**How Does Data Help When Looking into Crimes?**

Crime is inevitable. It happens everywhere and occurs in ways most people don’t even think about. When you consider the process of a crime occurring, it being reported, an investigation being started and in hopefully in the end, there being a suspect, there are important aspects that go unnoticed. If you’ve ever watched a show like Castle, NCIS, Dexter, Law, and Order, etc. they use tactics like phone records, fingerprints, car information, genetic history to help solve crimes. What most people fail to grasp is that all that information, is data. It’s pulled from huge data bases and exploited.

Let’s look at some famous cases:

BTK killer: Serial killer who tortured and killed at least 10 people in the span of 30 years before he confessed in 2005. He came up with the name BTK because he bound, tortured, and killed his victims (Tikkanen). He would send police cryptic messages and authorities were unable to trace anything from the letters. That was until he sent the police a floppy disc containing a Microsoft Word Doc on it where forensic experts were able to track the metadata within the disk leading to an identity (“3 Famous Cases Solved Through Digital Forensics”).

Craigslist killer: In April 2009, there was a woman murdered in her hotel room in Boston. There was another case reported documenting an assault of a woman robbed at gunpoint. Authorities were able to trace emails and found a similar individual from Craigslist on both victim’s emails. It turned out that both victims advertised their services on Craigslist and had an appointment with this individual during the time the crimes took place. Authorities went a step further after uncovering the commonalities between victims to uncover an IP address which led them to a 23-year-old medical student (“3 Famous Cases Solved Through Digital Forensics”).

Larry J. Thomas Vs. State of Indiana: In 2016, Larry J. Thomas was found guilty of an attempted robbery that led to murder. There was an eyewitness that established he was at the scene, but it was his Facebook feed that exposed him. During the investigation, authorities investigated his social media, specifically Facebook, and saw he was using the handle name “Slaughtaboi Larro” and had posted pictures holding an assault rifle. The ammo at the scene matched the rifle and in the pictures the suspect was wearing a bracelet that was found at the scene. This led to an arrest and imprisonment (“3 Famous Cases Solved Through Digital Forensics”).

Data can also be used in criminal profiling. Criminal Profiling is the process creating a detailed description of the suspect to aid in capturing the criminal. The description is created from clues, and behavior connections between the evidence. Trying to create criminal profiles by hand, would be extremely inefficient, would need many resources and there is likely to be more errors. Data is captured by taking the behaviors of non-criminals and comparing it to criminals. Then it can be taken to the next step to analyze patterns to attempt to capture and restrain criminals from continuing their behaviors.

Before Data Analytics, investigators had to sort through the data by hand. Now, there’s technology to do it for them. Big data analytics software can sort through extensive amounts of data rapidly and spot trends. For example, Communities in Colorado installed technology to read license plates for any vehicle that enters or leaves those communities. “A Denver police chief who’s in favor of using the technology mentioned 70% of all crimes committed involve vehicles.” (Matthews). Officers in Wyoming MI did something similar where they set up 12 cameras to capture license plates. They then use the information to solve crimes involving automobiles and illegal activities regarding specific vehicles. While it might seem that older cars don’t carry data, they do through their license plate. Today’s advanced cars keep data on preferences, routs, can operate features by voice and link their phones to the car. This provides police with more data if certain cars are involved in criminal activity (Flynn). “Lam Nguyen, director of the Defense Cyber Crime Center, said: "I'm sure everyone is aware of how much forensic data is on the phone. What people don't realize is a lot of that is being transmitted to a car just because you register the phone with the car."”. Also, many vehicle infotainment systems don’t lock/have locks so it’s easy to retrieve data. There was a suspect that stole the car of their victim he just murdered and used his voice to operate the stereo. They were able to obtain his voice command and use it as evidence they presented to the suspects relatives, who were able to confirm his identity.

University of Tennessee at Knoxville started a murder Accountability Project where they use an algorithm to help solve serial homicide cases. Authorities in Indiana are using data on behavior patterns, crime anniversaries, parole dates and more to expose pattern that could keep people safer. Telephone Data contains Call Detail Records (CRDs) which includes information on identification code, length of the call, and specific cell towers. There was a murder in 2016 of a woman who went for a run in Queens, NY. Prosecutors were able to convict the suspect due to phone records (Matthews).

Metadata is the information packet that comes in the background of your files. Meta means beyond (Data beyond data), in other terms data about your data. It can show who created the file, what time, and where. Any data you post online has this data attached to it. There is data embedded in many digital files, such as file name, file extension, date last accessed, date created, date last modified, email metadata, file, system data, etc. A lot of a user’s metadata is recorded in subsystem and other applications within the system. This is what forensic specialists use to spot inconsistencies. This type of investigation requires highly trained professionals (“How Forensic Specialists Solve Crime Through Metadata”).

As important as decreasing physical human danger, it’s also important to protect people’s online lives. A few ways cybercrimes can occur is in the form of hacking – illegally sending instructions to another computer to gain access to personal information-, child pornography and abuse -criminals seek minors by messaging systems in hopes of collecting child pornography-, and piracy -violating copyrights and downloads (Bandakkanavar). Data kept from the browser history can determine if an attack occurred within a system or from the outside. Browser data can also display possible issues by looking at search history and visited sites for example, it will sort out things like “How to make a bomb” or any other potentially threatening searches. It’s interesting when thinking about how this data comes to light. Authorities can get their hands on this data through warrants but also by purchasing it. On many sites it asks about privacy matter and usually people accept them without even reading them. This can grant access to the authorities even if a person isn’t involved in a crime. However, there is a new web 3.0 coming out that is supposed to give people more control of their privacy which will make it harder for authorities to be able to access that data. It remains important that authorities are to be careful with data, so they aren’t making biased decisions or coming to impaired conclusions (Flynn)**.** When it comes to online fraud, data mining and statistical analysis have been significant tools in decreasing this issue. Companies can use decision trees, machine learning, cluster analysis, etc. to scan transactions to identify patterns and spot fraudulent transactions (Volkov).

Looking at statistics, Identity theft was the most common type of complaint submitted by consumers to the Federal Trade Commission in 2021. There was a total of 1,686,121 reports of identity theft in 2021. Credit card fraud was the second leading action of identity theft, decreasing by 1% from 2020 with 389,845 reports. Majority of reports came from people between the ages 30 to 39 (Daly and Caporal).

There was a quote that said, “Data that cannot be collected cannot serve to answer a research question” (Van Gelder and Van Daele). And yes, it’s common sense, but data is the leading force in all the actions taken. You can’t collect criminal data as it’s occurring, this leads to restrictions when collecting research. However, it’s allowed focus on visible elements like offender backgrounds, demographics, and criminal careers. Another thing that must be considered is, just because there is data available, that doesn’t mean you have the resources to be able to analyze the data and extract the best results. The steps in the process of collecting the data and analyzing it can be resource intensive. This brings three big issues: Accessibility to data, type of data analysis and availability of resources (Van Gelder and Van Daele).

A top concern is of bias is from historical outcomes that Risk Assessment Instruments (RAI’s) learn to predict. If models start producing unfair practices, secondary models might start to pick them up too which pulls away from what the true risk for misconduct is. Consider this, Race groups have been estimated to consume the drug, Marijuana, at equal rates. However, black Americans have been convicted at a higher rate compared to other races. If a model is learning to predict convictions based off past data, Black Americans will be more susceptible to being targeted. This is unfair because the rate of using marijuana is the same for all race groups. The focus needs to be more on the selection of outcomes models are being trained with to properly forecast the accurate rates (Chohlas-wood).

There are two parts to modeling, the code and the mathematics done behind the scenes. Below will be a rundown of significant models and the meaningful parts that make the equations different from others that will be used later in the paper. This mathematical information is coming from Oscar Aguilar’s Machine Learning Course at Grand View University.

First, the Lasso variable selector model is computed by:

A picture containing diagram

Description automatically generated

This model assumes that there is a linear association between your variables, which is why variable selectors can be questionable if the data being used is not skewed.  is the classification and  is what is being used to decide the classification. is representing the number of samples in the data set. Lasso uses the Residual Sum of Squares (RSS) and the penalization term,

Text

Description automatically generated with medium confidence

This is an L1 penalty which means it will limit the size of the coefficients. The tuning parameter that holds these values is the lambda. When this tuning parameter is large, it can lead to some coefficients becoming zero and then they be taken out of the model.

Next, Random Forest, it’s a powerful tool to solve bias-variance trade-off problems. First, figure out how many trees your data needs to accurately assess the data. Too many can lead to overfitting the data, this will lead to high error in the testing set because the model won’t know what to do when it gets a completely new portion of data it wasn’t trained correctly for. If the output you receive shows multiple trees, with the same average cross-fold validation accuracy, having the best accuracy, you will choose to implement the least number of trees. The more trees in the model, the more complex the model is, so to keep it as simple as possible, proceed with the least number of trees with the highest average validation accuracy. The path for selecting importance’s is shown through,

A picture containing text, watch, clock

Description automatically generated

*n(j)* indicates the number of samples getting to the node *j*, the sum is to be calculated to all nodes where the feature is selected.  is representing the number of samples in the data set and  is the number of bootstrap samples we are applying with the data.  is the impurity reduction attained at node *j* after splitting using the feature *I*. The average is then computed over all trees. Which is shown by:

A picture containing text, clock

Description automatically generated

The output features are the “importance features” that have the most effect on the model. One thing to keep in mind is that you want as many variables as possible that you can control in your importance features. You can’t control the outputs of the features, but when you proceed with modeling, keep in mind how much control you have over the variables you’re using.

Once feature selection is completed, proceed to creating the random forest model. Random Forest don’t require any scaling and you can use continuous and categorical features together. They limit their randomness by choosing the best selection tuple from a smaller sample subset. The model creates many of these decision trees and then combines the results to find the tree averages and that is what is returned.

*Figure 1: Decision Tree Construction*

**Diagram

Description automatically generated**

*This is an example of how a Random Forest would be set up for decision trees to make combinations of the options available. Then, all the decision trees that were created get averaged and that is the output received. This image was taken from:* [*https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb*](https://medium.com/capital-one-tech/random-forest-algorithm-for-machine-learning-c4b2c8cc9feb)

Confusion Matrices are very common in classifications. It’s a table that groups predictions based on whether they match the actual value. One side of the table is for predicted values and the other is for the actual values.

*Figure 2: This diagram gives a visual of how a confusion matrix is set up*

A picture containing table

Description automatically generated

From *figure 2*, the four components that go into a confusion matrix are:

* + - 1. True Positive- Correctly classified (TP)
      2. False Negative- Incorrectly classified (FN)
      3. False Positive- Incorrectly classified (FP)
      4. True Negative- Correctly classified (TN)

From here, statistics on model behavior are computed. One that will be focused on for this study is recall. It’s a measure of how complete the results are. A model with high recall detains a large proportion of the positive samples which means it has a wide span. Higher the recall scores the better. Below is how recall is computed from the confusion matrix:

Another statistic that will be used is accuracy. The accuracy of a model tells us how good a model is at finding relationships between the training and testing data. The higher the accuracy scores the better our model is being trained. Below is how recall is computed from the confusion table:

Next, is Multi-Layer Perceptron models. These models use neural networks, which teach computers to process data in a way that is inspired by the human brain. It uses perceptron’s, which are neural network elements, that do computations to detect variables in the input data. This is a feed network which means the flow of information begins in the first layer and continues in the same direction until it hits the output layer. *W* and *H* are the weight matrices and have interrelated bias vectors ***b*** and ***c***. A picture containing text, clock

Description automatically generated is representing the weight associated with specific variables. If a perceptron has hidden layers, it’s denoted by *m*

*Logo

Description automatically generated with medium confidence* and . The transformations are defined below:

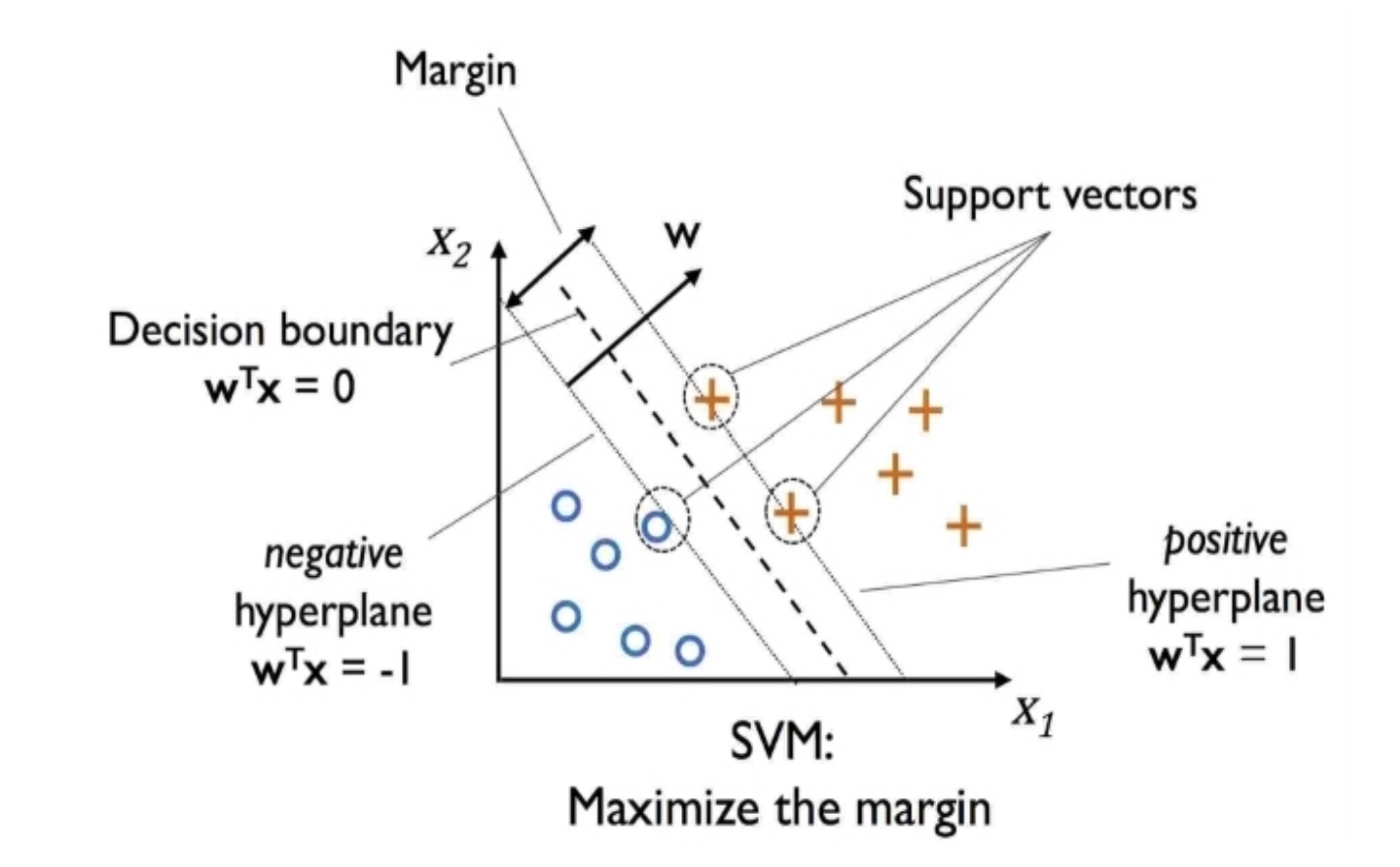
Text

Description automatically generated with medium confidence

MLP have many activations functions that can be applied, these can be considered as “tuning parameters” because it depends on the data inputted for what function will have the best performance, this is indicated by  and 

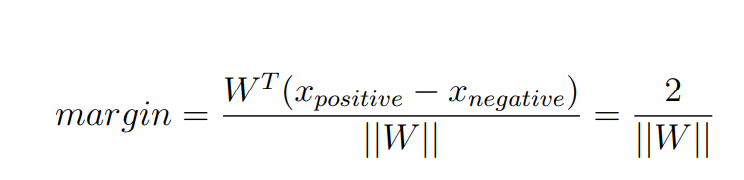
Similar to MLP, are Support Vector Machines because they are both neural networks. The goal of SVM’s is to find boundaries between classes so there are less misclassification errors. It finds the decision boundary that has the largest margin

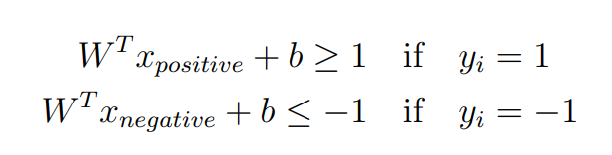
*Figure 3: Geometrical interpretation of maximum margin*



*From Oscar Agular Machine Learning Course*

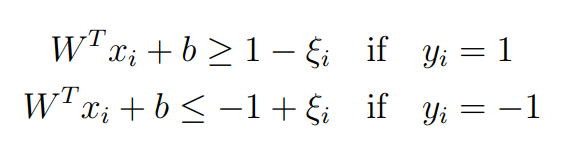
Larger margins have decreased misclassifications errors when compared to smaller margins. Below is the margin formula along with the constraints used to help the decision boundary classify all the points correctly:



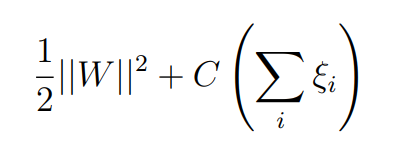


The module is computed by: A picture containing text

Description automatically generated. Hard margins fit the model with zero errors. Soft margins allow the model to be fitted with some errors. In this case, slack variables are added to change the linear constraints.



The next step is to minimize the error, using a cost penalty *C*:



Then there is Ada Boost. Ada Boost starts with a prediction data set:

Text

Description automatically generated with medium confidence

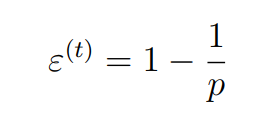
This is for a multi-class scenario so there are *p* number of classes the data can be predicted to.

A picture containing text

Description automatically generated

The threshold is pushed forward and A picture containing text

Description automatically generated will be zero when the following condition when:



Using the AdaBoost SAMME, which is a variation of the AdaBoost, called Stagewise Additive Modeling using Multi-class Exponential (SAMME), allows for the correction of dealing with multi-class problems without the bias from AdaBoost.M1 which hinders the error of multi-class predictions.

The last model discussed will be XG Boost. This model uses implementations of Gradient Boosting and parallel random forest and AdaBoost models. The framework of XG Boost is very complex. The output is given in a probability vector where each term accounts for the relative probability that the right class is the one it’s assigned to. The loss function is represented by the log-loss:

A picture containing text, watch

Description automatically generated

A picture containing text

Description automatically generated

\* *It’s important to note that if the point xi being used*

*is connected to the jth label, it equals 1 or 0 otherwise.*

Now that we’ve examined the equations that will be used, let’s walk through a. In this example you will be taken through the stages of looking at the data available, cleaning it, deciding which variables provide significance to the overall goal, the process of building models, and how end results are obtained.

Starting with the data, the Fraud Detection at Self-Checkouts in retail data set was taken from the 2019 Data Mining Cup Competition. The task was to use this historical data to create a model that will correctly classify transactions as fraudulent or not fraudulent. This data contains customer transaction information, the only thing related to the actual customer is their Trust Level (variable: *trustLevel*). The train data set consists of 1,879 participants (rows) transaction information variables (columns), in this data set, there are 10 columns. Below is a description of the data:

|  |  |  |
| --- | --- | --- |
| ***Column Name*** | ***Description*** | ***Value Range*** |
| **trustLevel** | Customers individual trust level | {1,2,3,4,5,6}  6 is the highest |
| **totalScanTimeInSeconds** | Total time in seconds between the first and last product scanned | Positive whole number |
| **grandTotal** | Grand total of products scanned | Positive decimal number with maximum two decimal places |
| **lineItemVoids** | Number of voided scans | Positive whole number |
| **scansWithoutRegistration** | Number of attempts to activate the scanner without scanning anything | Positive whole number or 0 |
| **quanityModifications** | Number of modified quantities for one of the scanned products | Positive whole number or 0 |
| **scannedLineItemsPerSecond** | Average number of scanned products per second | Positive decimal number |
| **valuePerSecond** | Average total value of scanned products per second | Positive decimal number |
| **lineItemVoidsPerPosition** | Average number of item voids per total number of all scanned and not canceled products | Positive decimal number |
| **fraud** | Categorization of fraud (1) or not fraud (0) | {0,1} |

Next a set of tasks were brainstormed. It’s important to have an idea of what you’re going to get out of your data to reach the overall goal. Through the creation of variable selector models, predictive models, and visualizations we will address the following questions:

* What variables are significant?
  + Create a lasso variable selector to see which variables are significant when predicting on fraudulent transactions
  + Create a Random Forest Importance variable selector to compare variables to the one’s the Lasso model came up with
    - Then use these variables to then create models
* How should performance be evaluated?
  + Recall and Accuracy
* What machine learning model will predict the best outcomes?
  + Lasso, Ada Boost, Random Forest, SVM, etc.

As these questions are kept in mind, an initial explanatory analysis should be done to see value ranges, relationships, discrepancies, observe patterns, etc. For this data set, it was deemed significant to note the spread of trust level since it’s the only aspect we have on the customers. Then the number of fraudulent transactions was compared to the number of transactions that weren’t. Next Trust level and transaction conclusion were paired to see if there was a relationship. Lastly one variable was selected to overlap transactions to see if there was a relationship with that specific random variable.

*Chart, bar chart

Description automatically generatedFigure 4*

Figure 4: It’s important to know your customer as well as their behaviors. With this being the only customer-related variable, it’s important to see how well the customers in this training study are trusted. The range is from 1 to 6, with 6 being the best score. It wasn’t explained how the trust level was computed. From the graph, we can see there are more “lower” trusted customers than higher trusted customers but not by very many. The difference between each group is less than 100.

*Figure* Chart

Description automatically generated*5*

*Figure 5: This chart shows the amount of fraudulent transactions total. It is being shown using a 0-1 scale, 0 being not a fraudulent transaction, 1 being a fraudulent transaction. We can see that this number is low compared to the amount of data that we have. In total we have 1,879 transactions, and we have a little over 100 fraudulent charges*

Chart, histogram

Description automatically generated*Figure 6*

*Figure 6: This chart represents the range of trust level, 0-6, and how many fraudulent transactions are reported for each trust level. We can see that the only trust levels with fraudulent transactions are 0 and 1, which are the lowest levels of trust, and the most being at trust level 1.*

Chart, bar chart

Description automatically generated*Figure 7*

*Figure 7: Darker blue represents more fraudulent transactions, where lighter blue represents less fraudulent transactions. This graph shows that the more times a scanner is activated without anything being shown, there is a higher likelihood of being fraudulent. There is some darker shading between 0 and 1 times the scanner was activated but the most dramatic effect is between activations 5 to 10 times. This gives the evidence that when moving forward with modeling, it could be an important variable in model training.*

Each variable could be compared to Trust Level and each other. It would be more efficient and accurate to run two tests that are made to pick out important variables. First is Lasso. Lasso uses an L1 penalty which means it will limit the size of the coefficients. When the tuning parameter is large, it can lead to some coefficients becoming zero and then be taken out of the model. The steps taken to create these results:

1. Import Data
2. Make sure there are no NA (missing values) values and if there are, drop them
3. Define a list to store appended lasso coefficients in
4. Run a for loop with 100 iterations\*, containing actions:
   1. Define the *x* and *y* variables
   2. Split the train data set in 80/20 → 80% to train and 20% to test
   3. Estimate the lambda
   4. Extract the Lambda
   5. Build the Lasso Model
   6. Append the lasso coefficients into the list
5. Convert list to a data frame and output the results

\*This number can be any number, 100 was chosen to get an accurate representation of the model.

Below are the results:

*Figure 8*

Table

Description automatically generated

The rows represent an iteration (1-100) and each column represents a variable being tested on (minus fraud because that’s what we’re predicting). Looking at *figure 8*, in order starting at 0, these are the variables associated with each column: *trustLevel, totalScanTimeInSeconds, grandTotal, lineItemsVoids, scansWithoutRegistration, quanitityModification, scannedLineItemsPerSecond, valuePerSecond, lineItemVoidsPerPosition.*

As you can see, the lambda that was generated by the model was large enough to shrink most of the variables down to zero. At first this could be very deceiving because it looks like all but one of the variables is insignificant. One thing to keep in mind though, is that Lasso assumes your data has a linear association (Aguilar). So, your results will be inaccurate if there isn’t a linear association in the data. The next test to run is the Random Forest Importance feature selector!

The steps to creating the Random Forest importance feature selector:

1. Import the data
2. Check to see if there are any NA (missing values) values, and if there are, drop them
3. Created a list to store appended results
4. Run a for loop with 100 iterations, containing actions:
   1. Define the x and y variables
   2. Split the train data set in 80/20 → 80% to train and 20% to test
   3. Build Rando Forest model
   4. Append important features into the list
5. Convert the list to a data frame and output results

*Figure 9*

**Graphical user interface, text, application

Description automatically generated**

The Lasso and Random Forest results might not always match and that’s okay because of the assumption lasso has on the data. Looking at *figure 9*, we can see the variables with the most importance (the highest numbers), are: trustLevel, scannedLineItemsPerSecond, totalScanTimeInSeconds, lineItemVoidsPerPosition, valuePerSecond. We are going to move forward with the results of the Random Forest Importance test and use the variables stated above to build the models to predict fraudulent transactions.

From here we will use our train data set to build models such as AdaBoost, Random Forest, Multi-Layer Perceptron, Gradient Boost, etc. to see which model reports the best accuracy measurements. While doing this we will have to adjust tuning parameters on models to see how the model should be set up to begin with, then we can compare the models to each other.

**Multi-Layer Perceptron and Support Vector Machines**

*Figure 10*

Chart, histogram

Description automatically generated

|  |  |
| --- | --- |
| **Model** | **Recall Score** |
| MLP 1 | .04523809523809524 |
| MLP 2 | 0.24904761904761905 |
| SVC 1 | 0.43 |
| SVC 2 | 0.21904761904761905 |

Looking at the graph above, there were two types of models tested, each having two models with different parameters. The Multi-Layer perceptron models had one model with one single hidden layer with 4 neurons (hyperbolic tangent as the activation function) and softmax as the activation function for the output, the stochastic descent gradient as the method to estimate the weights (optimizer = ’sgd’) and metrics = [’accuracy’], epochs = 100 and batch size = 500 to build the model. The second MLP model had one single hidden layer with 4 neurons (ReLU as the activation function) and softmax as the activation function for the output, the stochastic descent gradient as the method to estimate the weights (optimizer = ’sgd’) and metrics = [’accuracy’], epochs = 100 and batch size = 500 to build the model.

Also featured in the graph is the Support Vector Machine models, one model used ‘rbf’ as the kernel and the other used ‘poly’.

By looking at the graph, you can see that all four models will have low average recalls because of the consistent fluctuation of recall values. This taken into consideration with our future model recall scores, we decided that these models are insignificant to our overall study on fraud prediction.

**Random Forest Results**

With random forest testing having many variations of parameters to possibly test we put them all into a data frame, seen below, to get the different combinations we would want to see. In total there were 12 different combinations between 100 trees, 300 trees, 500 trees, and 1000 trees, along with depths of three, five, and seven. Using this data frame, a loop was created to run each model 100 times, with varying splits each time.

*Figure 11*

|  |  |
| --- | --- |
| **Original Data-frame** | **Result Data-frame** |
|  | **Recall-**  **A picture containing graphical user interface  Description automatically generatedAccuracy-**  **Application  Description automatically generated with low confidence** |

All the models with a depth of three had on average a recall score around .89, then the models with depths of five had on average values of .91 for recall, and models with depths of seven had recall values between .89 and .92. Next, we must take into consideration accuracy scores. Looking at the model that has the highest recall, which was a model with 1000 trees and a depth of 5, this model had an accuracy of .87, which is close to the average of all models seen above. To determine what the accuracy should be, it would depend on the business standards, but for this study we will continue with model 10, a Random Forest model with 1000 trees and a depth of 5 as the best model for this testing group because of its high recall and accuracy scores.

**XGBoost Results**

The result data frame has numbered columns that match the data frame with the list of trees. The results data frame shows many statistics but what we are focused on is the average recall and accuracy that was computed for each model (row-mean). XGBoost had 36 models that were tested. All the same combinations of models from Random Forest were used, except this type of modeling also takes into consideration learning rates, which are 0.001, 0.01. and 0.1 that were tested with those 12 previous models, thus giving us our 36 models. Using this data frame, a loop was created to run each model 100 times, with varying splits each time.

*Figure 12*

|  |  |
| --- | --- |
| **Original Data Frame** | **Results Data Frame** |
| **A picture containing diagram  Description automatically generated** | **Recall-**  **Table  Description automatically generatedTable  Description automatically generated**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Accuracy-**  **A picture containing application  Description automatically generated**  **Graphical user interface  Description automatically generated with low confidence**  **A picture containing text  Description automatically generated**  **Table  Description automatically generated** |

A data frame with all the trees being tested are above, for XGBoost there were 36 models run. For XGBoost, there are many models with the same recall score of 1. However, seeing these results we decided to take accuracy into account, and all the models with recall values of 1, had extremely low accuracy scores, in the range of 5.5e-2. This means that we would not choose any of these to predict fraud, because accuracy is so low. We then look for the model with the next highest recall scores, .93, which is seen with model 10, 300 trees a depth of three and a learning rate of .01. This model also has a higher accuracy score of .89, therefore this is the best XGBoost model for predicting fraud

**AdaBoost Results**

AdaBoost also had 36 models that were tested. Like XGBoost, this type of modeling takes into consideration learning rates. The same model combinations from XGBoost were tested with this type of model. Using this data frame, a loop was created to run each model 100 times, with varying splits each time.

*Figure 13*

|  |  |
| --- | --- |
| **Original Data Frame** | **Results Data Frame** |
| **A picture containing graphical user interface  Description automatically generated** | **Recall-**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Accuracy-**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Table  Description automatically generated**  **Table  Description automatically generated** |

Looking at *figure 13,* for the AdaBoost models, there were 35 models run. All 35 models had recall scores within similar ranges, and the accuracy scores were also similar. However, the model that stood out as best was model 30 because it had the slightly higher recall value of .591, and when looking at the accuracy score paired with this model, we see a higher score of .9185, as well. Therefore, although the models are similar overall, the best AdaBoost model for predicting fraud is model 30, which consists of 1000 trees, a depth of 5 and a learning rate of .1.

**Concluding Results:**

Taking the best models from each of the modeling types we have narrowed our choices down to a Random Forest model with 1000 trees and a depth of 7 for a recall score of .92 and an accuracy score of .87. An XGBoost model with 300 trees, a depth of 3, and a learning rate of .01 for a recall score of .93 and an accuracy score of .89. Along with an AdaBoost model with 1000 trees, a depth of 5, and a learning rate of .1 for a recall score of .591 and an accuracy score of .9185.

After comparing these three models, the conclusion is that the XGBoost model with parameters of having 300 trees, a depth of 3, and a learning rate of .01 is the best model overall for detecting fraud. As seen earlier this was not the highest recall score, we saw out of all the XGBoost models, however taking into consideration the accuracy scores this became our best choice of XGBoost models, and the overall models.

To improve our results, we could alter more parameters, execute more iterations, and even add more types of models. Being that there are so many different types of models, with different parameter combinations, there could be an infinite number of models possible, leading us to the conclusion that we are never certain what model is perfect in each scenario.

Throughout this process we learned the best way to write our models and perform them, through trial and error. One mistake we originally made was running the models on different splits causing the issue that the models could not be compared because they were not using the same data splits. This is because splits are completely randomized each iteration the models are not being trained the same way, creating inconsistent results. After these repetitions we were able to clean up our code and run the models correctly, giving us organized results to present.

The point of this study was to show how experts go about finding the best models to predict the best outcomes. As stated above, predicting fraudulent transactions is a touchy topic because you don’t want to accuse an innocent customer of a crime. This can be extended and made as an overall statement for the public: Authorities don’t want to suspect innocent people of vicious behavior. As technology continues to strengthen, more models are created and criminals stay active, it’s important to use technology in an appropriate manner to succeed in the overall goal of using technology to decrease crime.

Works Cited

Bandakkanavar, Ravi. “Causes of CyberCrime and Preventitive Measures”.  *Krazytech,* 18 June

2022, <https://krazytech.com/technical-papers/cyber-crime>

Chohlas-wood, Alex. “Understanding risk assessment instruments in criminal justice” *The*

*Brookings Institution*. 19 June 2020, [https://library.unr.edu/help/quick-how-tos/citing-a-website-in mla#:~:text=Author%20Last%20Name%2C%20First%20Name,of%20publication%20or%20update%20date%5D](https://library.unr.edu/help/quick-how-tos/citing-a-website-in%20mla#:~:text=Author%20Last%20Name%2C%20First%20Name,of%20publication%20or%20update%20date%5D)

Daly, Lyle and Jack Caporal. “Identity Theft and Credit Card Fraud Statistics for 2022”. *The*

*ascent,* 21 September 2022, <https://www.fool.com/the-ascent/research/identity-theft-credit-card-fraud-statistics/>

Flynn, Shannan. “How Does Law Enforcement Use Data to Solve Crime?”. *Make Use Of.* 12

February 2022,<https://www.makeuseof.com/data-in-law-enforcement/#:~:text=Law%20enforcement%20agencies%20often%20use,gaps%20in%20crime%2Dfighting%20coverage>.

“How Forensic Specialists Solve Crime Through Metadata”. *CaseGuard.* 28 July 2020,

<https://caseguard.com/articles/how-forensic-specialists-solve-crime-through-metadata/>

Matthews, Kayla. “How Data Analytics Is Solving Murders”.  *Towards Data Science*. 22 July

2019,<https://towardsdatascience.com/how-data-analytics-are-solving-murders-1cdac5432d6e>

O. Aguilar, personal communication, August 2022

Tikkanen, Amy. “Denis Rader: American Serial Killer”. *Britannica.* 3 November 2016,

<https://www.britannica.com/biography/Dennis-Rader>

Van Gelder, Jean-Louis and Stijn Van Daele. “Innovative data collection methods in

criminological research: editorial introduction”. *Biomed Central.* 12 August 2014, [Innovative data collection methods in criminological research: editorial introduction | Crime Science | Full Text (biomedcentral.com)](https://crimesciencejournal.biomedcentral.com/articles/10.1186/s40163-014-0006-1#Sec1)

Volkov, Michael. “Fraud Detection: New Technologies and Analytics (Part II of III)”. *Volkov Law,*

11 December 2018, <https://blog.volkovlaw.com/2018/12/fraud-detection-new-technologies-and-analytics-part-ii-of-iii/#:~:text=Data%20mining%20and%20statistical%20analysis,patterns%20and%20detect%20fraudulent%20transactions>.

“3 Famous Cases Solved Through Digital Forensics”. *EclipseForensics.* 1 July 2021,

<https://library.unr.edu/help/quick-how-tos/citing-a-website-in-mla#:~:text=Author%20Last%20Name%2C%20First%20Name,of%20publication%20or%20update%20date%5D>

**Appendix**

**Code used to generate graphs**

pip install xgboost

import boto3

import pandas as pd; pd.set\_option('display.max\_column', 100)

import numpy as np

from statistics import mean

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import recall\_score, accuracy\_score

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression, Lasso, LassoCV

from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier, GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

import tensorflow as tf

import matplotlib.pyplot as plt

from itertools import product

import xgboost as xgb

import warnings

warnings.simplefilter(action= 'ignore', category=FutureWarning)

s3= boto3.resource('s3')

bucket\_name= 'morgangant-bata-445-bucket'

bucket= s3.Bucket(bucket\_name)

file\_key= 'train.csv'

bucket\_object= bucket.Object(file\_key)

file\_object = bucket\_object.get()

file\_content\_stream = file\_object.get('Body')

#reading the datefile

project = pd.read\_csv(file\_content\_stream)

project.head()

#Dropping na values

project = project.dropna()

importances= list()

df= list()

for i in range (1,100):

## Defining the input and taregt variables

x= project[['trustLevel', 'totalScanTimeInSeconds', 'grandTotal', 'lineItemsVoids', 'scansWithoutRegistration', 'quanitityModification', 'scannedLineItemsPerSecond', 'valuePerSecond', 'lineItemVoidsPerPosition']]

y= project['fraud']

#Splitting data into train and test

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2)

#Building Model

RF\_md= RandomForestClassifier(n\_estimators= 500).fit(x\_train, y\_train)

#Extracting the feature importances

importances.append(RF\_md.feature\_importances\_)

#Estimating lambda for lasso

lasso\_cv= LassoCV(alphas= [.001, .01, .1, 1, 10, 100], normalize= True, cv=5).fit(x\_train, y\_train)

#Extracting best lambda

cv\_lambda= lasso\_cv.alpha\_

#print('Estimated lambda for the lasso model is:', cv\_lambda)

#Building lasso

lasso\_md= Lasso(alpha= cv\_lambda, normalize= True).fit(x\_train, y\_train)

df.append(lasso\_md.coef\_)

a = pd.DataFrame(importances)

a.columns= ['trustLevel', 'totalScanTimeInSeconds', 'grandTotal', 'lineItemsVoids', 'scansWithoutRegistration', 'quanitityModification', 'scannedLineItemsPerSecond', 'valuePerSecond', 'lineItemVoidsPerPosition']

a.apply(np.mean, axis = 0)

b = pd.DataFrame(df)

b

def expand\_grid(dictionary1):

return pd.DataFrame([row for row in product(\*dictionary1.values())],

columns = dictionary1.keys())

dictionary1 = {'n\_tree': [100, 300, 500, 1000],

'depth': [3, 5, 7]}

parameters1= expand\_grid(dictionary1)

parameters1= expand\_grid(dictionary1)

def expand\_grid(dictionary2):

return pd.DataFrame([row for row in product(\*dictionary2.values())],

columns = dictionary2.keys())

dictionary2 = {'n\_tree': [100, 300, 500, 1000],

'depth': [3, 5, 7],

'learning\_rate': [0.1, 0.01, 0.001]}

parameters2 = expand\_grid(dictionary2)

def expand\_grid(dictionary3):

return pd.DataFrame([row for row in product(\*dictionary3.values())],

columns = dictionary3.keys())

dictionary3 = {'n\_tree': [100, 300, 500, 1000],

'depth': [3, 5, 7],

'learning\_rate': [0.1, 0.01, 0.001]}

parameters3 = expand\_grid(dictionary3)

#defining data frames to hold data

rf\_recall= pd.DataFrame()

rf\_accuracy= pd.DataFrame()

ada\_recall= pd.DataFrame()

ada\_accuracy= pd.DataFrame()

xgb\_recall= pd.DataFrame()

xgb\_accuracy= pd.DataFrame()

#lists for results

md1results = list()

md2results = list()

md3results = list()

md4results = list()

#Define the input and target variable

x= project[['trustLevel', 'scannedLineItemsPerSecond', 'totalScanTimeInSeconds', 'lineItemVoidsPerPosition', 'valuePerSecond']]

y= project['fraud']

for i in range (0,100):

print([i])

#Splitting the Data

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.2, stratify= y)

# min-max transformation

scaler = MinMaxScaler()

x\_train = scaler.fit\_transform(x\_train)

x\_test = scaler.fit\_transform(x\_test)

##### Random Forest

for k in range(len(parameters1)):

print([k])

#Buidling the model

rf\_md1= RandomForestClassifier(n\_estimators= parameters1['n\_tree'][k], max\_depth=parameters1['depth'][k]).fit(x\_train, y\_train)

#Predicting on the model

rf\_pred1= rf\_md1.predict\_proba(x\_test)[:,1]

#changing likelihoods to labels

rf\_labels1= np.where(rf\_pred1 < .1, 0, 1)

rf\_recall.loc[i,k]= recall\_score(y\_test, rf\_labels1)

rf\_accuracy.loc[i,k]= accuracy\_score(y\_test, rf\_labels1)

#### XGBoost

for m in range(len(parameters2)):

print([m])

#creating model

xgb\_md= xgb.XGBClassifier(max\_depth= parameters2['depth'][m], n\_estimators= parameters2['n\_tree'][m], learning\_rate= parameters2['learning\_rate'][m] ).fit(x\_train, y\_train)

#predicitng on the model

xgb\_pred= xgb\_md.predict\_proba(x\_test)[:,1]

#Changling likleyhoods to labels

xgb\_labels1= np.where(xgb\_pred < .1, 0, 1)

#Computing recall and appending score

xgb\_recall.loc[i,m]= recall\_score(y\_test, xgb\_labels1)

xgb\_accuracy.loc[i,m]= accuracy\_score(y\_test, xgb\_labels1)

#### Ada Boost

for g in range(len(parameters3)):

print([g])

#Building the model

ada\_md1= AdaBoostClassifier(base\_estimator= DecisionTreeClassifier(max\_depth= parameters2['n\_tree'][g]), n\_estimators= parameters2['depth'][g], learning\_rate= parameters2['learning\_rate'][g]).fit(x\_train, y\_train)

#Predicting on the model

ada\_pred1= ada\_md1.predict\_proba(x\_test)[:,1]

#changing likelihoods to labels

ada\_labels1= np.where(ada\_pred1 < .1, 0, 1)

ada\_recall.loc[i,g]= recall\_score(y\_test, ada\_labels1)

ada\_accuracy.loc[i,g]= accuracy\_score(y\_test, ada\_labels1)

#Non tree models

### model 1 ###

# defining the model

md1 = tf.keras.models.Sequential([

tf.keras.layers.Dense(4, input\_dim = 5, activation= 'tanh'),

tf.keras.layers.Dense(2, activation = 'softmax')

])

md1.compile(optimizer='sgd', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

md1.fit(x\_train, tf.keras.utils.to\_categorical(y\_train, num\_classes = 2), epochs = 100, batch\_size= 500, verbose = 0)

# predicting on test

md1pred = md1.predict(x\_test)[:, 1]

# cut off at 15%

md1labels = np.where(md1pred < .15, 0, 1)

#computing the recall score

md1recall = recall\_score(y\_test, md1labels)

md1results.append(md1recall)

### model 2 ###

# defining the model

md2 = tf.keras.models.Sequential([

tf.keras.layers.Dense(4, input\_dim = 5, activation= 'relu'),

tf.keras.layers.Dense(2, activation = 'softmax')

])

md2.compile(optimizer='sgd', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

md2.fit(x\_train, tf.keras.utils.to\_categorical(y\_train, num\_classes = 2), epochs = 100, batch\_size= 500, verbose = 0)

#predicting on test

md2pred = md2.predict(x\_test)[:, 1]

#cut off at 15%

md2labels = np.where(md2pred < .15, 0, 1)

#computing the recall score

md2recall = recall\_score(y\_test, md2labels)

md2results.append(md2recall)

### model 3 ###

#First SVC Model

svm\_model3 = SVC(kernel = 'rbf', probability = True).fit(x\_train, y\_train)

#predicting on test

svm3\_pred = svm\_model3.predict\_proba(x\_test)[:,1]

#cut off at 15%

svm3\_labels = np.where(svm3\_pred < .15, 0, 1)

#computing the recall score

svm3\_recall = recall\_score(y\_test, svm3\_labels)

md3results.append(svm3\_recall)

### model 4 ###

#Second SVC Model

svm\_model4 = SVC(kernel = 'poly', probability = True).fit(x\_train, y\_train)

#predicting on test

svm4\_pred = svm\_model4.predict\_proba(x\_test)[:,1]

#cut off at 15%

svm4\_labels = np.where(svm4\_pred < .15, 0, 1)

#computing the recall score

svm4\_recall = recall\_score(y\_test, svm4\_labels)

md4results.append(svm4\_recall)

x= [i for i in range(0,100)]

plt.plot(x, md1results, color = 'blue')

plt.plot(x, md2results, color = 'orange')

plt.plot(x, md3results, color = 'red')

plt.plot(x, md4results, color = 'green')

plt.xlabel('Split Number')

plt.ylabel('Recall')

plt.legend(loc = 'upper right', labels = ['MLP 1', 'MLP 2', 'SVC 1', 'SVC 2'])

plt.grid()

plt.show();

mean(md1results)

mean(md2results)

mean(md3results)

mean(md4results)

rf\_recall.describe()

xgb\_recall.describe()

ada\_recall.describe()