection algorithms and different classifiers resulted in pretty high accuracies all around the board. When ignoring the results of the Intuitive classification model, I observed that the lowest accuracy found amongst the created models was 68.0581, while the highest accuracies found across my models were found between 99 and 100, with OneR and J48 classification models producing the most accurate classification models.

Strangely enough, I found that the method of attribute selection didn’t have much of an impact on the accuracy of the classification models, as nearly every method of attribute selection (with the exception of my intuitive models) produced 1 or 2 models with an accuracy between 99 and 100, deeming my project as a success for the most part. What I found by far the most interesting about my model is that the factors I had believed to be significant contributors to what crime you are most susceptible to ended up correlating very little with the category of crime, with other factors such as the description of the premises, the status of the case, and the weapon used in the case being much more important, resulting in my intuitive classification models having low accuracies.

**Part 5.2 TP/FP Rate**

While accuracy is oftentimes a fantastic predictor of how well a classification model functions, considering it as the only factor in this decision is simply unwise. While most of my models are indeed accurate, a great deal of them also struggle with having a high rate of false positives, most likely due to the imbalance of data between theft and the remaining categories of crime (as theft itself makes up essentially half of the dataset). The five best performing models by TPR are:

1. **ReliefF x J48**
2. **OneR x OneR (classifier)**
3. **OneR x J48**
4. **InfoGainAttributeEval x OneR (classifier)**
5. **CorrelationAttributeEval x OneR (classifier)**

To measure the performance of these models via TPR and FPR, I found that using the Theft attribute as an indicator of effectiveness would be the best way of measuring the model, as it is almost always the attribute with the greatest amount of false positives in nearly every attribute selection algorithm and classification model. Of the best performing models, I was able to achieve a minimum TPR of about 0.996, and a minimum theft FPR of around 0.0005, with both the OneR classification model and the J48 classification model performing the best using this form of analysis.

**Part 5.3 Area Under ROC Curve**

Because I determined that FP and TP rates of theft can serve as an effective indicator for the success of a given classification model, the ROC curve of theft should also serve as a satisfactory method of measuring my model’s performances. The top 5 models by the area under the curve metric are:

1. **ReliefF x J48**
2. **OneR x OneR (classifier)**
3. **OneR x J48**
4. **InfoGainAttributeEval x OneR (classifier)**
5. **CorrelationAttributeEval x J48**

With the lowest AUC of these models being 0.995, and the highest being an exact 1, with ReliefF x J48, OneR x J48, and CorrelationAttributeEval x J48 all performing the best.

**Part 5.3 Best Overall Classification Model?**

After examining the TPR, FPR, AUC, and Accuracy of each and every classification model, I determined that the best performing model of the ones I created for this project was CorrelationAttributeEval x J48, which had an accuracy of 99.5745, TPR of 0.996, FPR of 0.003, and AUC of 1.

**Part 6 Conclusion**

Overall this project served as a gateway to the field of machine learning, giving me first hand experience with the workflow of a supervised learning model. This project stressed every piece of information I knew about machine learning both in practice and in theory, requiring me to fully acknowledge how each piece of my machine learning model functions and how that then feeds into the rest of the project. It made me think critically about an extremely challenging dataset to manage, create effective solutions to issues with the structure of my dataset, and carefully analyze each and every statistic regarding my project in order to get the most accurate results.

**Part 7 How to Recreate**

1. First download the dataset from the official .gov website.
2. Go into the first python collab environment, put that file into the code and let the python code reduce the size of the dataset. (note, the regex code in my code is unable to account for single quote characters, so make sure you open the file in vscode and manually remove them with the control f and replace all functions.
3. Next, use the second collab notebook to finish preprocessing of the data, downloading the total data set and training and testing datasets as csv files.
4. Next go through the attribute selection algorithms with the larger dataset, taking note of what attributes agreed with each selection process.
5. Finally, create new datasets for each attribute selection process and run them in the classifier with each aforementioned classification algorithm.