**National Crime Victimization Survey:**

**Data Mining**

**Final Report**

Professor Jae Lee

CS699 - A1

FALL 2023

Jonathan Ko

Theerarun (Steve) Tubnonghee

# TABLE OF CONTENTS

Executive Summary …………………………………………………………. 3

Introduction ………………………………………………………………. 3

Preprocessing Methods…………………………………………………………. 3 - 6

Preprocessing Results ……………………………… ……………………. 6 - 9

Preprocessing Conclusion …………… ………………………………….. 9

Classification Methods …………… …………………………………… 9 - 12

Data Mining Procedure …………… …………………………………… 12 - 16

Data Mining Results & Evaluation …………………………………..… 17 - 22

Data Mining Discussion & Conclusion …….…………………....….…… 22 - 24

Division of Work …………………..……………………….…………… 24

Appendix …………………………...……………………….…………… 24

# 

# Executive Summary

This Final report provides a detailed analysis of the initial preprocessing, data exploration data cleaning, and classification models of the project data set derived from the 2013 school crime supplement of NCVS (National Crime Victimization Survey). This project is a practice of building and testing classification models using a real world data set. The codebook of the original dataset, *34980-0001-Codebook.pdf*, has been provided along with the assignment and csv file in question. We are interested in building classification models which would predict bullying victims, compare their performance, and select the “best” model.

# Introduction

As mentioned in the project assignment, we obtained data that has already been preprocessed, therefore we will clarify the goals of the Final report. At its current state, the data involved would not yield favorable results to predict bullying victims with the following criteria in mind: *accuracy, completeness, consistency, timeliness, believability, and interpretability*. Our goal is to further preprocess the data, run and build a classification model that would predict a favorable outcome. We also hope to identify to the reader the variables and the outliers we decide to remove and provide justification. We also hope to provide justification for the classification models chosen and the results carried out.

# Preprocessing Methods

As we clarified in the introduction, we have the following criteria in mind: *accuracy, completeness, consistency, timeliness, believability, and interpretability*. In order to preserve these aspects of data quality we will perform the following data cleaning techniques below: missing data check, categorical data check, non-unique variable check, and outlier check.

## Dealing with missing data

Since the data set that was distributed to the students did not include any n/a nor missing data, this initial step for preprocessing is complete and further action is not required. colSums(is.na(data)) is a piece of R code that we can leverage to check if any of the columns contain n/a or missing data. With the interest of running our model with *completeness*, we will check to remove any missing or incomplete data.

## Dealing with categorical data

As mentioned above, the data set distributed to the students does not include categorical data namely without numerical data. However, since we do have the codebook of the original dataset, we will reference the data. Because of this exploration, we do not need to take further action on the data set for this goal. With the interest of running our model with consistency, we will check to harmonize all non-numerical data values in the data set.

## Dealing with outliers

The data set contains mostly numerical data, therefore it is necessary to handle the noisy data. We leveraged the use of IQR methodology to compute the quantiles at the 25% and 75% level, and subtract Quartile 3 from Quartile 1 to get the interquartile range. This would be the case if we were dealing with numerical data that has not been transformed from categorical data. After the initial investigation, we found that dealing with outliers via the IQR methodology would not make much sense with our data set. Implementing the IQR methodology caused the dataset to be reduced to 3 tuples, and that would not yield a successful model. With the interest of running our model with *believability*, we will leverage the use of data reduction techniques.

## Removing unimportant attributes

After further investigation with the data set, we found that there were several non-unique variables which contained numerical values that would not benefit the model and distinguish one data value from another data value. The goal is to reduce our data in a way that is still meaningful. In order to accomplish this task, we created a sapply(df, function(col) length(unique(col)) <= 1) - where df is the data.frame of the dataset - function that would find variables with only one unique value. From there, we could clean the data set and only include variables that are meaningful to the dataset. Therefore, with the interest of running our model with *timeliness*, we will remove variables from the dataset that do not provide additional value. Still, after removing the variables, we can keep in mind that they represent the attribute of the sample of the whole dataset. Later on, we found out that we can expect zero-variance from the variables by using nearZeroVar function in R.

## Removing unimportant attributes via zero-variance or near zero-variance (part 2)

Once we removed the variables that contained only one unique value, we expanded the exploration to variables that contained a small amount of unique values. This information would also be meaningful since we could leverage the method of zero-variance or near zero-variance attributes. The underlying concept explains that an attribute that has the same value for all samples and/or the ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value is large, then the model in question is susceptible to this type of predictor, and it may be advantageous to remove the variable from the model. In some cases such predictors can also cause numerical problems and cause a model to crash. This could occur either due to division by zero (if a standardization is performed) or due to numerical precision issues. In order to prevent this issue, we imported the library caret and used the nearZeroVar() function on the data set with a frequency cutoff of 95/5, which is a good cutoff point regarding the lecture slides. We stored this output into a variable, and obtained the list of nearZeroVar columns. Then we indexed to get a cleaned data frame of non-nearZeroVar variables.

## Remove attributes to avoid collinearity

After considering zero-variance or near zero-variance attributes, we also wanted to explore removing attributes to avoid collinearity. As mentioned in the previous method, we want to avoid potential issues in our model data, therefore it is important to avoid variables that have high correlation. We are aiming for the “best” model, and if we have predictor variables in the same regression model that are correlated, then they cannot independently predict the value of the dependent variable. In other words, they explain some of the same variance in the dependent variable, which reduces their statistical significance. To prevent this issue, we leveraged the findCorrelation function to remove the minimum number of predictor attributes in such a way that all pairwise correlations in the remaining attributes are less than or equal to a predetermined threshold of 0.8. After gathering the variables that crossed this threshold, we indexed into our cleaned data further and took out any potential collinear variables. We chose 0.8 as the threshold after some research into multicollinearity and misleading statistical results. Current literature supports that if the variance inflation factor is , then multicollinearity exists. Thus, we will move forward with 0.8 as the threshold for our test.

## Remove high VIF

After working through classification methods with our original 90 variables, we were unhappy with the initial results. The common problem we encountered was that the Sensitivity metric would be too high (predicted non-bullied individuals) while the Specificity metric was too low. We theorized that this could be due to multicollinearity. It was possible that there could be a correlation between independent variables. Since we have such a large dataset with many attributes, we tried to reduce as much as possible within reason. Therefore, we wanted to continue to filter additional variables. Current literature supports fitting a linear model and passing vif() and filtering out vif higher than 5. A value of VIF higher than 5 means a moderate correlation as anything less than 5 indicates low correlation of that predictor with other predictors.

## Filter variables using information gain

Similar to VIF, we added information gain to reduce our data set. The goal was to gather more consistent results to our performance models and information gain was effective in doing so. Information gain measures the reduction in entropy or surprise by splitting the dataset according to a given value of a random variable. A larger information gain suggests a lower entropy group or groups of samples, hence less surprise. Lower probability events have more information, and higher probability events have less information. Entropy quantifies how much information there is in a random variable. Using the InfoGainAttributeEval() function, we filtered out any variables that had a value of 0. Therefore, we concluded that those variables were unimportant to the dataset.

# Preprocessing Results

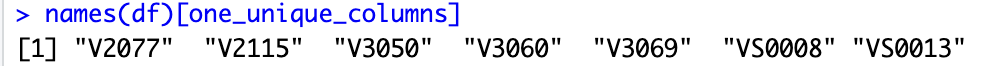
## Removing unimportant attributes

# Find columns with only one unique value

one\_unique\_columns <- sapply(df, function(col) length(unique(col)) <= 1)

# Print the column names that meet the criteria

names(df)[one\_unique\_columns]



\*see appendix for variable description

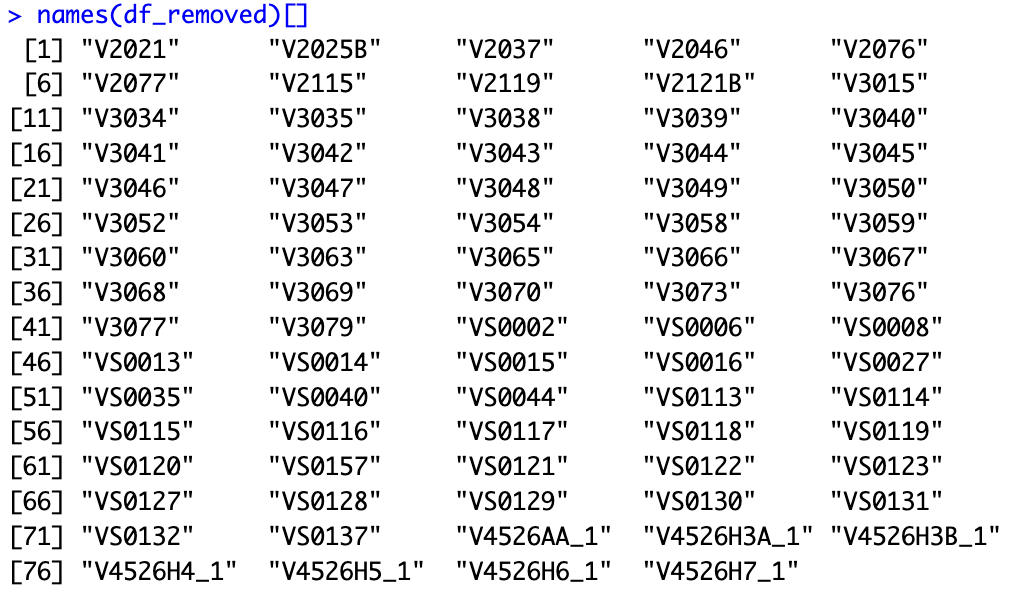
## Removing unimportant attributes via zero-variance or near zero-variance (part 2)

library(caret)

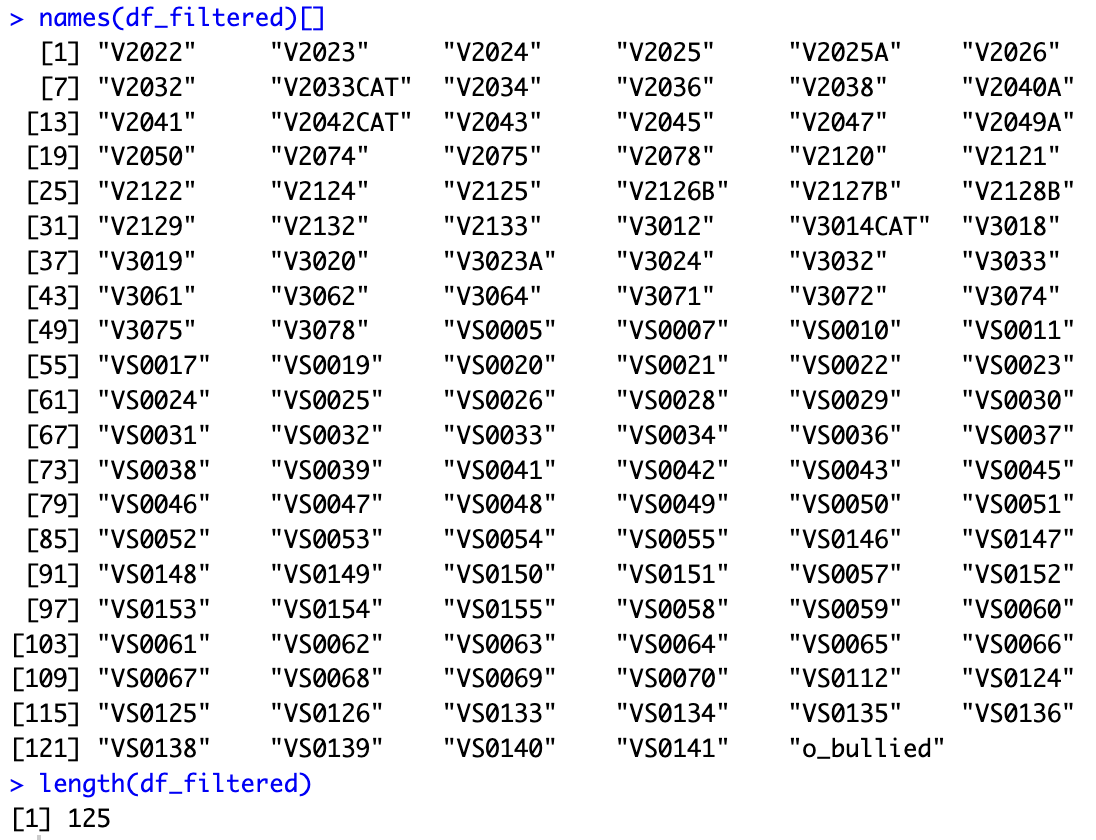
nearZeroVar(df)

nzv <- nearZeroVar(df,freqCut = 95/5)

# variables that had zero-variance



# remaining variables left from the data frame after near zero-variance filtering.



## Remove attributes to avoid collinearity

# Calculate the correlation matrix

correlation\_matrix <- cor(df\_filtered)

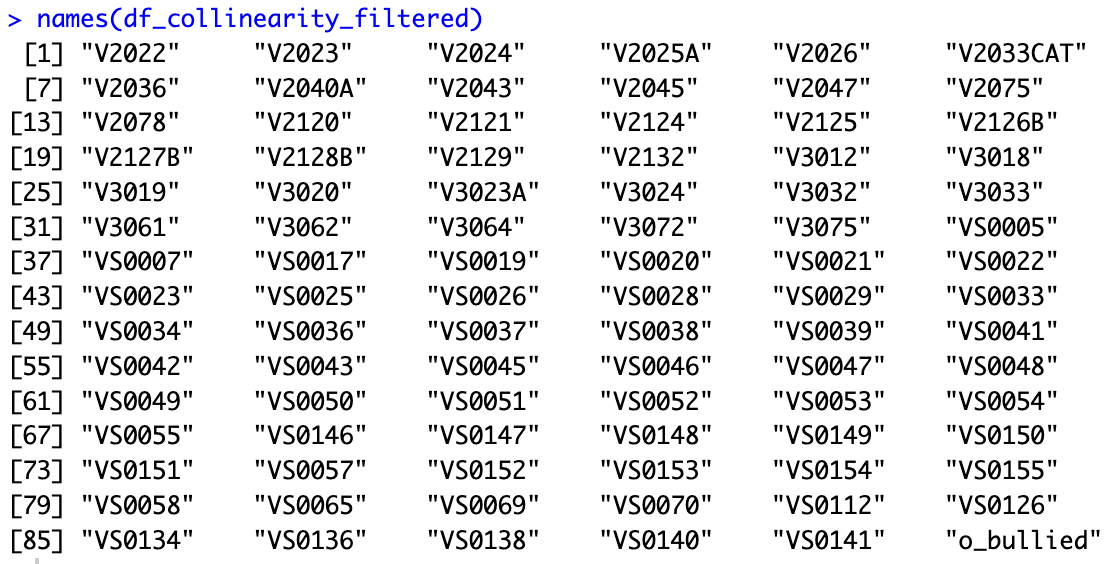
# Collinearity with cutoff at 0.8

highCorr <- findCorrelation(correlation\_matrix, cutoff = 0.8, names = TRUE)

length(highCorr)

[1] 35

# data frame after collinearity, near zero-variance and redundant attributes filtered.



## Remove high vif

We fit a linear model and pass it to vif() then we filter out the vif higher than 5.

model <- glm(o\_bullied ~ ., data = data\_clean)

vif\_values <- vif(model)

high\_vif <- vif\_values[vif\_values > 5]

data\_clean <- data\_clean[, !(names(data\_clean) %in% names(high\_vif))]

# 

## Filter variables using Information Gain

df <- data\_clean

df$o\_bullied <- factor(df$o\_bullied)

bullied.infogain <- InfoGainAttributeEval(o\_bullied ~ . , data = df)

sorted.features <- sort(bullied.infogain, decreasing = TRUE)

df <- data\_clean[,(names(data\_clean) %in% names(sorted.features[sorted.features>0]))]

df <- cbind(df,"o\_bullied"=data\_clean$o\_bullied)

# 

# Preprocessing Conclusion

To conclude, after data exploring and additional preprocessing, we found out of 204 variables in the original data set, 90 variables passed the threshold criteria we established for the data set. Through the first dataset filtering with redundant values in a given attribute, we found 7 variable violators. Next, we checked the data set for near zero-variance variables and removed an additional 79 attributes that did not pass the given cutoff value of 95/5. Finally, we examined the data set for collinearity between variables, and removed another 35 attributes which violated the correlation threshold of 0.8. Our final preprocessed dataset data frame contains 4947 tuples and 90 attributes.

After the intermediate report, we further reduced the size of predictors down to 39 variables. VIF and Information Gain greatly reduced our dataset so that we could gather consistent and clean results in the classification model steps.

# Classification Methods

After we have finished the preprocessing steps, we can continue to build classification models which would predict bullying victims, compare their performance, and select the "best" model. This step may require us to revisit the preprocessing steps if the performance criteria is not met. In this project, we are aiming to reach class 0 TPR >= 75% AND class 1 TPR >= 65%. Due to this criteria, we are interested in a few classification methods that will be described below:

## Bayesian Classification

Bayesian Classification performs probabilistic predictions. Theory is based off of Bayes Theorem which states that the probability of event A occurring given that event B has occurred is proportional to the probability of event B occurring given that event A has occurred, multiplied by the prior probability of event A and divided by the prior probability of event B. One advantage is that when the data set is large, it performs well. The underlying assumption is that class conditional independence which was weeded out during the preprocess phase.

## Boosting (Gradient Boosting, XGBoosting)

Friedman (1999) describes gradient boosting as a greedy algorithm that builds the model stage by stage. At each stage, it focuses on approximating the negative gradient of the loss function, which guides the learning process. The algorithm applies gradient descent optimization to minimize the loss function, and it updates the model iteratively by moving in the direction of the steepest decrease in the loss. Chen and Guestrin (2016) designed the optimized distributed gradient boosting library for efficient and scalable training of machine learning models. XGBoost, short for “Extreme Gradient Boosting” is a widely used machine learning algorithm due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in machine learning tasks such as classification and regression. The difference between XGBoost and other gradient boosting methods is regularization. XGBoost offers L1 (Lasso) and L2 (Ridge) regularization to limit the complexity of individual trees and improve the model’s generalization capabilities. L2 (Ridge) regularization is a technique that ensures you do not overfit your data. L1 (Lasso) is very similar to L2 (Ridge), however, instead of the ridge regression penalty (bias of slope) that is squared, L1 will take the absolute value.

## Decision Tree (Random Forest)

Random Forest is a popular ensemble learning method used for classification and regression tasks in ML. It is an extension of the decision tree algorithm and it combines the predictions of multiple decision trees to make more robust and accurate predictions. It is known for its high predictive accuracy and resistance to overfitting. Another reason we selected random forest is it can work well with datasets that have a large number of features and it can automatically select relevant features.

## Logistic Regression

Logistic regression is a popular machine learning method used for binary and multiclass classification tasks. Logistic regression models the probability that a given input belongs to a specific class. One key advantage is that it is a simple and interpretable algorithm making it easy to understand. It can be trained quickly on large datasets, and works well for linearly separable data. Logistic regression uses the logistic function (sigmoid function) to model the relationship between the input features and the probability of the input belonging to a particular class. In the training phase, logistic regression aims to find the optimal values of the parameters. Beta is then the best fit of the training data using a technique called maximum likelihood estimation.

## KNN (K-nearest neighbor)

K-nearest neighbor (KNN) is a simple and intuitive classification algorithm that is used for both binary and multiclass classification tasks. It’s a non-parametric and instance-based learning algorithm, meaning it makes predictions based on the similarity between the input data point and its neighboring data points from the training dataset. During the training process, KNN stores the training dataset, which consists of input feature vectors and their corresponding class labels. No explicit model is built because KNN is a lazy learner. KNN, then uses a distance metric (Euclidean Distance, Manhattan distance, or cosine similarity) to measure the similarity between data points. To make a prediction, KNN calculates the distance between the point and all the data points in the training dataset. It selects the K data points (neighbors) with the smallest distances. For classification, KNN performs majority voting among the K nearest neighbors. The class that is most frequently represented among the K neighbors is assigned to the new data point. In binary classification, K is often chosen as an odd number to avoid ties. The choice of K can significantly impact the model’s performance, since smaller values can make the model more sensitive to noise, while larger values provide smoother decision boundaries but may overlook patterns.

## Support Vector Machines

Support Vector Machines are powerful classification algorithms that are particularly effective for both binary and multiclass classification tasks, and they are known for their ability to find an optimal decision boundary that maximizes the margin between different classes in the feature space. SVM aims to find a hyperplane in the feature space that best separates the data into different classes. The hyperplane is chosen to maximize the margin which is the distance between the hyperplane and the nearest data points, support vectors, of each class. SVM is useful when the data is linearly separable meaning it can be separated by a hyperplane, however, it can be useful when the data is not linearly separable. SVM uses kernel functions to map the data into a higher-dimensional space where it becomes linearly separable. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid kernels. During the training phase, SVM finds the hyperplane (or decision boundary) that best separates the data while maximizing the margin. It does this by solving an optimization problem that minimizes the classification error and maximizes the margin. SVM includes a hyperparameter called C, which controls the trade-off between maximizing the margin and minimizing classification errors. A small C value emphasizes a larger margin, potentially at the cost of some misclassification. A large C value focuses on minimizing misclassification, potentially at the cost of a smaller margin. To classify new data points, SVM places them in the feature space and determines which side of the hyperplane they fall on. SVM is capable of binary classification. Some advantages of SVM is that it can handle datasets with a large number of features. SVM typically produces robust results and good generalization to new unseen data.

## 

## Neural Networks

Neural Networks are a class of machine learning models inspired by the structure and function of the human brain. A neural network is composed of artificial neurons, also known as nodes or units. These neurons are organized into layers, and a typically neural network consists of 3 types of layers: Input, Hidden, and Output. Input layer perceives the initial data or features, hidden layers process the data through a series of interconnected neurons, performing complex transformations, and the output layer produces the final classification or regression result. Neurons are connected to neurons in adjacent layers through connections with associated weights. These weights are adjusted during training to enable the network to make accurate predictions. Each neuron has an activation function, common activation functions include the sigmoid, ReLU (Rectified Linear Unit), and tanh functions. Features are fed into the input layer, and information is passed through the network from the input layer to the hidden layer, then to the output layer. This is done via feedforward propagation. During training, neural networks learn to adjust the weights of the connections between the predicted output and true output (loss/error). Backpropagation algorithm is used to calculate the gradient loss with respect to the weights in the network. This gradient is then used to adjust the weights in the opposite direction to minimize the loss. The output layer of a classification neural network typically uses an activation function like sigmoid to convert the network’s output into class probabilities. Once trained, the neural network can be used to classify new data. The class with the thresholded value is considered the predicted class.

# Data Mining Procedure

We proceeded with the typical procedure building of the model. The preprocessed dataset would be split between train and test datasets using the split() function with either a (70/66 - 30/34) split between train and test respectively. Once split, we would leverage the train dataset and build a model, evaluate, tune and choose the best model that would yield optimal results. Then, we take the test dataset and test and evaluate the performance of the chosen model based on class 0 TPR >= 75% AND class 1 TPR >= 65%. Unfortunately after running the data set through an initial pass for each of the classification methods above, we found that there were some anomalies with the results as well as a common trend. Typically the confusion matrix would yield a sensitivity measure that would be very high, while the specificity measure would be very low. Once discovering this abnormality, we sought to investigate the data set further.

An issue that was identified was the presence of an unbalanced class. The initial preprocess data had a clear uneven number of instances in each class. O\_bullied showed 3817 not bullied and 1130 bullied or (77% vs 23%). This would satisfy a moderate imbalance that would cause us investigators to make some action. An option we explored was resampling the dataset. The two methods (ROSE over-sampling vs ROSE undersampling) depend on which class is underrepresented, and in our case it was the minority group or “bullied” individuals. As a rule of thumb, it is suggested that for very large datasets, you should undersample, and vice versa for small datasets, however, in our investigation, we took a look at both methods. A final method which we did not use was the SMOTE (Synthetically Minority Over-sampling Technique), which works by selecting minority observations that are similar to each other and drawing a line between the examples in order to create new synthetic samples.

## Naive Bayes (bayes Theorem)

With every classification step, we initially took the data set and split it into a training set and testing set. Then, we trained our Naive Bayes model from the training dataset and using the model, we predicted the values on the test data set with classification parameters. After computing the predicted values, we gathered everything into a confusionMatrix to assess our performance. Once this step was completed, we looked into how to improve the model, which led to preprocessing the data further. We introduced the VIF which measured the amount of collinearity in the regression analysis. This allowed us to remove variables that wouldn’t contribute to the model. After we took out those variables, we also took a look at the classes to see if there were any imbalances. Since there was a stark difference between the positive and negative classes, we looked at oversampling and under-sampling the initial training dataset to see if it produced more favorable results.

## Boosting (Gradient Boosting, XGBoosting)

## XGBoosting was a rather tedious process. The way we prepared the data was vastly different from the other procedures. Like all classification methods, we started with splitting the dataset into training and testing sets. XGBoost requires the class attribute (labels) to be numeric, so we converted the training and testing set classes to numeric and prepared the data for the sparse matrix. XGB.DMatrix is the format that xgboost requires, therefore we convert the training and test data and labels to a matrix for model consumption. The first build of the model began with simple parameters; 2 rounds of boosting iterations and the objective function as binary:logistic. We continued to tune the model where we could maximize the model’s performance. Therefore, in the next iteration of the model, we introduced the parameter for maximum depth of each decision tree. Since this actually increased our logloss function, but improved the test error rate, we increased the boosting rounds, and leveraged the parameter early\_stopping\_rounds where if there is no improvement, the model would stop. Finally, we controlled for imbalanced classes using the scale\_pos\_weight parameter. We summed the negative cases and positive cases and took the ratio for each iteration of the boosting round. This significantly improved our model, however, we still did not reach the minimum threshold for the class performance rate. Therefore, we tuned the model by modifying the min\_child\_weight, learning\_rate, lambda, gamma, and previous parameters above. This tuning allowed us to get to a closer criteria. We were shy of the class 0 TPR >= 80% AND class 1 TPR >= 75%, so we decided to run the model again, while tweaking the initial training dataset. Since XGBoost already has a parameter to control for imbalanced classes, we thought that if the initial training dataset already has a balanced dataset, we could further improve the performance. Therefore, we tested the training dataset by over-sampling, under-sampling, and a bit of both. This drastically improved our model and we were able to reach the threshold.

## Decision Tree (Random Forest)

## The first algorithm implemented was the decision tree. We started with a simple Weka’s J48 decision model using all attributes. Using the J48() function, we had class (o\_bullied) as the explanatory variable and used the predict() function to find the performance of the model. Unfortunately, we did not see significant results. We continued down the journey and applied the PCA (Principal Component Analysis) to see if we could further data reduction and if this was the correct way to reduce. Using the prcomp() function, we excluded the class variable, centered and scaled the data. Gathering the summary of the data, we looked for 90% of the variability in the dataset for the first x principal components. In the first iteration of the decision tree, the first 46 principal components made up 90%. After this discovery, we ran the J48 decision model again to see if there were any improvements to the model, and found that the results were worse. PCA did not improve the prediction model and was ineffective altogether.

## 

## Then, we tried a more effective decision tree algorithm (random forest) and dropped PCA analysis. The random forest algorithm took the longest to run in the models we picked. The dataset was split into training and testing sets. We used multiple random forest algorithms to test against the dataset. First trained the random forest model by using trainControl() with parameters; method = “cv”, number = 5. Then, we used train() to train the training data set and set the method as “ranger” with 1000 trees. Once the model completed, we made predictions using the predict() function and evaluated the model’s performance with a confusion matrix.

## Logistic Regression

## The logistic regression model is the most basic one. After the data splitting, we ran glm() function with family=”binomial”. Then predict the model using predict() function. Compare predictions to the actual then we get confusion\_matrix. Then we extract TP,FN,FP,TN from the matrix to compute the measurements. As for ROC, we use roc()/auc() function passing y\_true(or actual) with the predictions to get area under roc. After computing for class0 and class1, we try to weight average the class to rebalance the o\_bullied values. Using glm(), we pass parameter weights = class\_weights to handle weighting. Then we ran it 10 times, however, the result in each run appears to yield the same, and not quite well enough. So we try to use ROSE package to oversample using method=”under” for this case, before running the model. The results yield better results.

## KNN (K-nearest neighbor)

Regarding the oversample using ROSE package, we continue to use method=”under” for KNN. For KNN evaluation, we ran the train model over 5 loops. For each loop, we use 10-fold cross-validation using trainControl(). Thus, we ran knnModel using train() with metric=”Accuracy” which is good for classification. preProcess=c(“center”,”scale”,”pca”) yield good results, and tunelength=10. Then, we ran prediction and confusion\_matrix as follows. As for the weight average, the train() function has parameter weights also. So we input weights=class\_weights. Next, we compute for other metrics for evaluations. The results yield slightly better for wt.average.

## Support Vector Machines

Regarding the oversample using ROSE package, we continue to use method=”under” for SVM.

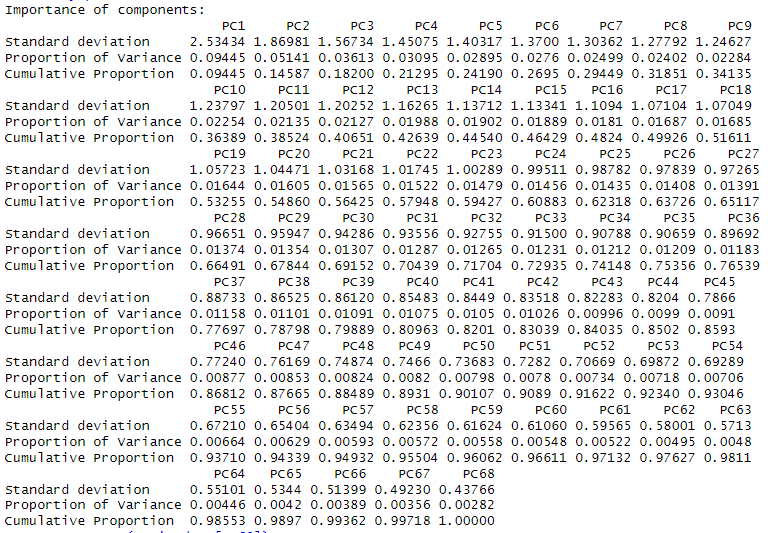
For SVM evaluation, we use 10-fold cross-validation using trainControl(). At first we try running over grid (sigma = seq(0.1, 0.4, by = 0.05), C = seq(1.0, 2, by = 0.1)) to cover the possible values. Then run a train() model with method=”svmRadial”, preProcess=c(“center”,”scale”,”pca”). Once we narrow down the better value we use a new grid. We ran it again, and compute for evaluation metrics. Also, with the same procedure with weights, by inputting the weights into the parameter of trian() function.

## Neural Networks

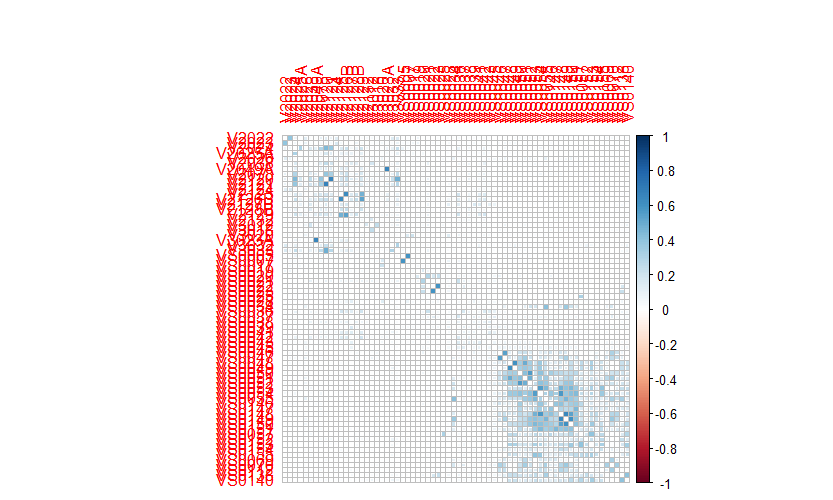
As for neural network, we cannot get the train(method=”nnet”) to be able to run. So we decide to use nnet() function instead. model <- nnet(o\_bullied ~ ., data = data\_train, size = 10, rang=0.7, decay=0.5, Hess=TRUE, maxit = 100) seems to yield good results for us. However, we cannot get a better model. Also, with the same procedure with weights, by inputting the weights into the parameter of nnet() function.

# Data Mining Results & Evaluation

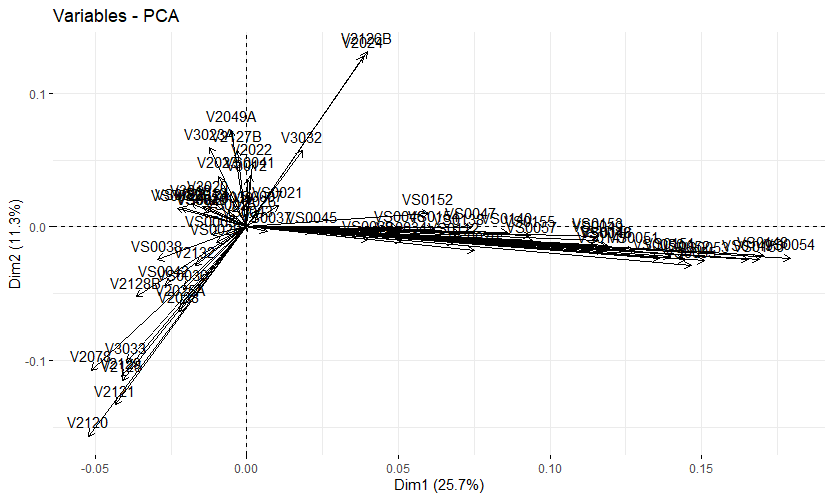
## Decision Tree (J48) + PCA



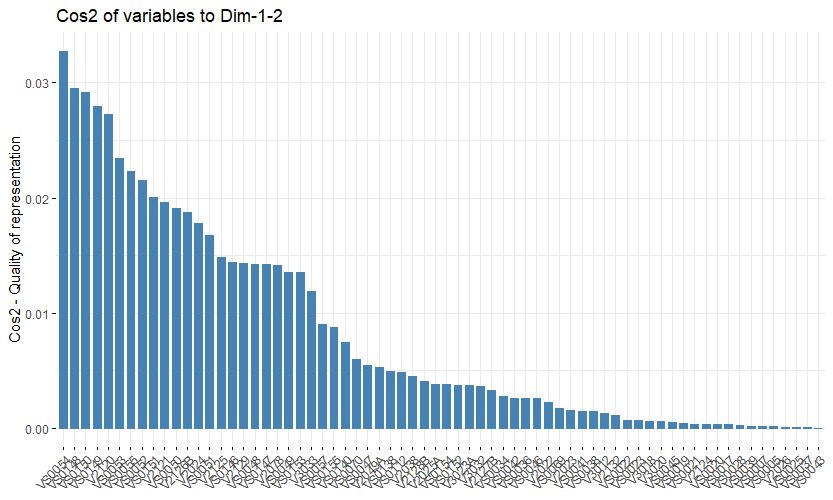
*PCA Analysis of filtered dataset*



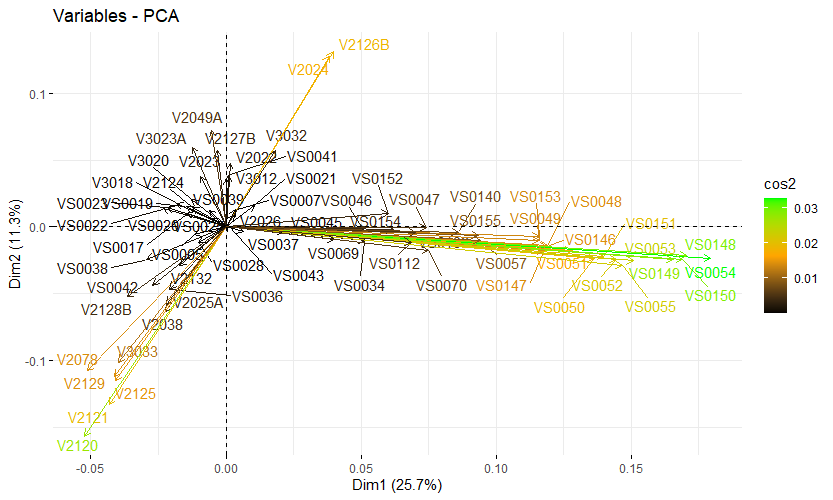
*Coorelation Analysis of filtered dataset*



*Variable Analysis of filtered dataset*



*Quality of representation of filtered dataset on variables*



*Variable Analysis of filtered dataset*

## Naive Bayes (bayes Theorem)

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 70.07% | 32.89% | 87.80% | 70.07% | 0.7794 | 0.6859 | 0.3209 | 0.2999 |
| Class 1 | 67.11% | 29.93% | 39.91% | 67.11% | 0.5005 | 0.6859 | 0.3209 | 0.2999 |
| Wt. Average | 67.11% | 29.93% | 39.91% | 67.11% | 0.5005 | 0.6859 | 0.3209 | 0.2999 |

## Boosting (Gradient Boosting, XGBoosting)

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 75.18% | 35.27% | 87.80% | 75.18% | 0.81 | 0.7694 | 0.3538 | 0.3408 |
| Class 1 | 64.72% | 24.82% | 43.57% | 64.72% | 0.5208 | 0.7694 | 0.3538 | 0.3408 |
| Wt. Average | 64.72% | 24.82% | 43.57% | 64.72% | 0.5208 | 0.7694 | 0.3538 | 0.3408 |

Parameters:

max.depth = 10, # the maximum depth of each decision tree

nround = 100, # number of boosting round

colsample\_bytree = 1,

subsample = 1,

early\_stopping\_rounds = 3, # if we dont see an improvement in this many rounds, stop

objective = "binary:logistic", # the objective function

scale\_pos\_weight = negative\_cases/postive\_cases, # control for imbalanced classes

min\_child\_weight = 10,

gamma = 0,

learning\_rate = 0.25

## Decision Tree (Random Forest)

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 75.18% | 34.27% | 88.13% | 75.18% | 0.8111 | 0.7045 | 0.3626 | 0.3483 |
| Class 1 | 65.73% | 24.82% | 44.1% | 65.73% | 0.5266 | 0.7045 | 0.3626 | 0.3483 |
| Wt. Average | 65.73% | 24.82% | 44.1% | 65.73% | 0.5266 | 0.7045 | 0.3626 | 0.3483 |

## Logistic Regression

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 72.66% | 36.87% | 86.94% | 72.66% | 0.7916 | 0.7266 | 0.3140 | 0.2995 |
| Class 1 | 63.13% | 27.34% | 40.61% | 63.13% | 0.4943 | 0.7266 | 0.3140 | 0.2995 |
| Wt. Average | 64.72% | 29.77% | 39.17% | 64.72% | -/4880 | 0.7274 | 0.3027 | 0.2842 |

## KNN (K-nearest neighbor)

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 66.38% | 47.27% | 82.58% | 66.38% | 0.7360 | 0.5956 | 0.1653 | 0.1550 |
| Class 1 | 52.73% | 33.62% | 31.72% | 52.73% | 0.3961 | 0.5956 | 0.1653 | 0.1550 |
| Wt. Average | 53.05% | 33.53% | 31.90% | 53.05% | 0.3985 | 0.5976 | 0.1689 | 0.1583 |

## Support Vector Machines

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 67.40% | 34.22% | 86.93% | 67.40% | 0.7593 | 0.6659 | 0.2842 | 0.2620 |
| Class 1 | 65.78% | 32.60% | 37.41% | 65.78% | 0.4769 | 0.6659 | 0.2842 | 0.2619 |
| Wt. Average | 66.05% | 33.07% | 37.16% | 66.05% | 0.4756 | 0.6649 | 0.2819 | 0.2589 |

## Neural Networks

|  | TPR | FPR | Precision | Recall | F-measure | ROC | MCC | Kappa |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class 0 | 71.78% | 34.32% | 87.60% | 71.78% | 0.7890 | 0.7524 | 0.3263 | 0.3085 |
| Class 1 | 65.68% | 28.22% | 40.82% | 65.68% | 0.5034 | 0.7524 | 0.3263 | 0.3085 |
| Wt. Average | 66.68% | 29.73% | 39.99% | 66.68% | 0.4995 | 0.7479 | 0.3199 | 0.2993 |

# Data Mining Discussion & Conclusion

In this section, we will describe how we strategically explored different ways to model classification and found the best method. Each classification method was built and run several times. Some models took longer than others, but every model that we chose initially performed horribly. With our initial dataset of 90 variables, it was clear that in order to achieve high performance; preprocessing, selecting attributes and creating a balanced training data was a necessary first step. Once we got to classification, we had to reduce the 90 variables somehow, and that reasoning is discussed in this section. It is also clear that classification models are not perfect and that the preparatory data work beforehand is crucial to training your model and predicting results. The development and implementation of classification models for predicting bullying data was an effort that required analysis and rework in order to meet the minimum requirements of class 0 TPR >= 75% AND class 1 TPR >= 65%.

As mentioned in the Data Mining procedure, a critical step we took was implementing a balanced training dataset. Beforehand, we ran classification models with an unbalanced training dataset and consistently received high Class 0 TPR and extremely low Class 1 TPR. This was our first discovery in the dataset. Even the best model that is run thousands of times would be unable to predict a dataset if the training is done on an unbalanced dataset. Once we corrected the balanced training dataset for under-sampling which is over-representing the minority class (bullied individuals), we found significantly better results from training, and our models were able to predict both classes consistently. The ROSE package provided the functions to deal with binary classification problems in the presence of imbalance classes. Artificial balanced samples are generated according to a smoothed bootstrap approach and allow for aiding both the phases of estimation and accuracy evaluation of a binary classifier in the presence of a rare class. Imbalanced learning was a key discovery that ultimately led to our findings of the best classification model.

As mentioned in the preprocess and data mining procedure sections, another forward step we took after the intermediate report was implementing attribute selection methods. We deduced that variable inflation factor was a component that needed to be introduced to the dataset. VIF was able to reduce multicollinearity which could have negatively affected our model. Due to the number of predictors in our dataset, we knew this would be a problem for modeling. Once the model was reduced down to 69 predictors, we ran our classification models again and received an improvement on our results. However, we still did not achieve high performance. Even parameter tuning at this point would not be in favor due to the variability of testing other datasets. We would not be able to replicate performance on independent datasets if we continued with our models. Therefore, we needed to reduce dimensionality further.

We proceeded with Principal Component Analysis by finding that selecting the first 49 principal components yielded 90% of variability of the dataset. However, we still received results that were unfavorable, and greatly reduced the performance of our model. After diving deeper into the PCA graphs and results, it was clear that PCA should be used mainly for variables which are strongly correlated. Since the relationship between the variables was weak, PCA did not work well to reduce the data and we dropped PCA as a dimensionality reduction.

Still, we had an issue with performance on the 69 variables, but there were more effective methods to reduce dimensionality.

Information gain as an attribute selection worked in our favor. With all our models before information gain, we were getting inconsistent results. Similar to the problem before, we kept receiving high sensitivity and low specificity. Out of the 7 classification models we investigated, 2 achieved high performance after information gain attribute selection. Random Forest and XGBoost were able to pass the threshold of class 0 TPR >= 75% AND class 1 TPR >= 65%. This was the clear winner in our dataset, as every model improved performance with the reduction of predictors. Information gain removed 30 variables that contributed 0 value to the dataset. We were left with 39 predictors in the dataset. Once we narrowed our dataset down to these predictors, we began to see promising results.

We chose the classification models based on what was learned in lecture. We knew that some models were not going to produce the best results, but we picked ones that would be effective in categorical data. The best algorithms became apparent, and they followed concepts that we hypothesized would be successful. XGBoost and Random Forest are both ensemble methods, meaning they build multiple models and combine their predictions to make a more accurate and robust prediction. Both algorithms use a form of bagging (bootstrap aggregating) to create a subset of the training data for training individual models. In random forest, each tree is trained on a bootstrap sample of the data, and in XGBoost, each boosting round is trained on a subset of the data. Both XGBoost and Random Forest are less prone to overfitting compared to individual decision trees. Once we found these two algorithms to be the best, parameter tuning was the final step to tweak the models for the best results.

It is important to note that parameter tuning is very crucial performance, however, tuning for one model on a data set could differ from others. That is why we left parameter tuning as a means of final correction on the model, and tried to find optimal performance with a balanced training dataset, attribute selection, and finding the best classification algorithm.

XGBoost required the most tuning out of the 2 models. We tried a variety of hyperparameter tuning with modifying the following parameters: max.depth, nround, colsample\_bytree, subsample, early\_stopping\_rounds, min\_child\_weight, gamma, scale\_pos\_weight and learning\_rate. It is worth mentioning that scale\_pos\_weight is a built in parameter that can correct class imbalance. We saw this with the first iteration of the dataset (90,69 predictors) and before ROSE sampling where XGBoost yielded very favorable results. At one point we passed the class 0 TPR >= 80% AND class 1 TPR >= 70%, however, it was not a consistent result. Tuning is a practice and a bit of a gamble, we would test various results that would not move the needle too far. However, max.depth, nround, subsample, and learning\_rate were by far the most effective. For max.depth, deeper trees can capture more complex patterns but may lead to overfitting. For nround, these were the amount of boosting rounds, and a higher number can lead to better performance but risk overfitting. For subsample, a value of less than 1.0 introduces randomness and helps prevent overfitting, a too low value might lead to underfitting. Learning rate makes the model more robust but requires more boosting rounds. Again, hyperparameter tuning is an art form, but in areas where we increased one parameter, we tried to correct the other way by decreasing another. We also tested gamma and lambda parameters that would help in regularization.

In conclusion, this project has been a true force in illuminating the capabilities of classification. Classification models stand as fundamental tools in machine learning, playing a pivotal role in solving a wide array of real-world problems. These models harness the power of data to categorize and classify unseen instances based on learned patterns. The effectiveness of a classification model hinges not only on the choice of algorithm, but also on feature engineering, data preprocessing, and hyperparameter tuning. Finding the balance between model complexity and interpretability is a consistent challenge in the domain. It has been clear that machine learning continues to evolve, and classification will be at the forefront to handle increasingly complex tasks. Today, we were able to predict bullying, but applications on other data opens the door for opportunities. It is important to understand the limitations and potential pitfalls of these models in order to prevent overfitting, underfitting and wrong interpretations of the data.

# Division of Work

Classification Models:

**Jonathan**

Decision Tree (J48 + PCA)

Naive Bayes  
Boosting

Decision tree

**Theerarun**

Logistic Regression

KNN

Support Vector Machines

Neural Networks

# Appendix

Lunardon, Nicola, Giovanna Menardi, and Nicola Torelli. "ROSE: A Package for Binary Imbalanced Learning." R Journal, vol. 6, no. 1, 2014,<https://journal.r-project.org/archive/2014/RJ-2014-008/index.html>.