MSIA 420 Predictive Modeling Project

Don’t Get Kicked!

Predicting if a Car Purchased at an Auction is a Bad Buy

Shiyi Chen
Ameer Khan
Xiang Li
Steven Lin
Sanjeevni Wanchoo

Abstract

Auto community calls an unfortunate purchase a "kick" if an auto dealership purchases a used car with serious issues at an auction that prevents it from being sold to customers. In this report, 6 models were used to predict which cars had a high risk of being “kicked”. These models included logistic regression, decision tree, neural network, random forest, gradient tree boosting, and an ensemble method. Different criteria for model evaluation, such as misclassification rate and the Gini impurity index, were used.

External demographic data was also leveraged to improve the predictive power of the models. However, the vehicle purchase data played a more important role than the demographic information in predicting outcomes. The WheelType variable was the most critical predictor across different models that determined if a vehicle was a bad buy. The results also indicate misclassification errors of around 10% and Gini indices around 0.20. Based on the Gini index, Gradient Tree Boosting and Ensemble Model (Maximal Voting) were found to be the best models in predicting the probability of a car being a bad buy with Gradient Tree Boosting giving a misclassification rate of 10.11% and Gini index of 0.2357.

1. Introduction

One of the biggest challenges of an auto dealership purchasing a used car at an auto auction is the risk that the vehicle might have serious issues that prevent it from being sold to customers. The auto community calls these unfortunate purchases "kicks".

Kicked cars often result from issues like tampered odometers, difficult-to-address mechanical problems, difficulty obtaining vehicle title from seller, or some other unforeseen problem. Kicked cars can be very costly to dealers after accounting for additional transportation costs, throwaway repair work, and market losses in reselling the vehicle.

A predictive model that can figure out which cars have a higher risk of being kicks can provide real value to dealerships, who try to provide the best inventory selection possible to their customers. [Source: http://www.kaggle.com/c/DontGetKicked]

The goal of the project is to determine if a car purchased at an auction is a kick, and hence a bad buy.

2. Data Preprocessing

The original dataset consisted of 72,000 observations and 34 attributes, with each row representing a purchase transaction of a car at an auction and each column a different attribute of the car or the purchase, such as the make, vehicle age, purchase date, etc. The full list of the attributes can be found in Appendix A.

Training and test datasets were created by applying a random 70/30 split on the original dataset. The resulting training set had about 51,000 observations. Both the training and test set were verified to have same overall statistics for majority of the attributes. The following data processing steps were taken to prepare the data for model fitting:

- The strategy for using categorical variables was to maintain a balance between the information gained and the increase in complexity. For this purpose, for example, we chose to include the make of a car (33 categories), but excluded the vehicle model (1063 categories).
- Some categories were combined for certain predictors to reduce complexity. For instance, the state where the sale took place (VNST), has most records for southern and western states of the US. The Midwest and Eastern states were combined into regional categories in a new variable called region, while others were retained as-is since they had substantial number of observations.
- About 315 observations pertaining to missing Manheim Market Report (MMR) price fields were removed. The distribution of good and bad buys amongst these observations was compared with that of the original dataset, and found to be very similar. This helped us confirm that the missing data was not systematically biased in any way. Additionally, if a make category contained three or fewer observation, it was removed. About five observations were removed in this way.
- Vehicle Age (VehicleAge) and Vehicle year (VehYear) sum up to the year of purchase date (PurchDate). Furthermore, the purchases in our dataset were limited to 2009-2010 period. Hence, only VehicleAge was used as a predictor.
- The variable *WarrantyCost* was log-transformed, as it made the distribution closer to normal, and remedied the right-skewness.
- MMR prices are correlated and provide the same information. Thus, a new variable called *Differential* was created as the percentage difference of the retail price and vehicle price.

$$Differential = \frac{VehBCost - MMRAcquisitionRetailPrice}{MMRAcquisitionRetailPrice}$$

- In order to consider the effect of the location of a sale, the Zip code of the purchase transaction was linked to external demographic data (e.g. income) from the census.

3. Exploratory Analysis

Before fitting various predictive models, an exploratory analysis was performed by computing descriptive statistics to provide a quantitative and graphical summary of the data, detect patterns, assess the nature of the relationship between attributes, and to help avoid incorrect assumptions for model fitting. For the categorical variables, the descriptive statistics included frequencies and proportions for the various categories of the variables. For continuous variables, the descriptive analysis consisted of computing central tendency and distribution measures (e.g. skewness). All outputs from this section can be found in Appendix B.

Table 1: Descriptive Statistics for Continuous Variables

Attribute	Mean	Std. Dev.	Median	Min	Max
VehYear	2005.34	1.73	2005	2001	2010
VehicleAge	4.18	1.71	4	0	9
VehOdo	71500	14578.91	73361	4825	115717
MMRAcquisitionAuctionAveragePrice	6128.91	2461.99	6097	0	35722
MMRAcquisitionAuctionCleanPrice	7373.64	2722.49	7303	0	36859
MMRAcquisitionRetailAveragePrice	8497.03	3156.29	8444	0	39080
MMRAcquisitonRetailCleanPrice	9850.93	3385.79	9789	0	41482
MMRCurrentAuctionAveragePrice	6132.08	2434.57	6062	0	35722
MMRCurrentAuctionCleanPrice	7390.68	2686.25	7313	0	36859
MMRCurrentRetailAveragePrice	8775.72	3090.7	8729	0	39080
MMRCurrentRetailCleanPrice	10145.39	3310.25	10103	0	41062
VehBCost	6730.93	1767.85	6700	1	45469
WarrantyCost	1276.58	598.85	1155	462	7498

The descriptive statistics for the continuous variable WarrantyCost show large positive skewness (right-skewed distribution), suggesting that a transformation is needed to symmetrize and reduce the range as well. The distribution of the variable warranty cost as well as its log transformation is shown below.

Figure 1: Histogram of WarrantyCost and its transformation

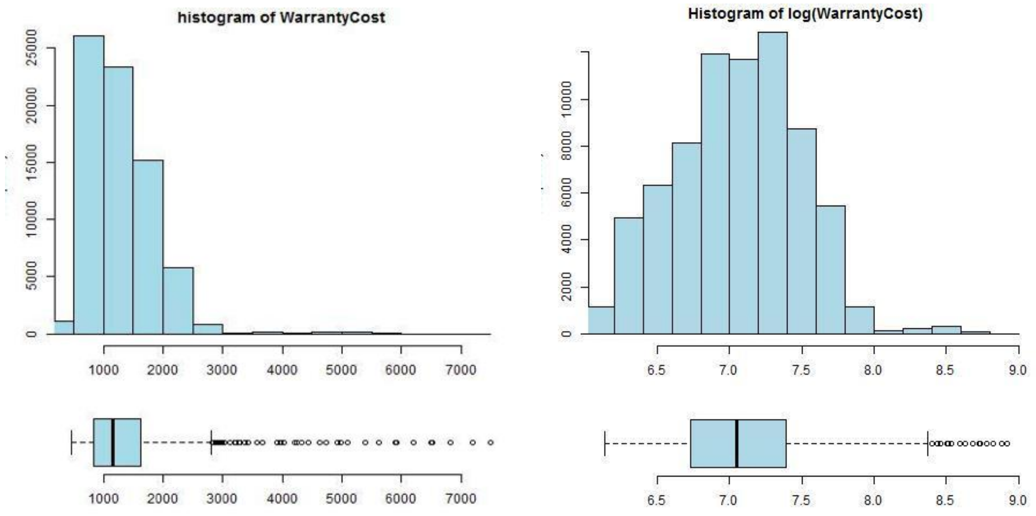


Table 2: Descriptive Statistics for Categorical Variables

Categorical Variables	Levels
IsBadBuy	2
Auction	3
Make	33
Model	1,063
Trim	135
SubModel	864
Color	17
Transmission	5
WheelTypeID	5
WheelType	4
Nationality	5
Size	13
TopThreeAmericanName	5
PRIMEUNIT	3
AUCGUART	3
VNST	37
IsOnlineSale	2

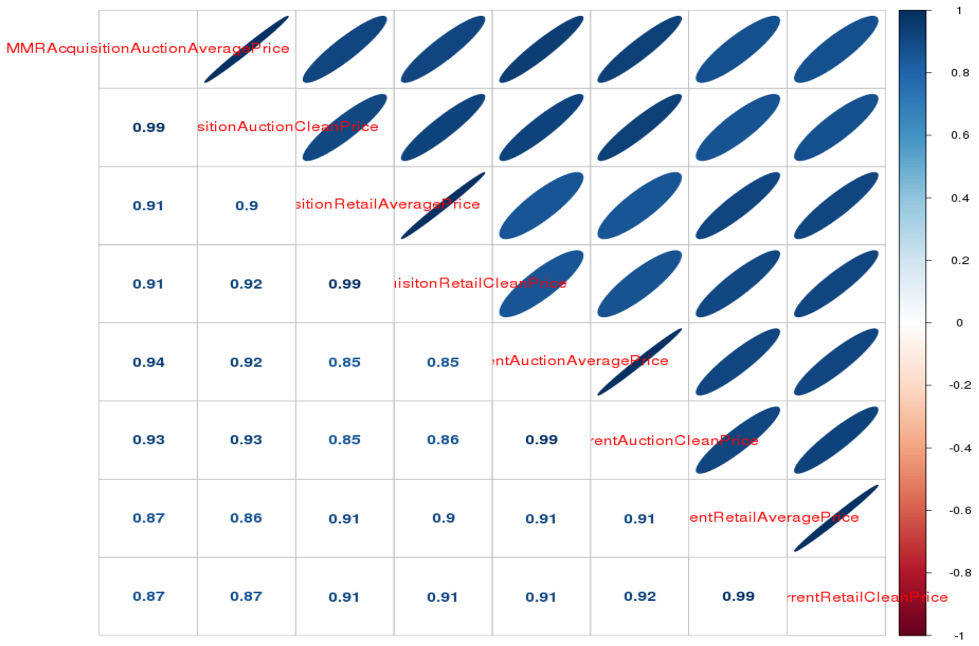
The response variable, IsBadBuy, is imbalanced in the sense only 12.3% of the vehicle purchases are marked as bad buys. Thus, misclassification rates of the model should be interpreted with caution as they can be misleading. From the dealer's perspective, false negative and false positives might be a more important consideration than overall prediction. This issue is discussed in detail in the next section. The table below shows the frequency for the response variable. The frequency tables of other categorical variables can be found in the Appendix.

Table 3: Frequency of Response Variable

IsBadBuy	Frequency	Proportion
0	64007	87.70%
1	8976	12.30%
Total	72983	

A correlation plot was also used to assess the relationship between the price variables. The figure below shows the correlation plots for MMR price variables. The correlations between MMR price variables all are above 0.85, indicating that they are highly correlated with each other. This was a consideration for model fitting, as discussed in the next section.

Figure 2: Correlation plot of price variables



4. Model Fitting

Several predictive models were used to predict a bad buy at an auction based on various characteristics of a vehicle purchase at an auction. The model diagnostics and results for some of the attempted models are listed in the following sections. The misclassification rates for all models have been computed at a cut-off of 0.5.

Based on the exploratory analysis, and the trade-off relevant to categorical variables for large number of categories (discussed in the data preprocessing section), the following predictors were used for modeling:

- Continuous Predictors: VehicleAge, VehOdo, VehBCost, WarrantyCost, Differential
- Categorical Predictors: Auction, Make, Color, Transmission, WheelType, Nationality, Size, AUCGUART, PRIMEUNIT, IsOnlineSale, region, TopThreeAmericanName

The data set has only 12% cases of positive bad buys or kicks, which means that the values for the binary response variable IsBadBuy are mostly zero. In this case, the Gini index is a better method to evaluate a model's performance on the test set rather than the misclassification error, since we are interested in the occurrence of the less frequent event. A model can achieve low misclassification rates by categorizing majority of the predicted responses as 0s or good buys. The Gini impurity index is analogous to the area under the ROC curve (AUC) for a binary response variable. It is a better indicator of the model performance since it balances precision and recall, which are more relevant to the problem at hand.

4.1 Logistic Regression

Logistic regression model is used to model binary outcome variable. In the model, the log-odds of the binary outcome is modeled as a linear combination of the predictors.

An initial model was fitted and stepwise regression was used to select potential significant predictors. Based on the result of stepwise regression, insignificant variables were dropped from the model. After these changes, a third model was fit and all predictors in the model were significant at the 5% level. Next, we checked if any multicollinearity existed among the predictors. The variance inflation factor (VIF) was computed for the model and variables with a VIF higher than 10 were dropped. The coefficients of the final model are indicated below:

Coefficients:				
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	5.915e-01	1.240e-02	47.691	< 2e-16 ***
VehicleAge	2.680e-02	9.961e-04	26.906	< 2e-16 ***
WheelTypeAlloy	-5.957e-01	6.585e-03	-90.475	< 2e-16 ***
WheelTypeCovers	-6.068e-01	6.618e-03	-91.700	< 2e-16 ***
WheelTypeSpecial	-5.730e-01	1.445e-02	-39.657	< 2e-16 ***
VehOdo	4.414e-07	1.008e-07	4.379	1.2e-05 ***
VehBCost	-9.692e-06	8.237e-07	-11.767	< 2e-16 ***
median_income	3.451e-07	7.816e-08	4.416	1.01e-05 ***
Differential	-5.677e-02	7.024e-03	-8.081	6.54e-16 ***

The odds ratio of the coefficients of the fit is given below:

	OR	2.5 %	97.5 %
(Intercept)	1.8066965	1.7633067	1.8511540
VehicleAge	1.0271627	1.0251594	1.0291700
WheelTypeAlloy	0.5511591	0.5440919	0.5583182
WheelTypeCovers	0.5450658	0.5380417	0.5521817
WheelTypeSpecial	0.5638131	0.5480693	0.5800091
VehOdo	1.0000004	1.0000002	1.0000006
VehBCost	0.9999903	0.9999887	0.9999919
median_income	1.0000003	1.0000002	1.0000005
Differential	0.9448144	0.9318958	0.9579120

The most important predictor is WheelType, which affects the odds of a purchase being bad the most. The base group of WheelType is NULL. The coefficient of WheelType Alloy is 0.55 and it tells odds of a bad buy decreases by 45% for Alloy vs. NULL. The coefficients of WheelType, VehOdo and Differential are negative. It means these variables have a negative relationship with odds of a purchase being a bad buy. A car with a known WheelType is significantly less likely to be a kick. VehOdo is the mileage of a car when it was purchased and it has a positive coefficient. The result supports our assumption which is the more mileage a car has, more likely that it is a bad buy.

For the training set, the misclassification rate is 10.45%. The test set misclassification rate is 10.67%. The model accuracy is measured by the area under the ROC curve. The AUC of the final logistic model is 74.12%. The Gini index for the test set is 0.2114.

Table 4: Model Statistics for Logistic Regression

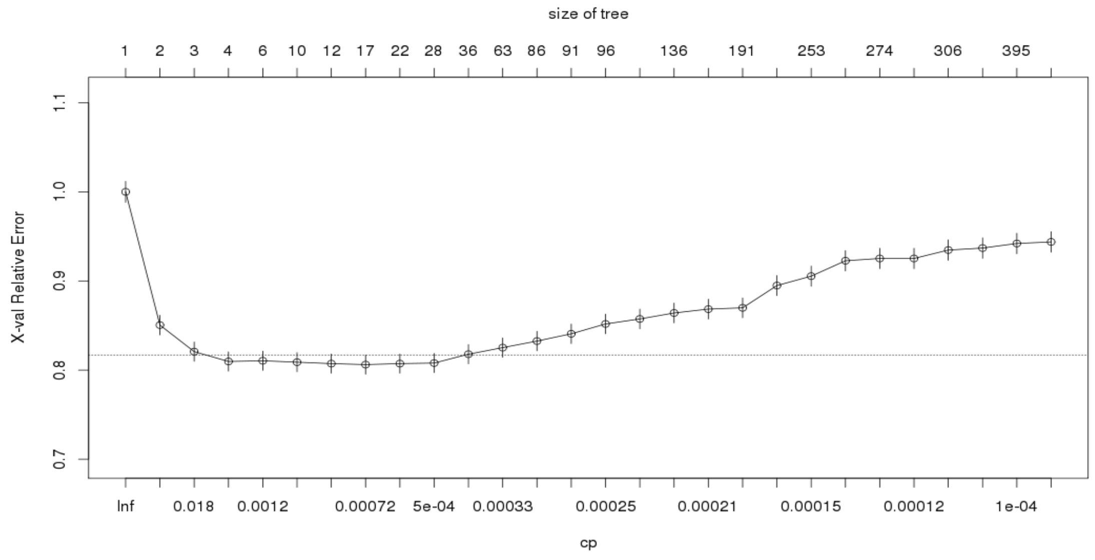
Training Misclassification Rate	10.45%
Test Misclassification Rate	10.67%
Test Gini Index	0.2114
Test AUC	74.12%

4.2 Decision Trees

Decision tree models are not sensitive to the distribution of predictor variables and can handle missing values, so there is no need to do perform any variable transformation. We fit a tree model using the rpart package in R and looked for the best tree with the minimum cross-validation error. The minimum CV-error was obtained for tree 8 with 16 splits. Refer to Table A1 for cross-validated errors of different complexity factors. The training misclassification rate for the best tree is 0.09767. Variables actually used in tree construction are (in the order of importance): WheelType, VehicleAge, MMRAcquisitionAuctionAveragePrice, region, Auction, Differential, Make, Transmission, AUCGUART, WarrantyCost, VehOdo, VehBCost, and Size.

The best-fitted tree model was benchmarked on the test dataset and scored a misclassification rate of 10.07% and Gini index 0.1819.

Figure 3: The Cross-Validation Deviance vs. the Size of Tree



The pruned decision tree model is shown below:

Figure 4: Pruned Classification Tree

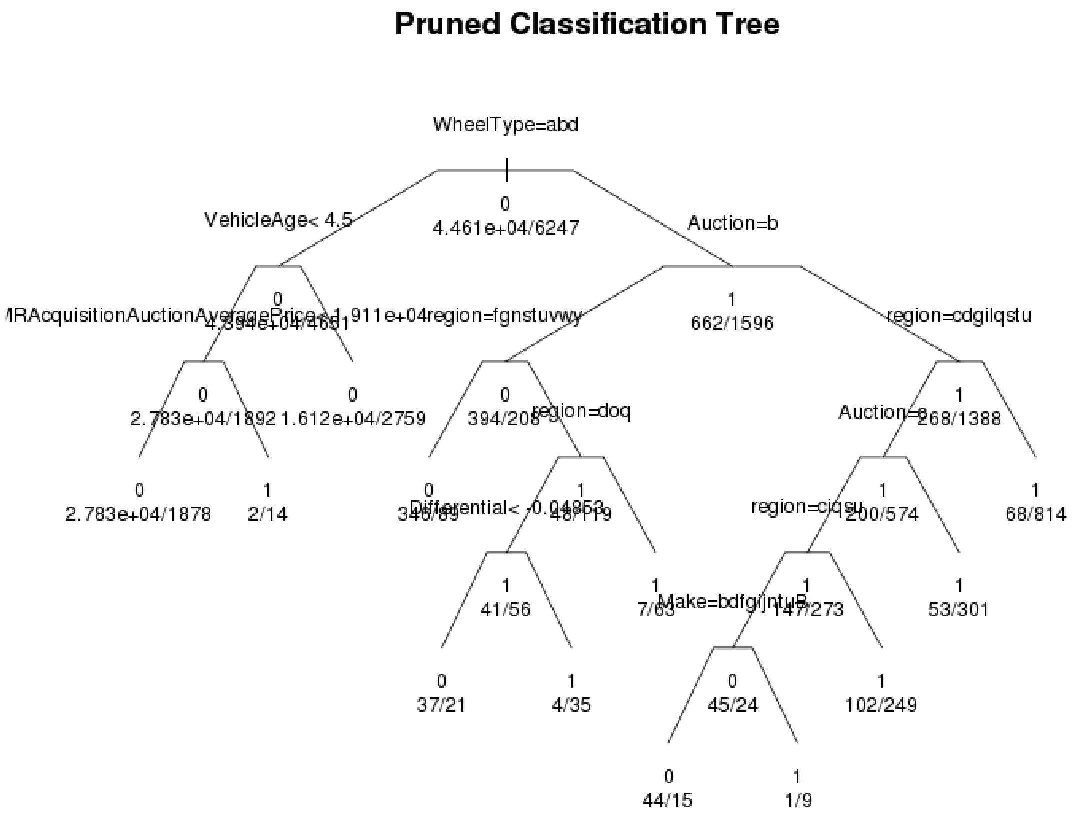


Table 5: Model Statistics for Decision Tree

Training Misclassification Rate	9.77%
Test Misclassification Rate	10.07%
Test Gini Index	0.1819041
Test AUC	70.66%

4.3 Neural Networks

In order to fit a neural network to our dataset, the continuous variables were standardized. Due to the inherent complexity and long runtimes associated with fitting neural network models across several values of nodes and lambda, the analysis was limited to a subset of such combinations. For the same reason, the external demographic predictors were excluded from this model.

The number of hidden nodes and the decay parameter (lambda) were obtained using 5-fold cross-validation with five repetitions. Two criteria can be used to determine the best combination of hidden nodes and lambda. In the training set, number of nodes = 3 and lambda = 0.5 gave us lowest misclassification rate, while number of nodes = 2 and lambda = 0.1 gave the highest Gini.

The area under the curve (AUC), and other statistics for the test set were calculated using the first model mentioned above (nodes = 3, lambda = 0.5). These are reported in the table below:

Table 6: Model Statistics for Neural Networks

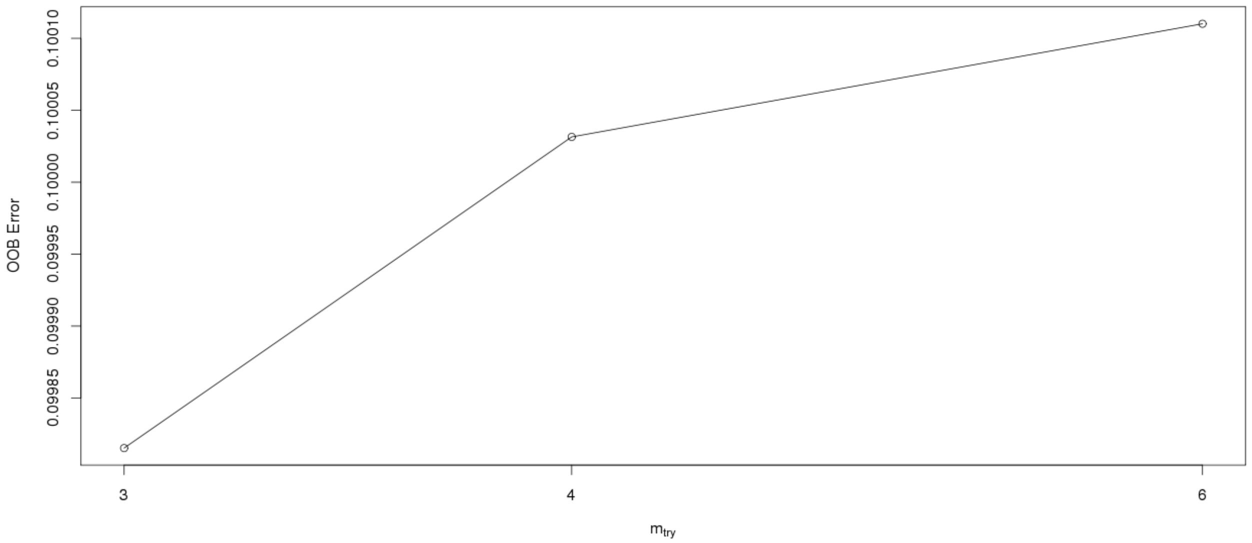
Training Misclassification Rate	9.99%
Test Misclassification Rate	10.19%
Test Gini Index	0.2291166
Test AUC	76.14%

4.4 Random Forest

A random forest model was used to predict the probability of the vehicle purchase being a bad buy. In random forests, the estimate of the test error is computed internally during the runs by using different bootstrap samples from the original data and leaving cases out of the bootstrap sample for fitting a tree in each run. Therefore, there is no need to perform cross-validation to address complexity and over-fitting.

Out-of-bag (OOB) error estimates are reported instead, which were used to find the optimal parameters. The first step was to find the optimal value (with respect to out-of-bag error estimate) for the number of variables randomly sampled as candidate at each split (called mtry in R function randomForest) by testing different values for the number of trees to grow at each tuning step using the tuneRF function. The optimal was consistently mtry $= 3$ for different different values for the number of trees to grow as shown in the figure below.

Figure 5: The OOB Error vs. the Optimal Number of Predictors for Each Tree mtry



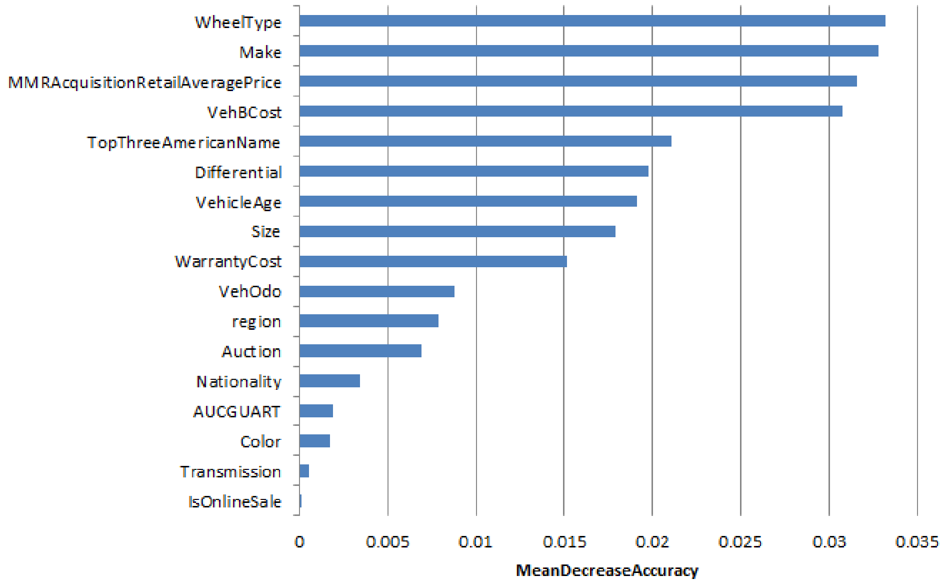
The next step was to run 10 repetitions, varying the number of trees (ntree) and the minimum size of terminal node (nodesize). The results indicated that ntree $= 300$ and nodesize $= 5$ performed the best on average. The summary of the results of the different performance criteria are shown in the table below.

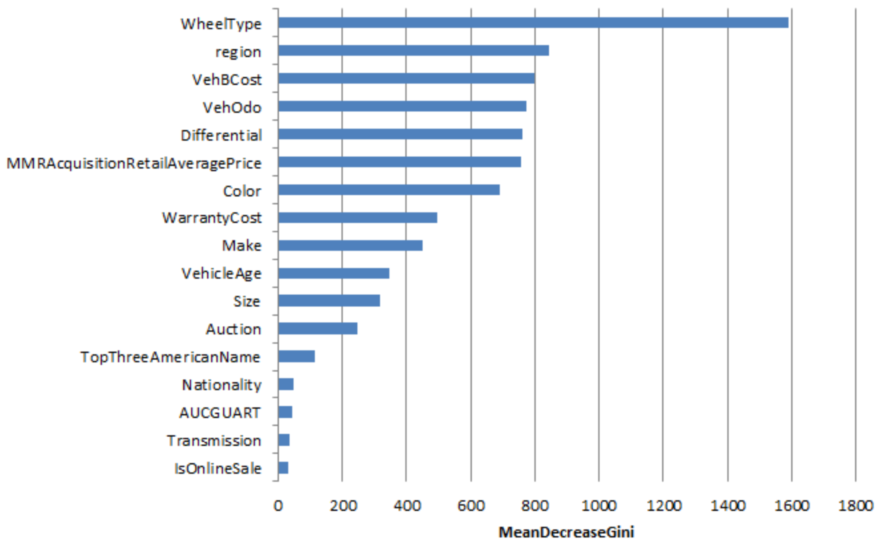
Table 7: Model Statistics for Random Forest

Training Misclassification Rate	10.00%
Test Misclassification Rate	10.19%
Test Gini Index	0.2186
Test AUC	74.94%

The variable plot also shows the importance of the variables in terms of the mean decrease in accuracy over all classes and the mean decrease in Gini index. By far the most important variable is the Wheel Type. The price, vehicle cost and differential predictors also seem to be important in predicting whether a car is a good buy or not.

Figure 6: Random Forest Variable Importance Plots





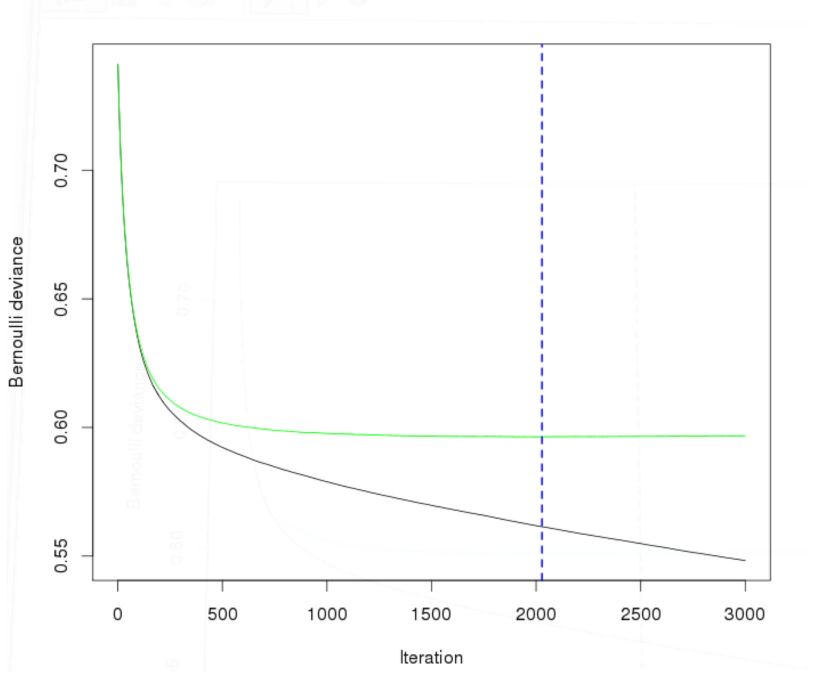
4.5 Gradient Tree Boosting

For the gradient boosting model, the parameters were chosen by 10-fold cross-validation with 5 repetitions. The range of the interaction depth explored was from 2 to 8 and the shrinkage parameter ranging from 0.01 to 0.1. The minimum cross-validation deviance with 10-fold CV was 0.5963, obtained when the interaction depth was 6 and the shrinkage parameter was 0.01.

The external demographic predictors were excluded from the modeling in order to reduce the complexity of the model.

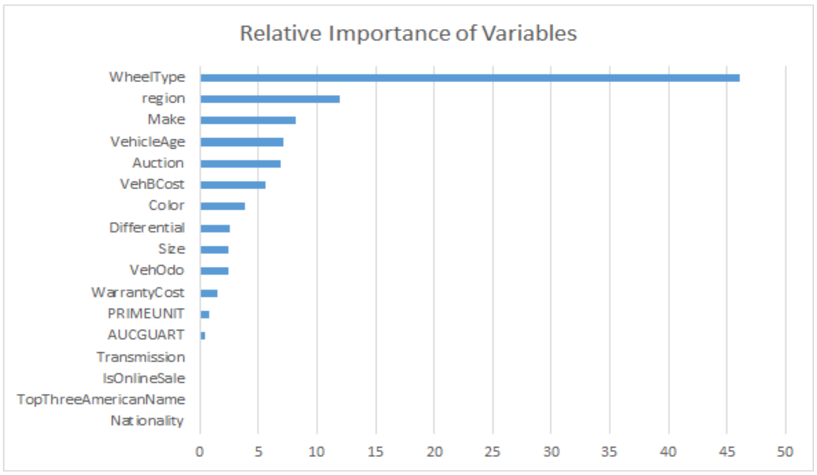
When the full model was fit for CV error for this configuration of parameters flattened out when the number of trees was 2082. The plot of the Bernoulli deviance against the number of trees is given below.

Figure 7: Bernoulli Deviance vs. Number of Trees for Gradient Boosting



The relative influence of the variables is given below:

Figure 8: Relative Influence of Variables for Gradient Boosting



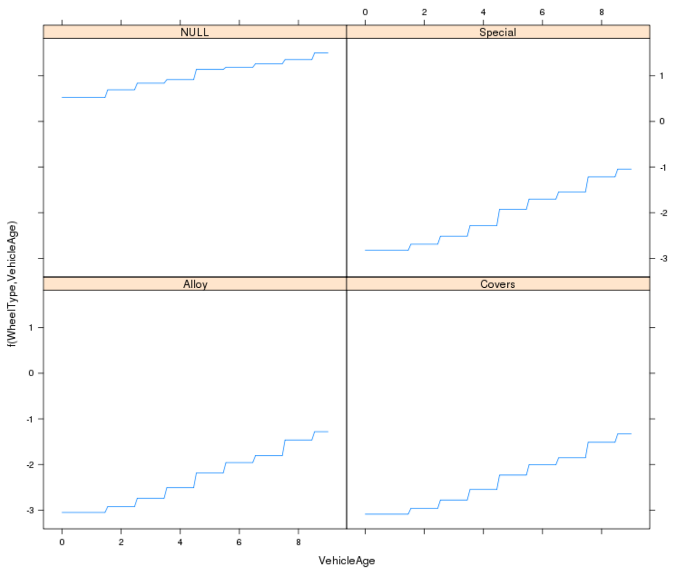
The statistics of the gradient boosted model are tabulated below:

Table 8: Model Statistics for Gradient Tree Boosting

Training Deviance	0.5965303
Training Misclassification Rate	9.63%
Test Misclassification Rate	10.11%
Test Gini Index	0.2357035
Test AUC	76.9%

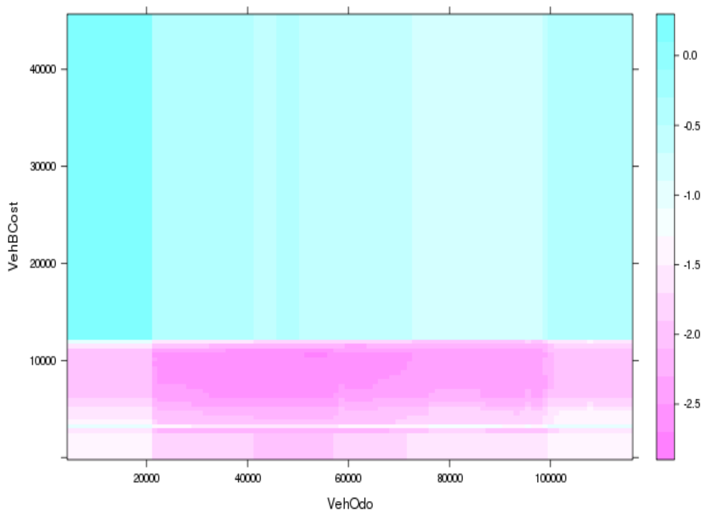
The marginal plots of the final model is given below:

Figure 9: Wheel Type vs. Vehicle Age Marginal Plot



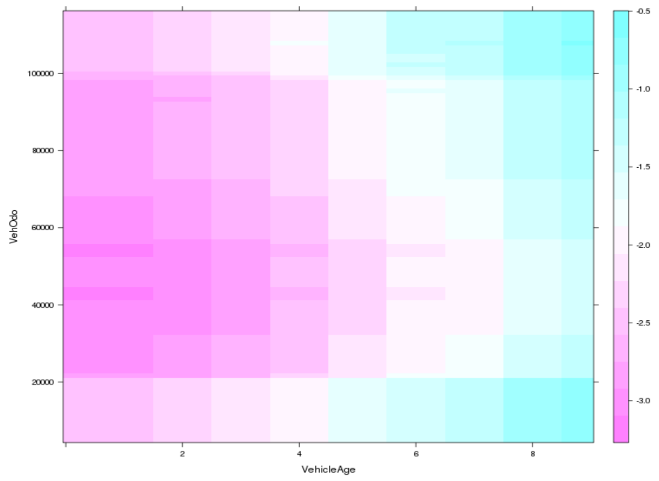
The plot above shows that the probability of a car being kicked increases with its age, but it is especially higher when the wheel type is not recorded. The wheel type is the most important predictor according to the gradient boosting model to determine if the car is a bad buy.

Figure 10: Vehicle Cost vs. Vehicle Odometer Marginal Plot



This plot indicates that when the price of the vehicle is below $11,000 and the odometer reads between 20,000 and 70,000 miles, it is generally a good buy. The plot also says that vehicles priced over $11,000 generally have higher probability of getting kicked, with the highest probability being for cars with the odometer reading either below 20,000 or greater than 100,000 miles. This plot also indicates interaction between the vehicle cost and the mileage since there is a thin strip of high chance of a bad buy in the pink region.

Figure 11: Vehicle Odometer vs. Vehicle Age Marginal Plot



The plot above shows that the probability of a bad buy is high when the vehicle is old. The probability is the maximum when the vehicle is more than 8 years old, and has less than 20,000 miles or more than 100,000 miles on the odometer. This can be explained as old cars with low mileage most likely have tampered odometers.

4.6 Ensemble Model

The team attempted an ensemble method for the given dataset, combining the top two atomic models obtained based on the test Gini index - the gradient tree boosting model and the neural network, using the optimal parameters obtained for each.

The gradient tree boosting has an interaction depth of 6 and shrinkage of 0.01. Since there were several repetitions for the model, cross-validation was not used. Instead, the best tree size was computed using the out-of-bag error for each run of the gradient boosting tree. The OOB error is analogous to the CV error as it is computed using a fraction (in this case 50%) of the training dataset that was not used for fitting.

The neural network has 3 nodes in the hidden layer, and shrinkage 0.5. Each model was run 200 times with a random sample half the size of the training dataset chosen randomly for each run. The predicted probabilities vector $\widehat{p_{ij}}$ is calculated for every run j of a model i, and the mean probabilities $\hat{p}_{gbm}$and $\hat{p}_{nnet}$ are computed as $\hat{p}_i = \frac{1}{N}\Sigma_j^N \hat{p}_{ij}$.The final probability $\hat{p}$ vector was obtained using two voting methods: selecting the maximum probability from the two mean probability vectors $\hat{p}_{gbm}$and $\hat{p}_{nnet}$for each observation, or the average of the two vectors.

The results for the maximal voting mechanism are tabulated below:

Table 8: Model Statistics for Ensemble Method with Maximal Voting Mechanism

Test Misclassification Rate	10.21%
Test Gini Index	0.232091
Test AUC	76.48%

The results for the average voting mechanism are tabulated below:

Table 9: Model Statistics for Ensemble Method with Average Voting Mechanism

Test Misclassification Rate	10.13%
Test Gini Index	0.2316595
Test AUC	76.43%

The maximal voting mechanism performs slightly better than average voting, but still underperforms gradient boosting on its own.

5. Results and Discussion

The misclassification error and the Gini index on the test dataset for each of the models are reported below. The best model is selected on the basis of the maximum Gini value on the test set, as it is a more robust indicator of performance. As discussed in previous sections, the misclassification error might not be a good measure in this case. The results confirmed this as weak learners also performed better on this measure.

Based on the Gini index, it seems gradient tree boosting is the best model in predicting the probability of a vehicle at an auction being a kick. Across different models, the vehicle data played a more important role than the demographic information in outcome predictions. More specifically, WheelType was found to be consistently the most important predictor for bad buys.

Table 10: Summary of Models

Model	Misclassification Error	Gini Index
Logistic Regression	10.67%	0.2114
Decision Tree	10.07%	0.1819
Neural Networks	10.19%	0.2291
Random Forest	10.19%	0.2186
Gradient Tree Boosting	10.11%	0.2357
Ensemble Model (Maximal Voting)	10.21%	0.2321

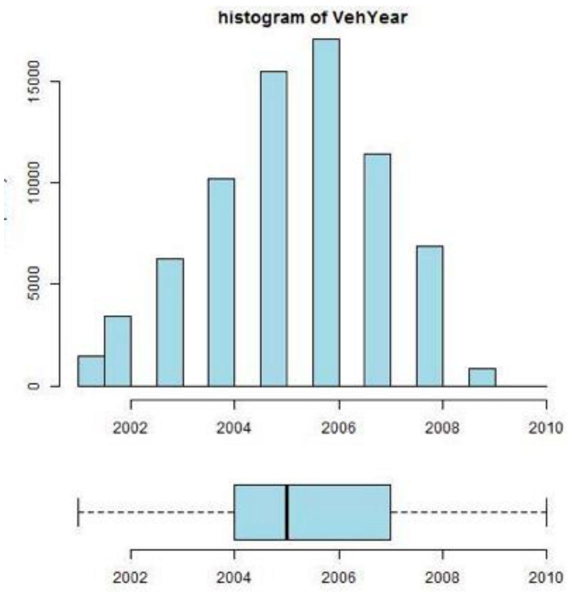
Appendix A: Data Dictionary

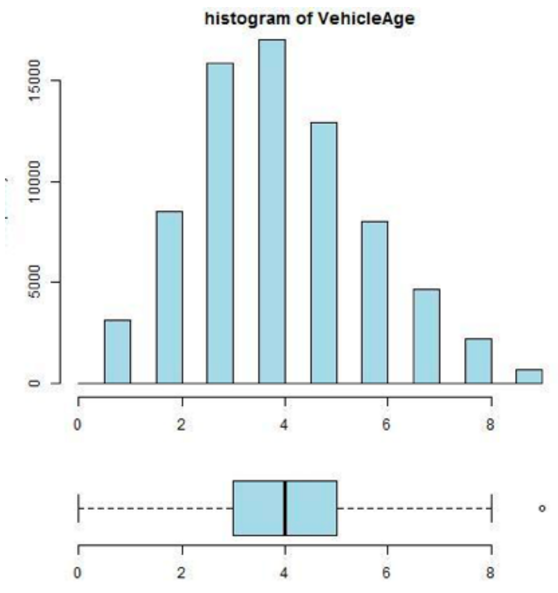
Field Name	Definition
RefID	Unique number assigned to vehicles
IsBadBuy	Identifies if the kicked vehicle was an avoidable purchase
PurchDate	The Date the vehicle was Purchased at Auction
Auction	Auction provider at which the vehicle was purchased
VehYear	The manufacturer's year of the vehicle
VehicleAge	The years since the manufacturer's year
Make	Vehicle Manufacturer
Model	Vehicle Model
Trim	Vehicle Trim Level
SubModel	Vehicle Sub-model
Color	Vehicle Color
Transmission	Vehicles transmission type (Automatic, Manual)
WheelTypeID	The type ID of the vehicle wheel
WheelType	The wheel type description (Alloy, Covers)
VehOdo	The vehicles odometer reading
Nationality	The Manufacturer's country
Size	The size of vehicle (Compact, SUV, etc.)
TopThreeAmericanName	Identifies if manufacturer belongs to top American manufacturers
MMRAcquisitionAuctionAveragePrice	Acquisition price for this vehicle in average condition at time of purchase
MMRAcquisitionAuctionCleanPrice	Acquisition price for this vehicle in the above Average condition at time of purchase
MMRAcquisitionRetailAveragePrice	Acquisition price for this vehicle in the retail market in average condition at time of purchase
MMRAcquisitonRetailCleanPrice	Acquisition price for this vehicle in the retail market in above average condition at time of purchase
MMRCurrentAuctionAveragePrice	Acquisition price for this vehicle in average condition as of current day
MMRCurrentAuctionCleanPrice	Acquisition price for this vehicle in the above condition as of current day
MMRCurrentRetailAveragePrice	Acquisition price for this vehicle in the retail market in average condition as of current day
MMRCurrentRetailCleanPrice	Acquisition price for vehicle in the retail market in above average condition as of current day
PRIMEUNIT	Identifies if vehicle would have a higher demand than a standard purchase
AcquisitionType	Identifies how vehicle was acquired (Auction buy, trade in, etc.)
AUCGUART	The level guarantee provided by auction for the vehicle (Green light - Guaranteed/arbitral, Yellow Light - caution/issue, red light - sold as is)

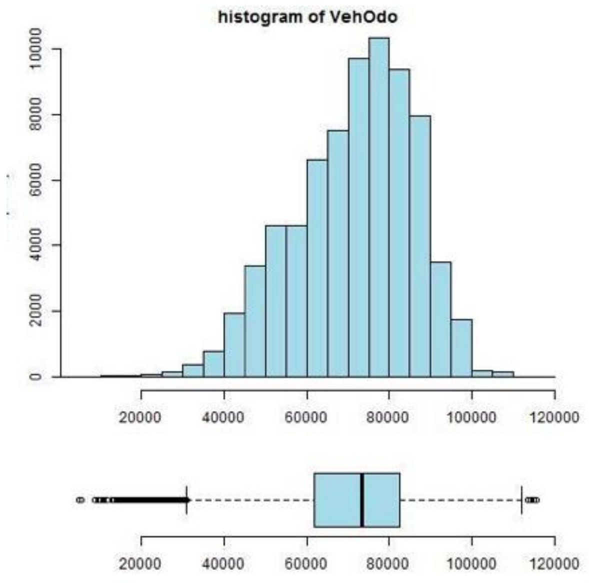
KickDate	Date the vehicle was kicked back to the auction
BYRNO	Unique number of buyer that purchased vehicle
VNZIP	Zip code where the car was purchased
VNST	State where the car was purchased
VehBCost	Acquisition cost paid for vehicle for purchase
IsOnlineSale	Identifies if the vehicle was purchased online
WarrantyCost	Warranty price

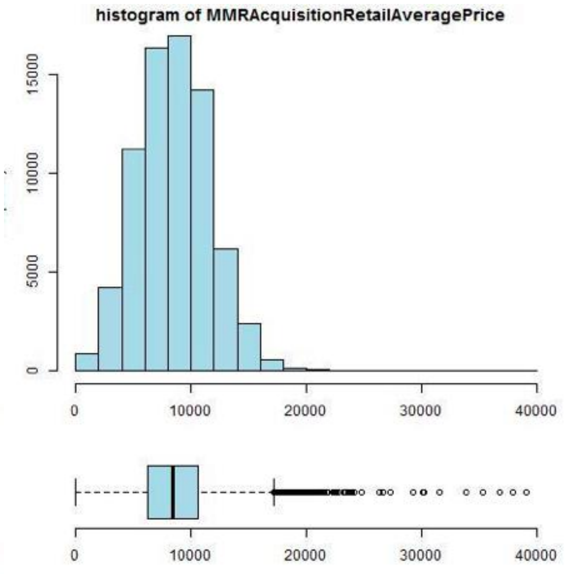
Appendix B: Summary Statistics

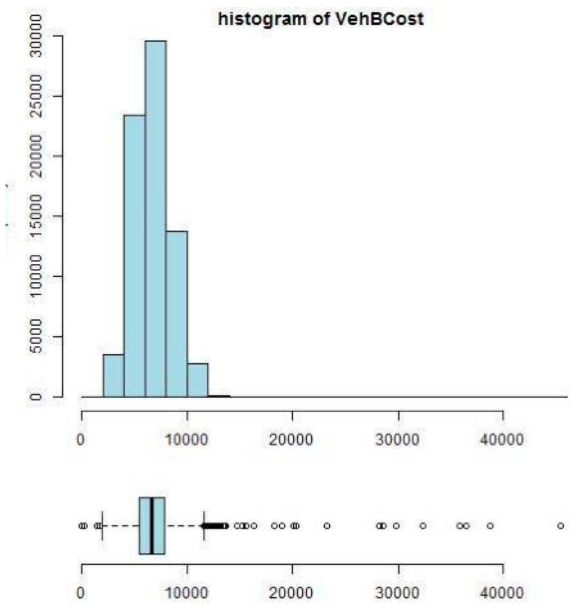
Distributions of Continuous Variables:

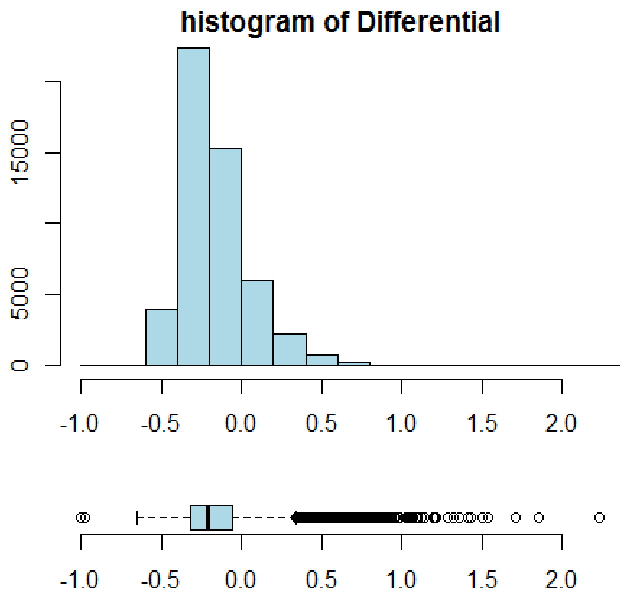




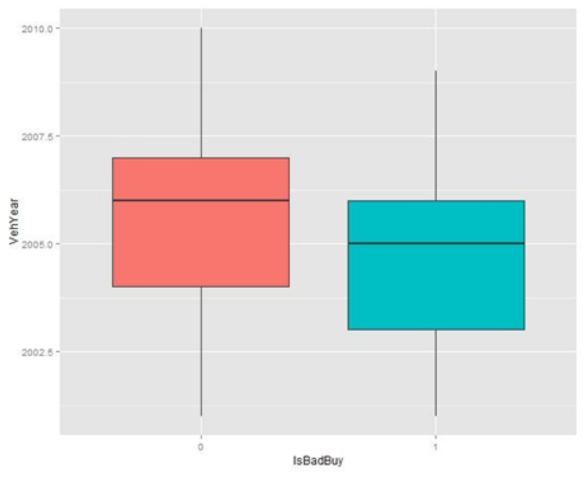


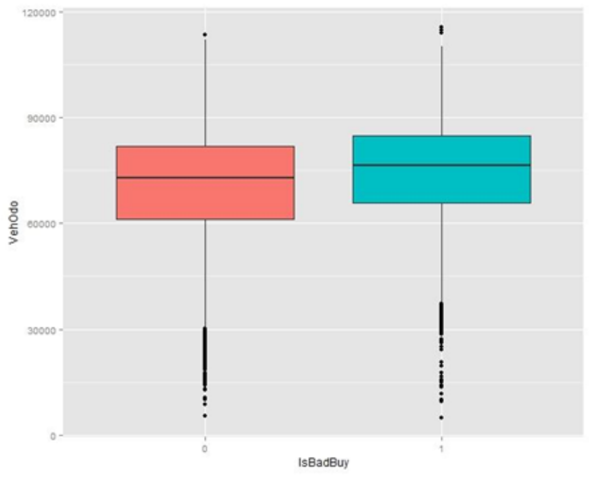


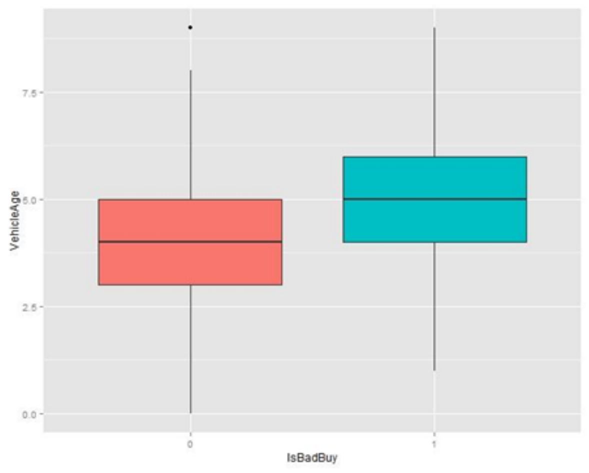


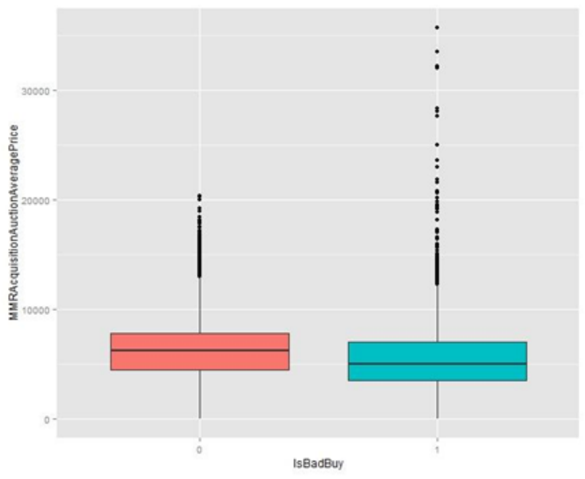


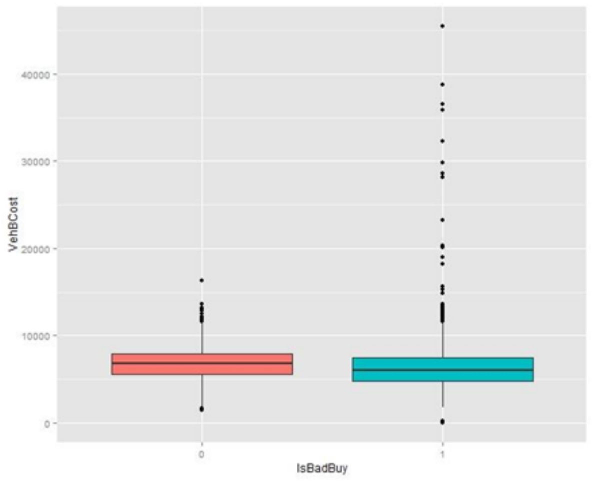
Box Plots of Continuous Predictors vs. Response Variables

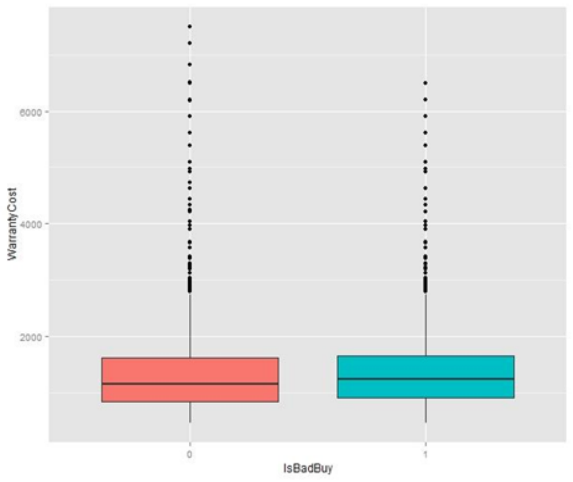












Frequency Tables of Selected Categorical Variables:

IsOnlineSale:

	Frequency	Percent
0	71138	97.47201403
1	1845	2.527985969
Total	72983	100

Nationality:

	Frequency	Percent
AMERICAN	61028	83.61947303
NULL	5	0.00685091
OTHER	195	0.267185509
OTHER ASIAN	8033	11.00667279
TOP LINE ASIAN	3722	5.099817766
Total	72983	100

Auction:

	Frequency	Percent
ADESA	14439	19.7840593
MANHEIM	41043	56.23638382
OTHER	17501	23.97955688
Total	72983	100

WheelType:

	Frequency	Percent
Alloy	36050	49.3950646
Covers	33004	45.22148994
NULL	3174	4.348957977
Special	755	1.034487483
Total	72983	100

AUCGUART:

	Frequency	Percent
GREEN	3340	4.576408205
NULL	69564	95.31534741
RED	79	0.108244386
Total	72983	100

Transmission:

	Frequency	Percent
	1	0.001370182
AUTO	70398	96.45807928
Manual	1	0.001370182
MANUAL	2575	3.5282189
NULL	8	0.010961457
Total	72983	100

TopThreeAmericanName:

	Frequency	Percent
CHRYSLER	23399	32.06089089
FORD	12315	16.87379253
GM	25314	34.68478961
NULL	5	0.00685091
OTHER	11950	16.37367606
Total	72983	100

Appendix C:

Table C1. CP Table for Decision Tree Model

tree	CP	nsplit	rel error	xerror	xstd
1	0.14951177	0	1	1	0.01185
2	0.02977429	1	0.85049	0.85049	0.011042
3	0.01136546	2	0.82071	0.82071	0.010869
4	0.00128061	3	0.80935	0.80967	0.010804
5	0.00112054	5	0.80679	0.81047	0.010808
6	0.00096046	9	0.80215	0.80887	0.010799
7	0.00080038	11	0.80022	0.80727	0.010789
8	0.00064031	16	0.7951	0.80615	0.010783
9	0.00053359	21	0.7919	0.80727	0.010789
10	0.00046022	27	0.7887	0.80791	0.010793
11	0.00034683	35	0.78502	0.81783	0.010852
12	0.00032015	62	0.76981	0.8252	0.010895
13	0.00028013	85	0.76085	0.83256	0.010938
14	0.00025612	90	0.75924	0.84072	0.010986
15	0.00024012	95	0.75796	0.85177	0.011049
16	0.00022868	125	0.74932	0.85737	0.011081
17	0.00021344	135	0.7466	0.86409	0.011119
18	0.0002001	180	0.73347	0.86842	0.011144
19	0.00019209	190	0.73107	0.86986	0.011152
20	0.00016008	195	0.73011	0.89483	0.011291

Table C2: Neural Networks: Misclassification Error and Gini Index on Training Set

node	lambda	misclass	Gini
2	0.01	0.101034	0.225152
2	0.05	0.10011	0.225699
2	0.1	0.100543	0.227492
2	0.5	0.100405	0.226145
3	0.01	0.101526	0.22115
3	0.05	0.100562	0.221979
3	0.1	0.100779	0.222763
3	0.5	0.099992	0.225445
4	0.01	0.10186	0.21674
4	0.05	0.101782	0.215473
4	0.1	0.100877	0.217888
4	0.5	0.100621	0.222493
7	0.01	0.102096	0.210942
7	0.05	0.103335	0.207434
7	0.1	0.102332	0.212984
7	0.5	0.101329	0.218869
8	0.01	0.103827	0.207647
8	0.05	0.104574	0.208685
8	0.1	0.103453	0.208571
8	0.5	0.101447	0.216667

Appendix D: Source Code

D0: Exploratory Analysis

```
# https://www.kaggle.com/c/DontGetKicked/forums/t/965/external-data
# https://www.census.gov/econ/geo-zip.html
# http://proximityone.com/zipequiv.htm
#### Load Data ###########################################

# AGinity
# main = "\\\\nas1/labuser169"
# course = "MSIA401_Project"
# setwd(file.path(main,course))

# My PC
main = "C:/Users/Steven/Documents/Academics/3_Graduate School/2014-2015 ~ NU/"
course = "MSIA_420_Predictive_Analytics"
datafolder = "Project"
setwd(file.path(main,course, datafolder))

#opts_knit$set(root.dir = getwd())

# Import data
filename = "training.csv"
mydata = read.csv(filename,header = T)

#### Process Data ########################################

# Look at data
names(mydata)
head(mydata)
nrow(mydata)
ncol(mydata)
summary(mydata)

str(mydata)

# not many levels of buyers (vs number of rows), so keep this column
nlevels(factor(mydata$BYRNO))
hist(mydata$BYRNO)

# not many levels of zipcodes (vs number of rows), so keep this column
nlevels(factor(mydata$VNZIP1))

# trim levels depends on car model?
length(levels(mydata$Trim))
nlevels(mydata$Trim)
```

```
levels(mydata$Trim)

# can potentially link zipcodes to demographic-economic data

# drop first column REFID
mydata = mydata[,-which(names(mydata)=="RefId")]
ncol(mydata)

str(mydata)

resp = "IsBadBuy"

var_cont = c("VehYear", "VehicleAge","VehOdo",
        "MMRAcquisitionAuctionAveragePrice",
        "MMRAcquisitionAuctionCleanPrice",
        "MMRAcquisitionRetailAveragePrice",
        "MMRAcquisitonRetailCleanPrice",
        "MMRCurrentAuctionAveragePrice",
        "MMRCurrentAuctionCleanPrice",
        "MMRCurrentRetailAveragePrice",
        "MMRCurrentRetailCleanPrice","VehBCost",
        "WarrantyCost")

var_cat = c("IsBadBuy" , "Auction","Make", "Model",
        "Trim","SubModel","Color","Transmission",
        "WheelTypeID","WheelType","Nationality", "Size",
        "TopThreeAmericanName","PRIMEUNIT",
        "AUCGUART", "BYRNO", "VNZIP1","VNST","IsOnlineSale")

# no Kickdate?
var_date = c("PurchDate")

# convert date to numeric?

# factors and levels in R
# http://www.stat.berkeley.edu/~s133/factors.html

# convert factor to character
# http://stackoverflow.com/questions/2851015/convert-data-frame-columns-from-factors-to-characters

# find every factor variable and convert to character (doesn't apply here)
# i = sapply(mydata, is.factor)
# mydata[i] = lapply(mydata[i], as.character)

# convert character to numeric
# http://www.statmethods.net/management/typeconversion.html
# convert levels variable to numeric (prices)
# NULL values present, when converted to numeric they will become NA
sum(as.character(mydata[,var_cont[4]])=="NULL")

# convert first to character then to numeric for continuous variables
i = sapply(mydata[,var_cont], is.factor)
mydata[,var_cont][i] = lapply(mydata[,var_cont][i], function(x) as.numeric(as.character(x)))
sum(is.na(mydata[,var_cont[4]]))

# convert numeric to factor for categorical variabes
i = sapply(mydata[,var_cat], is.numeric)
mydata[,var_cat][i] = lapply(mydata[,var_cat][i], as.factor)
# drop response variable from categorical variable list
var_cat = var_cat[-which(var_cat=="IsBadBuy")]

str(mydata)

# date format
# http://www.statmethods.net/input/dates.html
# http://www.ats.ucla.edu/stat/r/faq/string_dates.htm
# http://stackoverflow.com/questions/9216138/find-the-day-of-a-week-in-r
# http://stackoverflow.com/questions/9749598/r-obtaining-month-and-year-from-a-date
# http://www.quantlego.com/howto/introduction-dates-and-times-in-r/

# convert to dates
i = sapply(mydata[var_date], is.factor)
mydata[,var_date] = as.character(mydata[,var_date])
mydata[,var_date] = as.Date(mydata[,var_date],"%m/%d/%Y")


# create additional variables based on dates
mydata$PurchWeekDay = factor(weekdays(mydata[,var_date]),
                    levels = c("Monday", "Tuesday",
                            "Wednesday","Thursday",
                            "Friday", "Saturday",
                            "Sunday"),ordered=T)

mydata$PurchMonth = factor(months(mydata[,var_date]),
                            levels=c("January","February","March",
                            "April","May","June","July","August","September",
                            "October","November","December"),ordered=TRUE)

mydata$PurchQuarter = factor(quarters(mydata[,var_date]),
                    levels = c("Q1","Q2","Q3","Q4"),
                    ordered=TRUE)

mydata$PurchYear= as.numeric(format(mydata[,var_date], "%Y"))
```

```
#### Exploratory: categorical ###############################
# Add names to table in R
# http://stackoverflow.com/questions/19963960/adding-extra-column-name-and-row-name-to-a-table-in-r

# Visualization contingency tables
# http://statmath.wu.ac.at/projects/vcd/
# http://www.statmethods.net/advgraphs/mosaic.html
# http://www.datavis.ca/online/mosaics/about.html

library(vcd)

# too many levels to print
noprint = c("Model","Trim","SubModel")

crosstabs_list =list()

for (var in var_cat){
  mytable = table(mydata[,var],mydata[,resp])
  names(dimnames(mytable))=c(var,resp)

  crosstabs_list[[var]]= mytable

  mytable2 = cbind(prop.table(mytable, 1),mytable)
  names(dimnames(mytable2))=c(var,resp)

  if(!(var %in% noprint)){
        print(mytable2)

        save_name = paste(c("mosaic_",var,".jpg"),sep="",collapse="")
        jpeg(save_name)
        mosaic(mytable,shade=TRUE,legend=TRUE)
        dev.off()
  }
  save_name = paste(c("crosstab_",var,".csv"),sep="",collapse="")
  write.csv(mytable2, save_name)

}

fit = glm( IsBadBuy ~ . - Model - Trim - SubModel , family=binomial(link="logit"), data=mydata)

####### Univariate stats (e.g. mean, skewness, frequency)

# http://en.wikibooks.org/wiki/R_Programming/Descriptive_Statistics
# http://www.statmethods.net/stats/descriptives.html

# install.packages("psych")
library(psych)
```

```
### Continuous Variables
ntotal = dim(mydata)[1]

stats_cont = describe(mydata[,var_cont])
n_na = ntotal-stats_cont[,"n"]
stats_cont = cbind(n=stats_cont[,"n"],n_na,stats_cont[,c("mean","sd","median","min","max",
                                                          "range","skew","kurtosis","se")])
stats_cont = round(stats_cont,2)
stats_cont

write.csv(stats_cont, "stats_cont.csv")

stats_cont_byIsBadBuy = describeBy(mydata[,var_cont], group= mydata[,"IsBadBuy"])
stats_cont_byIsBadBuy

### Dates
summary(mydata[,var_date])
write.csv(summary(mydata[,var_date]), "stats_cat.csv")


### Categorical

#install.packages("descr")
library("descr")

freq(mydata[,resp])

for (var in var_cat){
  print(var)
  save_name = paste(c("freq_",var,".csv"),sep="",collapse="")
  write.csv(freq(mydata[,var],main=var), save_name)

}


########

# look at dates
plot(table(mydata$datead6),ylab="freq",xlab="datead6")
plot(table(mydata$datelp6),ylab="freq",xlab="datelp6")

par(mfrow=c(3,3))
par(mfrow=c(1,1))

x=lapply(1:dim(mydata)[2],function(x) hist(mydata[,x],breaks=20))
print(x[[2]])
```

```
y=hist(mydata$datead6,breaks=20)
print(y)


####### Distribution histogram with boxplot

# distributions
############# plots
# http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/
# http://www.cookbook-r.com/Manipulating_data/Summarizing_data/
# http://seananderson.ca/2013/12/01/plyr.html
# http://stackoverflow.com/questions/16083275/histogram-with-marginal-boxplot-in-r
# http://rgraphgallery.blogspot.com/2013/04/rg-plotting-boxplot-and-histogram.html

hist()
mydata2 = subset(mydata,mydata$IsBadBuy!=0) # remove zeros
dim(mydata2)

plotHistBox = function(variable,xl,xu,title){
  nf <- layout(mat = matrix(c(1,2),2,1, byrow=TRUE),  height = c(3,1))
  par(mar=c(3.1, 3.1, 1.1, 2.1))
  hist(variable,xlim=c(xl,xu), col = "lightblue", main=title)
  boxplot(variable, horizontal=TRUE,  outline=TRUE,ylim=c(xl,xu), frame=F, col = "lightblue", width = 10)


}


for (i in 1:length(var_cont)){
variable = mydata[,var_cont[i]]
xl = min(variable,na.rm = TRUE)
xu = max(variable,na.rm = TRUE)
title = paste(c("histogram of ",var_cont[i]),sep="",collapse="")
save_name = paste(c(var_cont[i],".jpg"),sep="",collapse="")
jpeg(save_name)
plotHistBox(variable,xl,xu,title)
dev.off()}



######

# Now used the clean data with new variable for the following analysis


############## Correlation plot + matrix
```

```
numeric_var = var_cont
corr = round(cor(mydata[,numeric_var ]),2)
corr

library(corrplot)
corrplot.mixed(corr, upper = "ellipse", lower = "number")
# corrplot function changes margin, so next graph looks messed up. change to default
par(mar=c(5,4,4,2)+0.1)


pairs(corr)

############## boxplot predictors by targdol_bin
library(ggplot2)
mydata2=mydata
mydata2$IsBadBuy =as.factor(mydata$IsBadBuy)
mydata2$color[mydata$IsBadBuy==0]="red"
mydata2$color[mydata$IsBadBuy==1]="blue"

ggplot(mydata2, aes(x=IsBadBuy, y=VehYear, fill=IsBadBuy)) + geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=VehicleAge, fill=IsBadBuy)) + geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=VehOdo, fill=IsBadBuy)) + geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRAcquisitionAuctionAveragePrice, fill=IsBadBuy)) +
geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRAcquisitionAuctionCleanPrice, fill=IsBadBuy)) +
geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRAcquisitionRetailAveragePrice, fill=IsBadBuy)) +
geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRAcquisitonRetailCleanPrice, fill=IsBadBuy)) + geom_boxplot()
+ guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRCurrentRetailAveragePrice, fill=IsBadBuy)) + geom_boxplot() +
guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=MMRCurrentRetailCleanPrice, fill=IsBadBuy)) + geom_boxplot() +
guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=VehBCost, fill=IsBadBuy)) + geom_boxplot() + guides(fill=FALSE)
ggplot(mydata2, aes(x=IsBadBuy, y=WarrantyCost, fill=IsBadBuy)) + geom_boxplot() +
guides(fill=FALSE)
```

D1: Logistic Regression

```
car_test <- read.csv("/sscc/home/a/amk202/Predictive/car_test.csv")
car_train <- read.csv("/sscc/home/a/amk202/Predictive/car_train.csv")
mydata = car_train

mydata=car_test <- read.csv("/sscc/home/a/amk202/Predictive/car_test.csv")
drop = c("PurchDate","VehYear","Model","Trim","SubModel","PRIMEUNIT","AUCGUART","VNST")
mydata = mydata[!(names(mydata) %in% drop)]
car_test = car_test[!(names(car_test) %in% drop)]

fit<-glm(IsBadBuy~., data = mydata)
summary(fit)

library(MASS)
#use stepwise to select predictors
fit2<-stepAIC(fit)
summary(fit2)
formula(fit2)
mydata$WheelType = relevel(mydata$WheelType,"NULL") #set "NULL" group to be base category
#drop insignificant variables from the model
f = update(formula(fit2), .~. - bachelor-Auction-Make-region- WarrantyCost -mean_income-Transmission-
Size)
fit3 = glm(f,data=mydata)
summary(fit3)

#check multicollinearity among predictors
vif(fit3)

#drop predictors has vif larger than 10, which mean there are multicollinearity within them
f = update(formula(fit3), .~. -median_income-MMRCurrentRetailCleanPrice- total_population-
MMRAcquisitionRetailAveragePrice-moved_total-MMRAcquisitionAuctionAveragePrice-
MMRAcquisitonRetailCleanPrice-MMRCurrentAuctionAveragePrice)
fit4 = glm(f,data=mydata)
summary(fit4)

#exponential coefficients to get the coefficients of predictor on odds of bad-buy vs. good-buy
exp_coef = exp(cbind(OR = coef(fit4),confint(fit4)))

#predict outcome based on training set
yhat = predict(fit4,type="response")
yp = yhat
#set cut off to be 0.5
yp[yhat>0.5] = 1
yp[yhat<=0.5] = 0
y = mydata$IsBadBuy
#get the classification table
```

```
table = table(y,yp)
table
#compute the misclassification rate
sum(yhat != y)/length(y)

#set up Gini function
Gini = function ( actual, predicted ) {
  n = length( actual );
  Gini_data = as.data.frame( cbind( actual, predicted, row = 1:n ) );
  Gini_data = Gini_data[ with( Gini_data, order( -predicted, row ) ), ];
  sum( cumsum( Gini_data[ , 1 ] ) / sum( Gini_data[ , 1 ] ) - ( 1:n ) / n ) / n;
}

#predict outcome based on test set
phat = predict(fit4, car_test, type ="resp")
yhat = phat
#set cut off to be 0.5
yhat[phat>0.5] = 1
yhat[phat<0.5] = 0
y= car_test$IsBadBuy
#test set classification table
tab2 = table(y, yhat)
tab2
#misclassification rate on test set
sum(yhat != y)/length(y)

y_test = car_test$IsBadBuy
y_test_hat = phat
Gini(y_test,y_test_hat) # get Gini on test set

library(pROC)
#plot ROC curve
roc = plot.roc(car_test$IsBadBuy,phat)
#get area under ROC curve
auc(roc)
```

D2: Decision Trees

```
setwd("/sscc/home/s/scg324/")
car_train <- read.csv("/sscc/home/a/amk202/Predictive/car_train.csv")
car_test <- read.csv("/sscc/home/a/amk202/Predictive/car_test.csv")

# Variable Preselection
varUsed <- c("IsBadBuy", "Auction", "Make", "Transmission", "WheelType","Nationality",
        "Size", "TopThreeAmericanName","AUCGUART", "IsOnlineSale","VehicleAge",
        "VehBCost", "WarrantyCost", "MMRAcquisitionAuctionAveragePrice",
        "region", "Differential","VehOdo"
        #"MMRAcquisitionAuctionCleanPrice","MMRAcquisitionRetailAveragePrice",
        #"MMRAcquisitonRetailCleanPrice", "MMRCurrentAuctionAveragePrice",
            #"MMRCurrentAuctionCleanPrice","MMRCurrentRetailAveragePrice",
            #"MMRCurrentRetailCleanPrice"
)
car_train1<- car_train[varUsed]
car_test1 <- car_test[varUsed]

# Tree
library(rpart)
fit.tree <- rpart(IsBadBuy~.,method = "class", car_train1, cp = 0.0001)
printcp(fit.tree)
plotcp(fit.tree)
summary(fit.tree)

# Prune the Tree
pfit<- prune(fit.tree, cp=fit.tree$cptable[which.min(fit$cptable[,"xerror"]),"CP"])
# Misclassification Rate
class.pred <- table(predict(pfit, type="class"), car_train1$IsBadBuy)
1-sum(diag(class.pred))/sum(class.pred)

# CV with Test set
# Gini
preds <- predict(pfit, newdata = car_test1, type = "prob")
Gini(car_test1$IsBadBuy,preds[,2])
# Misclassification Rate
preds.class <- predict(pfit, newdata = car_test1, type = "class")
preds.class <-as.numeric(as.character(preds.class))
predTable <- table(preds.class, car_test1$IsBadBuy)
1-sum(diag(predTable))/sum(predTable)

# ROC
library(pROC)
plot.roc(car_test1$IsBadBuy, preds[,2])
```

D3: Neural Network

```
###### Load Data #######
setwd("~/Predictive")
train = read.csv("car_train.csv", header=T)
test = read.csv("car_test.csv", header=T)
all = read.csv("car_data_clean.csv", header=T)

var_cont = c("VehYear", "VehicleAge","VehOdo",
            "MMRAcquisitionAuctionAveragePrice",
            "MMRAcquisitionAuctionCleanPrice",
            "MMRAcquisitionRetailAveragePrice",
            "MMRAcquisitonRetailCleanPrice",
            "MMRCurrentAuctionAveragePrice",
            "MMRCurrentAuctionCleanPrice",
            "MMRCurrentRetailAveragePrice",
            "MMRCurrentRetailCleanPrice","VehBCost",
            "WarrantyCost","Differential")

var_cat = c("IsBadBuy" , "Auction","Make", "Model",
          "Trim","SubModel","Color","Transmission",
          "WheelTypeID","WheelType","Nationality", "Size",
          "TopThreeAmericanName","PRIMEUNIT",
          "AUCGUART", "BYRNO", "VNZIP1","VNST","IsOnlineSale")

var_census = c("moved_total","moved_diff_county","moved_diff_state",
             "median_income", "mean_income", "total_population",
             "high_school", "bachelor", "urban" , "rural" , "region")

var_cont_include = var_cont[c(2,3,12,13,14)]
var_cat_include = var_cat[c(1,2,3,7,8,10,11,12,13,14,15,19)]
var_census_include = var_census[c(11)]
var_inc = c(var_cont_include, var_cat_include,var_census_include)

# Take log of WarrantyCost
train$WarrantyCost = log(train$WarrantyCost)
test$WarrantyCost = log(test$WarrantyCost)

# Factor response
train$IsBadBuy <- as.factor(train$IsBadBuy)

# Standardize dataset
train[var_cont_include] <- sapply(train[var_cont_include], function(x) (x-mean(x))/sd(x))
test[var_cont_include] <- sapply(test[var_cont_include], function(x) (x-mean(x))/sd(x))



#######A function to determine the indices in a CV partition##################
CVInd <- function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices for each
part
  m<-floor(n/K)  #approximate size of each part
  r<-n-m*K
  I<-sample(n,n)  #random reordering of the indices
  Ind<-list()  #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart]  #indices for kth part of data
  }
  Ind
}
##################################################################################
Gini = function ( actual, predicted ) {
  n = length( actual );
  Gini_data = as.data.frame( cbind( actual, predicted, row = 1:n ) );
  Gini_data = Gini_data[ with( Gini_data, order( -predicted, row ) ), ];
  sum( cumsum( Gini_data[ , 1 ] ) / sum( Gini_data[ , 1 ] ) - ( 1:n ) / n ) / n;
}
#################################################################################

#####################
## Param Selection ##
#####################
# Continuous and some categorical
train.sub = train[var_inc]
folds <- 5
Ind<-CVInd(n=nrow(train),folds)
K <- length(Ind)
y <- train.sub$IsBadBuy
yhat<-y
phat <- rep(0,n)
n <- length(y)

misclass = data.frame(j=numeric(0),
hnode=numeric(0),lambda=numeric(0),misclass=numeric(0),Gini=numeric(0))
h = c(2:9)
lambdas = c(.01,.03,.05,.07,.1,.3,.5,.7)

for(j in 1:5){
  for(hnodes in h){
    for(lambda in lambdas){
      for (k in 1:K) {
        print(c(hnodes, lambda, k))
        out<-nnet(IsBadBuy~.,train.sub[-Ind[[k]],], linout=F, skip=F, size=hnodes, decay=lambda,
maxit=1000, trace=F)
        yhat[Ind[[k]]] <- predict(out, train.sub[Ind[[k]],], type = "class")
        phat[Ind[[k]]] <- predict(out, train.sub[Ind[[k]],], type = "raw")
      }
      mis.rate = sum(y!=yhat)/n;
      Gini.train = Gini(as.numeric(y==1), phat)
      misclass<-rbind(misclass,c(j,hnodes,lambda,mis.rate,Gini.train))
      names(misclass) <- c("run", "hnode", "lambda", "misclass", "Gini")
      print(misclass)
      #      write.table(misclass,"nn_cv.csv", append=T, col.names=F, row.names=F, sep=',')
    }
  }
}
misclass[with(misclass,order(misclass)),]
misclass[which.min(misclass$misclass),]

######  Test  ######
```

```
y = train$IsBadBuy
n = nrow(train)
start = proc.time()
out<-nnet(IsBadBuy~.,train[var_inc], linout=F, skip=F, size=3, decay=.5, maxit=1000, trace=F)
runtime = proc.time()-start

### Test  ###
y = train$IsBadBuy
n = nrow(train)
start = proc.time()
out<-nnet(IsBadBuy~.,train[var_inc], linout=F, skip=F, size=3, decay=.5, maxit=1000, trace=F)
runtime = proc.time()-start

# Test Gini
test.y = test$IsBadBuy
phat <- predict(out, newdata=test, type="raw")
Gini(as.numeric(test.y==1), phat)

# Test Misclassification
test.n = length(test$IsBadBuy)
ynewhat = as.numeric(phat2>0.5)
e = sum(test.y!=ynewhat)
misclass_rate = e/test.n; misclass_rate
table(test.y, ynewhat)
prop.table(table(test.y, ynewhat),margin=1)

# Area under curve
library(pROC)
plot.roc(test.y,phat, main="Area Under Curve (Neural Network)")
roc=roc(test.y,phat2,percent=TRUE)
```

D4: Random Forest

```
setwd("/sscc/home/a/amk202/SuddenLink/predictive")
# Import data
filename = "/sscc/home/a/amk202/Predictive/car_train.csv"
mydata = read.csv(filename,header = T)

filename = "/sscc/home/a/amk202/Predictive/car_test.csv"
test = read.csv(filename,header = T)

str(mydata)

var_drop = c("PurchDate","VehYear","Model", "Trim","SubModel","PRIMEUNIT","VNST")

mydata = mydata[!(names(mydata) %in% var_drop)]
```

```
var_cont = c( "VehicleAge","VehOdo",
          "MMRAcquisitionRetailAveragePrice","VehBCost",
          "WarrantyCost", "Differential" )

var_drop = c("MMRAcquisitionAuctionAveragePrice",
          "MMRAcquisitionAuctionCleanPrice",
          "MMRAcquisitonRetailCleanPrice",
          "MMRCurrentAuctionAveragePrice",
          "MMRCurrentAuctionCleanPrice",
          "MMRCurrentRetailAveragePrice",
          "MMRCurrentRetailCleanPrice",
          "moved_total","moved_diff_county",
          "moved_diff_state","median_income","mean_income",
          "total_population", "high_school", "bachelor", "urban",
          "rural")

mydata = mydata[!(names(mydata) %in% var_drop)]

names(mydata)

resp = "IsBadBuy"
pred = names(mydata)[names(mydata)!=resp]
mydata[,resp] = as.factor(mydata[,resp])

# sample to test code
#mydata = mydata[sample(rownames(mydata),5000,replace=FALSE),]

#### Scale: Standardize Predictors ####

#standardize = function( x, newdata, data ) {
  #( newdata[ , x ] - mean( data[ , x ] ) ) / sd( data[ , x ] );
#}

#standardize("bachelor",test,mydata)

#mydata_pre_scale = mydata
#mydata[,var_cont]= sapply(mydata[,var_cont], function(x) (x-mean(x))/sd(x)) #standardize predictors
#test[,var_cont]= sapply(test[,var_cont], function(x) (x-mean(x))/sd(x)) #standardize predictors
# or use:
# scale(mydata[,var_cont])


#### GINI ####

Gini = function ( actual, predicted ) {
  n = length( actual );
  Gini_data = as.data.frame( cbind( actual, predicted, row = 1:n ) );
  Gini_data = Gini_data[ with( Gini_data, order( -predicted, row ) ), ];
  sum( cumsum( Gini_data[ , 1 ] ) / sum( Gini_data[ , 1 ] ) - ( 1:n ) / n ) / n;
}

#### Random Forest ####

library (randomForest)

# Tune parameters
ptm <- proc.time()
tuneRandomForest = tuneRF(mydata[,pred], mydata[,resp], ntreeTry = 300, stepFactor=1.5)
proc.time() - ptm

# Start the clock!
ptm <- proc.time()

out =randomForest(IsBadBuy~.,data=mydata,ntree=300,mtry=3,nodesize=5 ,importance =TRUE)

# Stop the clock
proc.time() - ptm

# train
y = mydata[,resp]
yhat = predict(out,type="class")
phat = predict(out,type="prob")[,2]
sum(y != yhat)/length(y)

# test
y = test[,resp]
y = as.numeric(y==1)
yhat = predict(out,newdata= test, type="class")
phat = predict(out,newdata= test, type="prob")[,2]
sum(y != yhat)/length(y)
Gini(y,phat)

varImpPlot (out)

# install.packages("pROC")
library(pROC)
plot.roc(y,phat)

roc=roc(y,phat,percent=TRUE)
auc(roc)


# Start the clock!
ptm <- proc.time()
```

```
Nrep = 10
n=nrow(mydata)
y = mydata[,resp]

Ntrees = seq(100,500,100)
Npred= seq(1,10,1)

#n iterations , 30 sper iteraion
Nrep*length(Ntrees)*length(Npred)
loops = Nrep*length(Ntrees)*length(Npred) # without folds

#misclass= matrix(0,Nrep*length(lambdas)*length(models),4) # misclass for each model and each
replicate
misclass = c()

for (j in 1:Nrep) {

  for (m in Ntrees){

        yhat = y; # reset yhat (actually don't need this)

        for(l in Npred){
        out =randomForest(IsBadBuy~.,data=mydata,ntree=m,nodesize=l, importance =TRUE)

        yhat = predict(out,type="class")
        phat = predict(out,type="prob")[,2]

        misclass = rbind(misclass,c(j,m,l,sum(y!=yhat)/length(y)))

        loops = loops - 1
        print(paste("Remaining Iterations: ",loops))
        print(misclass[nrow(misclass),])
        write(misclass[nrow(misclass),],file="myfile.csv",append=TRUE)

        }# end of l loop

  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm

colnames(misclass) = c("rep","Ntree","Npred","misclass")
misclass
misclassAVE = aggregate(misclass~ Ntree + Npred, misclass, mean ) # aggregate reps
misclassAVE = misclassAVE[order(misclassAVE[,"Ntree"]),] # sort
misclassAVE

param_opt = misclassAVE[which.min(misclassAVE[,"misclass"]),]
param_opt

write.csv(param_opt, file = "best_RandomForests.csv",row.names=FALSE)

write.csv(misclassAVE, file = "misclassAVE__RandomForests.csv",row.names=FALSE)

#### Use best ####

test = read.csv(filename,header = T)
#mydata[,resp] = as.factor(mydata[,resp])
#test[,var_cont]= sapply(test[,var_cont], function(x) (x-mean(x))/sd(x)) #standardize predictors

# Start the clock!
ptm <- proc.time()

out =randomForest(IsBadBuy~.,data=mydata,
                  ntree=as.numeric(param_opt["Ntree"]),
                  nodesize=as.numeric(param_opt["Npred"]), importance =TRUE)

# Stop the clock
proc.time() - ptm

# train
y = mydata[,resp]
yhat = predict(out,type="class")
phat = predict(out,type="prob")[,2]
sum(y != yhat)/length(y)

t = table(y,yhat)
prop.table(t,1)

# test
y = test[,resp]
y = as.numeric(y==1)
yhat = predict(out,newdata= test, type="class")
phat = predict(out,newdata= test, type="prob")[,2]
sum(y != yhat)/length(y)
Gini(y,phat)

# install.packages("pROC")
library(pROC)
plot.roc(y,phat)
```

```
roc=roc(y,phat,percent=TRUE)
auc(roc)

out$importance
importance(out)
varImpPlot(out)

title = paste(c("Variable Importance Plot using Random Forests"),sep="",collapse="")
save_name = paste(c("RF_Plot",".jpg"),sep="",collapse="")
jpeg(save_name)
varImpPlot (out)
dev.off()
```

D5: Gradient Tree Boosting

```
library( corrplot );
library( caret );
library( gbm );

setwd( "/sscc/home/a/amk202/Predictive" );

Gini = function ( actual, predicted ) {
  n = length( actual );
  Gini_data = as.data.frame( cbind( actual, predicted, row = 1:n ) );
  Gini_data = Gini_data[ with( Gini_data, order( -predicted, row ) ), ];
  sum( cumsum( Gini_data[ , 1 ] ) / sum( Gini_data[ , 1 ] ) - ( 1:n ) / n ) / n;
}

carTrain = read.csv( "car_train.csv", header = T );
carTest = read.csv( "car_test.csv", header = T );

n = nrow( carTrain );

N = 10;

interactDepth = 2:7;
lambda = seq( 0.01, 0.1, 0.02 );

CVerror = matrix( NA, length( interactDepth ), length( lambda ) );

for ( i in 1:length( interactDepth ) ) {
  for ( l in 1:length( lambda ) ) {
    CVerr = c( );

    for ( j in 1:N ) {
      fit_gbm = gbm( IsBadBuy ~ ., data = carTrain[ , c( varResp, varPred ) ], distribution = "bernoulli",
                     n.trees = 3000, shrinkage = lambda[ l ], interaction.depth = interactDepth[ i ],
                bag.fraction = 0.5, train.fraction = 1, n.minobsinnode = 20, cv.folds = 10, keep.data = T );

      bestIter = gbm.perf( fit_gbm, method = "cv", plot.it = F );
      CVerr = c( CVerr, fit_gbm$cv.error[ bestIter ] );
    }

    CVerror[ i, l ] = mean( CVerr );
  }
}

indexMin = which.min( CVerror );
IDMin = interactDepth[ 1 + ( indexMin - 1 ) %% ( length( interactDepth ) ) ];
lambdaMin = lambda[ 1 + ( indexMin - 1 ) %/% ( length( interactDepth ) ) ];

# IDMin = 4, lambdaMin = 0.01

fit_gbm = gbm( IsBadBuy ~ ., data = carTrain[ , c( varResp, varPred ) ], distribution = "bernoulli",
               n.trees = 3000, shrinkage = lambdaMin, interaction.depth = IDMin, bag.fraction = 0.5,
               train.fraction = 1, n.minobsinnode = 20, cv.folds = 10, keep.data = T );

bestIter = gbm.perf( fit_gbm, method = "cv" );
summary( fit_gbm, n.trees = bestIter );

print( pretty.gbm.tree( fit_gbm, bestIter ) );

plot( fit_gbm, i.var = c( 7, 13 ), n.trees = bestIter );
plot( fit_gbm, i.var = c( 13, 16 ), n.trees = bestIter );

phat = predict( fit_gbm, newdata = carTest, n.trees = bestIter, type = "response" );
yhat = as.numeric( phat > 0.5 );

sum( yhat != carTest[ , "IsBadBuy" ] ) / nrow( carTest );
table( carTest[ , "IsBadBuy" ], yhat );

Gini( carTest[ , "IsBadBuy" ], phat );

plot.roc( carTest[ , "IsBadBuy" ], phat )
```

D6: Ensemble Model

```
library( foreach );
library( nnet );
library( gbm );

standardize = function( x, newdata, data ) {
  ( newdata[ , x ] - mean( data[ , x ] ) ) / sd( data[ , x ] );
}
```

```
carTrain = read.csv( "car_train.csv", header = T );
carTest = read.csv( "car_test.csv", header = T );

n = nrow( carTrain );

varResp = 1;
varPred = c( 3, 5:6, 10:16, 25:26, 28:30, 41:42 );
varScale = c( "VehicleAge", "VehOdo", "VehBCost", "WarrantyCost", "Differential" );

train = carTrain;
test = carTest;

train_scale = carTrain;
train_scale[ , varScale ] = scale( train_scale[ , varScale ] );
test_scale = carTest;
test_scale[ , varScale ] = as.data.frame( lapply( colnames( test_scale[ , varScale ] ), standardize,
                                  newdata =  test_scale, data = carTrain ) );

N = 200;

predictions_gbm = foreach( run = 1:N, .combine = cbind ) %do% {
  n = nrow( train );
  sampleboot = sample( n, n );

  fitboot_gbm = gbm( IsBadBuy ~ ., data = train[ sampleboot, c( varResp, varPred ) ], distribution =
"bernoulli",
                  n.trees = 600, shrinkage = 0.01, interaction.depth = 6, bag.fraction = 0.5,
                  train.fraction = 0.5, n.minobsinnode = 10, cv.folds = 0, keep.data = T );
  bestIter = gbm.perf( fitboot_gbm, method = "OOB" );

  print( bestIter );

  phat_gbm = predict( fitboot_gbm, test, n.trees = bestIter, type = "response" );
}

predictions_nnet = foreach( run = 1:N, .combine = cbind ) %do% {
  n = nrow( train_scale );
  sampleboot = sample( n, 0.5 * n );

  fitboot_nn = nnet( IsBadBuy ~ ., train_scale[ sampleboot, c( varResp, varPred ) ], linout = F, skip = F,
                maxit = 1000, trace = F, size = 2, decay = 0.1 );

  print( run );

  phat_nn = predict( fitboot_nn, test_scale, type = "raw" );
}
```

```
phat_gbm = rowMeans( predictions_gbm );
phat_nnet = rowMeans( predictions_nnet );

phat1 = apply( cbind( phat_gbm, phat_nn ), MARGIN = 1, FUN = max );
phat2 = rowMeans( cbind( phat_gbm, phat_nn ) );

yhat1 = as.numeric( phat1 > 0.5 );

sum( yhat1 != carTest[ , "IsBadBuy" ] ) / nrow( carTest );
table( carTest[ , "IsBadBuy" ], yhat1 );

Gini( carTest[ , "IsBadBuy" ], phat1 );

yhat2 = as.numeric( phat2 > 0.5 );

sum( yhat2 != carTest[ , "IsBadBuy" ] ) / nrow( carTest );
table( carTest[ , "IsBadBuy" ], yhat2 );

Gini( carTest[ , "IsBadBuy" ], phat2 );
```