Allstate Purchase Prediction Challenge

Predict 454 DL SEC-55

Final Report

Dan Tattersall

Annie Bruckner

Sherman Chan

Ahmar Mirza

Fabian Peri

Paul Bertucci

NORTHWESTERN
UNIVERSITY

Introduction

In this report, we developed multiple machine-learning models to predict which Allstate insurance policy options a customer purchases. Customers had seven insurance coverage options to choose from. We created several classification models, including Linear Discriminant Analysis (LDA), k-Nearest Neighbors (KNN), Boosted Trees, Random Forests, and Support Vector Machines (SVM), in order to determine which model best predicts the type of insurance policy a customer selects. Predicting the correct insurance policy a customer purchases was our primary goal, with predicting the coverage option in a timely manner as our secondary goal. The sooner we were able to predict the insurance plan the customer would purchase, the more likely we were to retain the customer's business.

This report is divided into seven main sections: (1) The Modeling Problem; (2) Literature Review; (3) The Data; (4) Exploratory Data Analysis; (5) Predictive Modeling: Methods and Results; (6) Comparison of Results; and (7) Conclusions. We conducted all calculations, analyses, and modeling efforts in the R software package using various packages (see R code in Appendix A).

The Modeling Problem

The Allstate Purchase Prediction Challenge is a business problem presented through Kaggle by the Allstate Insurance Company. The objective of the challenge is to predict a customer's policy purchase based on their shopping and quote history. If the policy purchase can be predicted sooner within the shopping window, the quoting process will be shortened and the issuer is less likely to lose the customer's business. Can we predict the policy purchased using the customer's shopping history?

Each policy option is a combination of seven different coverage options, represented by alpha characters 'A' through 'G' in the data, each of which can have two, three, or four ordinal values. Two of the seven policy components (B & E) require a binary response, while two (A & D) have 3 response options, and the remaining three policy components (C, F, & G) have 4 response options. The data includes quote history for each customer, which is comprised of a selection of all seven options, a point in the shopping history, characteristics of the customer, and characteristics of the quote, including day, time, and cost. The training data include shopping history for 97,009 customers, which shopped for unique 1,809 policies, in which 1,522 policies were purchased.

The initial modeling approach can be explored as 2 binary classification and 5 multiclass classification problems. A full policy prediction would consist of a vector of option predictions from the seven classification problems. The scope of this document will not extend into a full policy prediction, but rather provide potential methods and techniques of creating a prediction containing all seven policy options. In addition to the individual component classification problems, insights regarding the relationship between the last quoted plan and the purchased plan will be explored.

There are additional caveats to this approach. A policy is a combination of seven predicted components, each with their own error. This compounding error will decrease the accuracy of the policy prediction. In

addition, shopping history in the test set has been truncated to simulate making predictions earlier in the shopping window. It is expected that the selection of the last quoted plan as a predicted purchase to be much less accurate in the test set than it is in the training set, which returned approximately 67% accuracy. It is also possible for a customer to purchase a plan that was not viewed in their shopping history.

Literature Review

Researchers from the Foundation for Advancement of Science and Technology's National University of Computer and Emerging Sciences (FAST-NU) in Pakistan studied this data set (Shah & Saeed, 2014). Their approach was based on the last viewed policy for a customer. They imputed 0's for the missing values in location, C_previous, and previous_duration, and they used a prediction model to impute the missing values for risk_factor. The researchers used a data set containing the purchased policies of a customer, not the quoted policies, and used logistic regression, Naive Bayes, SVM, and Random Forest to model the predictions for each policy option. Because their models were resulting in a low accuracy, they modified their method to take into consideration the last quoted policy for each customer. This led to significant improvement in the accuracy of their predictions. The researchers continued to try different methods, such as replacing the policies that underwent the most change with the policies they most frequently changed to. In the end, the researchers found that customers are more likely to select the last quoted policy when the number of quotes they receive increases.

The Data

Data Types

Understanding the basic dimensions and characteristics of response and predictor variables is vital to the modeling process. The Allstate Purchase Prediction data set contains seven response variables (A, B, C, D, E, F, G) which represent different coverage options and, when combined, represent a single policy for a customer. Each coverage option variable is categorical and has its own possible set of values as can be seen in **Table 1**.

Table 1: Possible values of coverage option response variables.

Option Name	Possible Values
A	0, 1, 2
B	0, 1
C	1, 2, 3, 4
D	1, 2, 3
E	0, 1
F	0, 1, 2, 3
G	1, 2, 3, 4

Additionally, there are 18 predictor variables in the data set gathered about the customer who was searching for an insurance policy. These variables range from demographic information, such as marriage status, to information about the customer's site visits, such as time of site visit. Below, **Table 2** summarizes the type and description of each predictor variable.

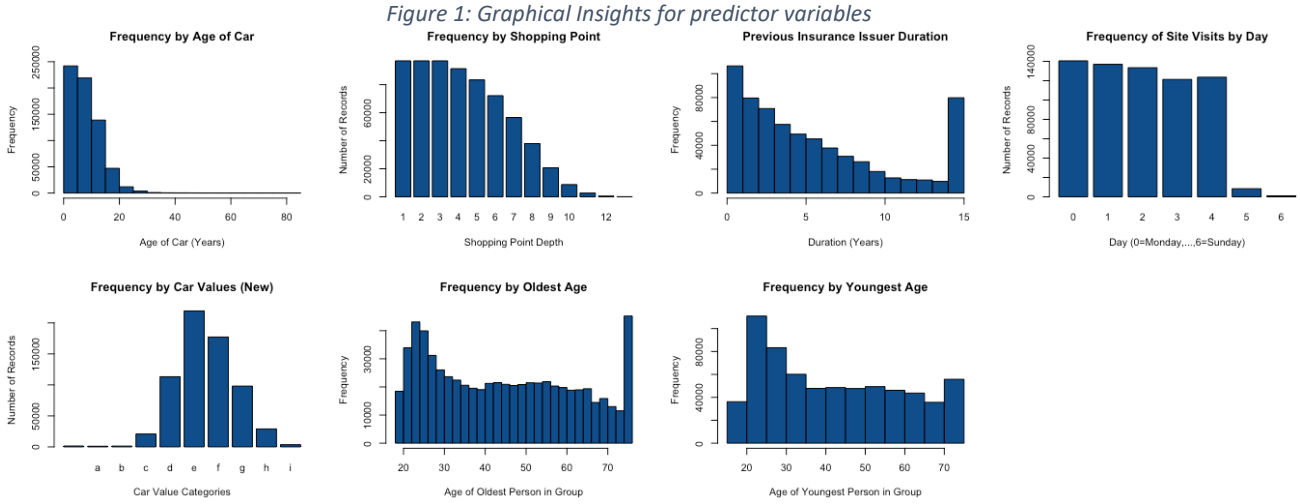
Table 2: Description of predictor variables

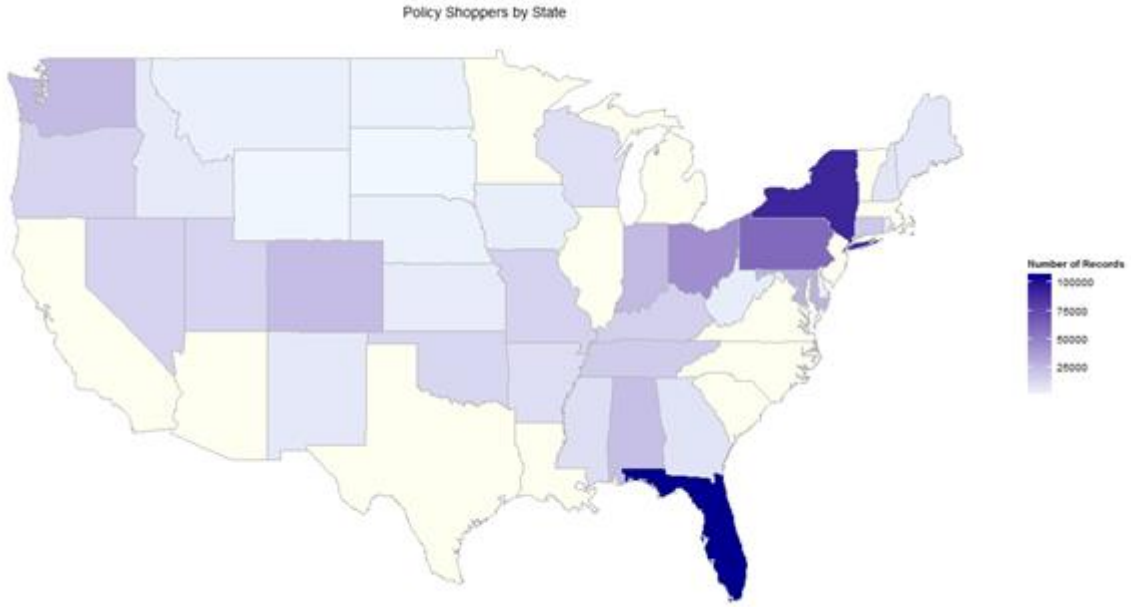
Variable Name	Type	Description
car_value	Categorical	How valuable was the customer's car when new
day	Categorical	Day of the week (0-6, 0=Monday)
state	Categorical	State where shopping point occurred
location	Categorical	Location ID where shopping point occurred
C_previous	Categorical	What the customer formerly had or currently has for product option C (0=nothing, 1, 2, 3, 4)
record_type	Categorical (Binary)	0=shopping point, 1=purchase point
homeowner	Categorical (Binary)	Whether the customer owns a home or not (0=no, 1=yes)
married_couple	Categorical (Binary)	Does the customer group contain a married couple (0=no, 1=yes)
risk_factor	Categorical (Ordinal)	An ordinal assessment of how risky the customer is (1, 2, 3, 4)
car_age	Continuous	Age of the customer's car
age_oldest	Continuous	Age of the oldest person in customer's group
age_youngest	Continuous	Age of the youngest person in customer's group
cost	Continuous	Cost of the quoted coverage options
duration_previous	Discrete	How long (in years) the customer was covered by their previous issuer
shopping_pt	Discrete	Unique identifier for the shopping point of a given customer
group_size	Discrete	How many people will be covered under the policy (1, 2, 3 or 4)
customer_ID	Index	A unique identifier for the customer
time	Interval/Ratio	Time of day (HH:MM)

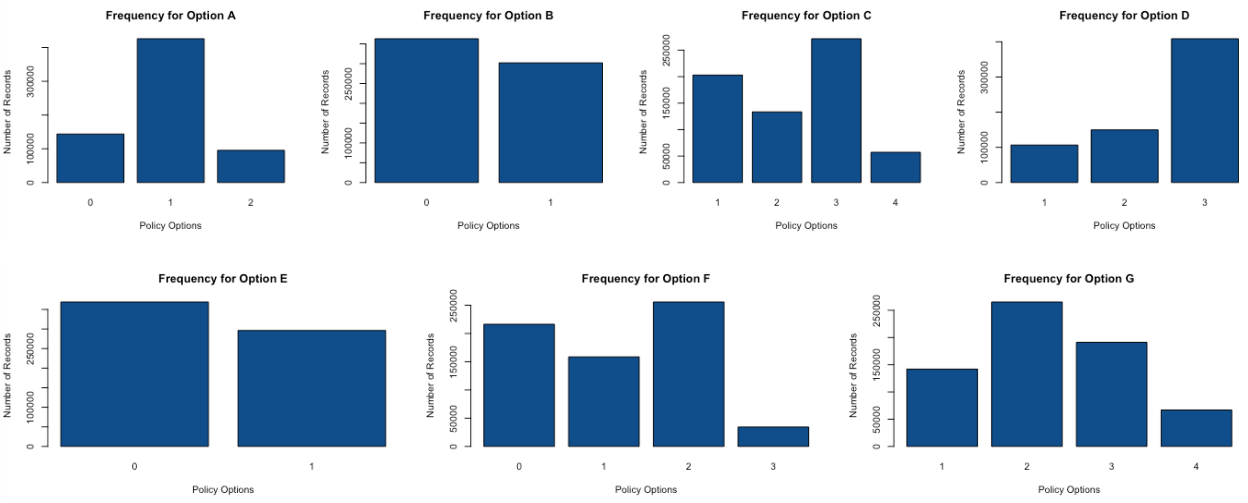
Data Distributions

When summarizing the variables, we will consider their distributions based on number of rows, not grouped over users. While both methods could be informative, it will be useful to know what our data looks like on a row-level for input into our prediction method. **Figure 1** below contains bar and histogram plots for the variables which provide insight to their distributions. Features to look for include long tails (e.g., right tails on car_age, shopping_pt, duration_previous), high frequency in certain categorical values (e.g., day, car_value), and potential anomalies (e.g., spikes at 75 for age_oldest and age_youngest and spike at 15 for duration_previous). Further investigation will be conducted on data that exhibit these features.

Figure 1: Graphical Insights for predictor variables







Data Summaries

While many of the variables have easy to interpret graphical distributions due to their categorical nature, others require a more expansive investigation to better understand them due to their more complex distributions. Looking at descriptive statistics and boxplots of the variables can help provide further information about the variables. Below, **Table 3** shows descriptive statistics on variables that had interesting features in their graphical distributions.

Table 3: Descriptive statistics for select variables

Variable/ Stat	shopping_pt	day	car_age	age_oldest	age_youngest	duration_previous	cost
Minimum	1	0	0	18	16	0	260
1st Quartile	2	1	3	28	26	2	605
Median	4	2	7	44	40	5	635
Mean	4.22	1.969	8.139	44.99	42.58	6.004	635.8
3rd Quartile	6	3	12	60	57	9	665
Maximum	13	6	85	75	75	15	922

When considering these descriptive statistics, it is worth noting any major differences in the mean and the median. For this data set, each variable's mean and median are very similar. It is also important to note large differences in the minimum and first quartile in comparison to differences between the third quartile and the maximum since this may indicate skew due to outliers. Variables that contain potential outliers are discussed in the next section.

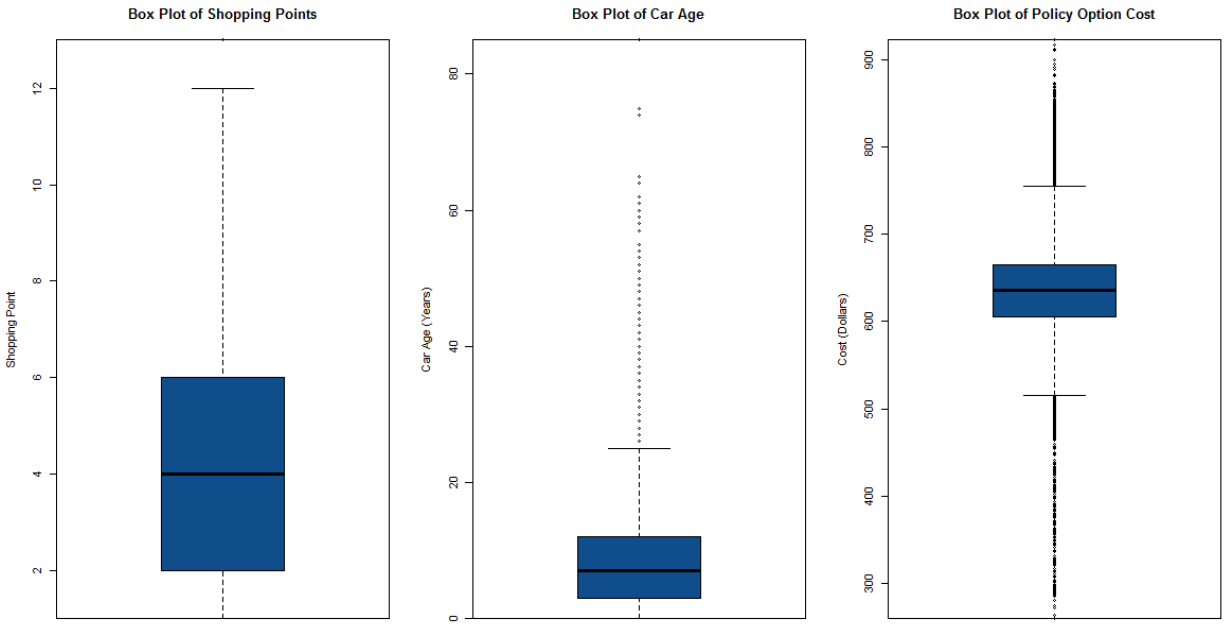
Outliers

Of the variables in the data set, the graphical distribution and descriptive statistics indicate that shopping_pt, car_age, and cost might contain outliers. It is important to be aware of outliers because, while they are not problematic by rule, they may impact analysis results. One useful way to look for outliers is to create boxplots of data as can be seen in **Figure 2** below. Often, values beyond the whiskers of the plot are considered outliers as they lie beyond the interquartile range; however, this is not a hard rule as each programming language has different defaults for determining the whisker length.

On detailed analysis, it was found that shopping_points, car_age, and cost have influential data points rather than true outliers. For example, there are a few long term customers with insurance company will

have higher shopping points, similarly there could be collection cars from 1930's and policy option cost can be higher or lower based on customer's demographics and purchased coverages and location demographics. It was decided to leave influential data points as-is.

Figure 2: Boxplots of select variables



Missing Values

While it is important to consider the values of the each variable available, it is also important to consider the data that is missing in each record. Variables with missing, or NA, values in train and test data sets are shown in **Table 4**.

Table 4: Missing values by variable (Train & Test data sets)

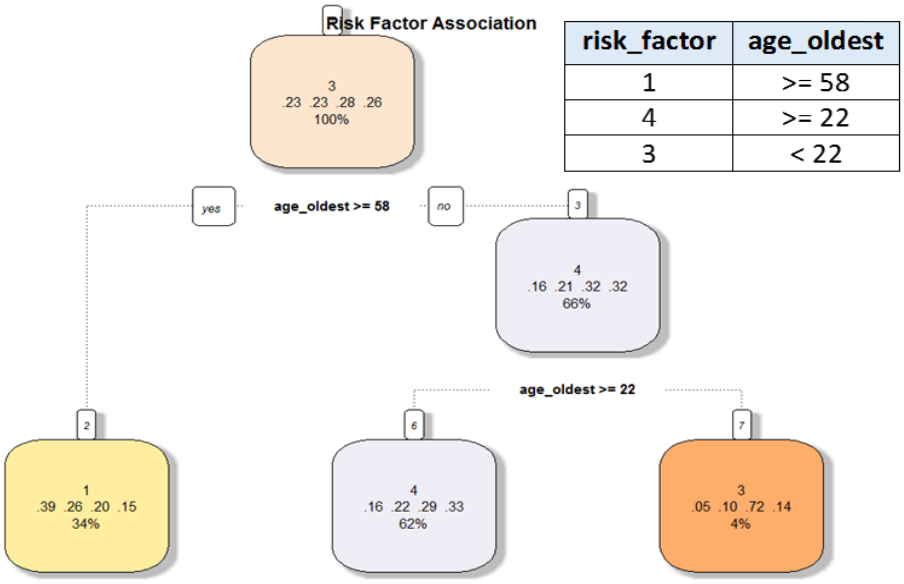
Data Set	Variable	risk_factor	C_previous	duration_previous	location
Train	Number of NA	240418	18711	18711	None
	NA % of Total	36.14%	2.81%	2.81%	None
Test	Number of NA	75487	9769	9769	678
	NA % of Total	37.96%	4.91%	4.91%	0.34%

It's worth noting here that there are a lot of missing values from the risk_factor variable. In fact, of the 665,249 records, about 36% of them are missing. Further analysis of risk_factor variable is done using decision tree to find any associations between different variables to impute the missing values. **Figure 3** shows the decision tree of risk_factor variable and it shows that risk_factor values vary based on different values of age_oldest variable. For example, risk_factor is 1 for age_oldest >= 58.

The search for missing values coupled with the graphical distribution of C_previous has revealed that it seems as though the data dictionary from the source of the data was incorrect: a status of no previous policy for option C does not have a value of 0, but has a value of NA.

Additionally, the location variable is only missing in test data. Location data has strong relationship with state. Missing values will be imputed based on corresponding state value.

Figure 3: risk_factor Association - Decision Tree



risk_factor	age_oldest
1	>= 58
4	>= 22
3	< 22

Exploratory Data Analysis

The statistical problem is a classification problem. The seven coverage options that are to be predicted are all categorical columns. The variable shopping_pt has a different distribution in the training set than it does in the test set, as shown in **Figure 4**. Because the prediction challenge is set to simulate uncertainty and encourage the prediction to be made earlier in the shopping window, model techniques that cater to earlier predictions will be explored. One possible method is to provide stronger weights to more recent quotes in the shopping window when training a model. Since there is a time component to the problem, it may make sense to look at time-series models.

Figure 4: shopping_pt variable distribution in training and test data sets

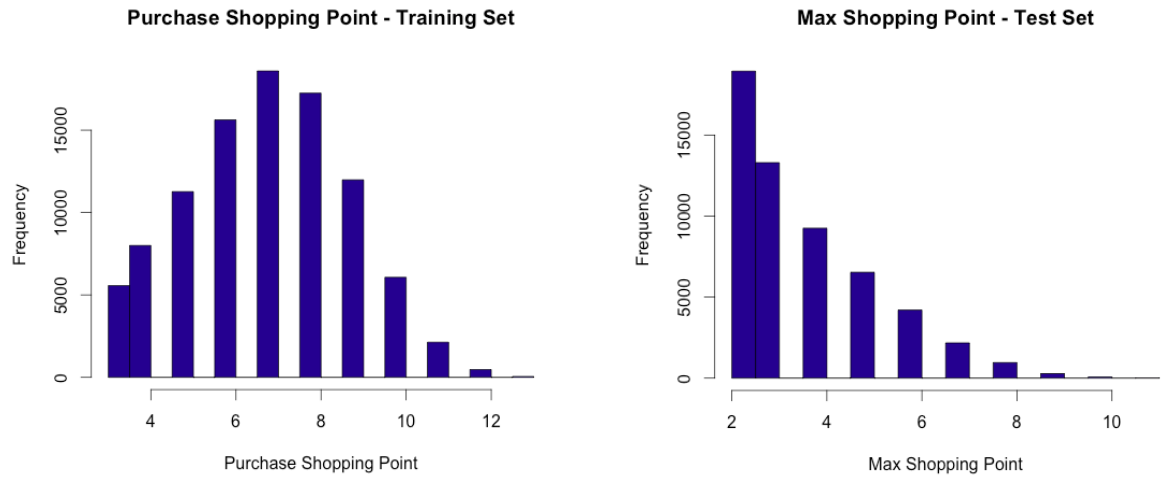
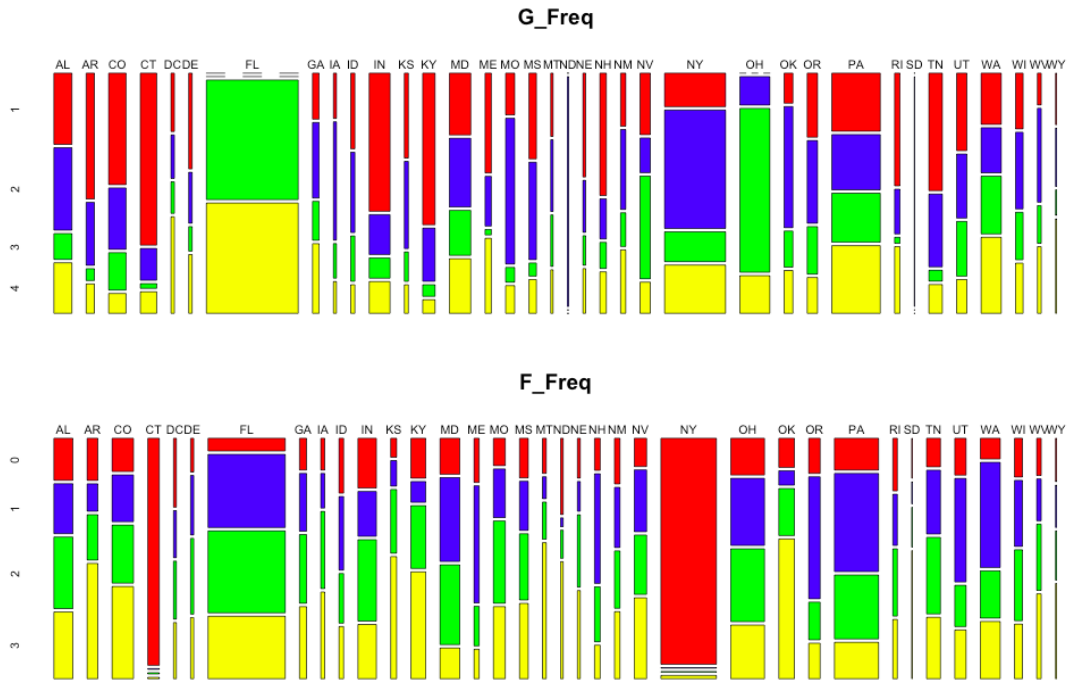


Figure 5 shows the frequency of each policy option by state, which we studied during the exploratory analysis. Interesting relationships discovered may imply policy restriction in certain states. In Florida, option 'F', policy selections '1' & '2' were not purchased at all. Similar insights were discovered in North Dakota, in which only option '2' was selected. Policy option 'G' also had unique characteristics within certain states. In New York and Connecticut, option '0' was purchased in an overwhelming majority. This insight can provide rule-based predictions for options within states.

Figure 5: Policy G & F option distributions by state



We look next on relationships between the variables themselves. This will potentially help with variable selection for our model. We ran a correlation analysis for all the numerical factors. The correlations are in **Table 5**.

Table 5: Correlations > abs(.30)

Var1	Var2	Correlation
age_youngest	age_oldest	0.9142
group_size	married_couple	0.7803
shopping_pt	record_type	0.4511
homeowner	age_oldest	0.4341
homeowner	age_youngest	0.3800
risk_factor	age_oldest	-0.3031

There is not much information to extract by looking at the correlation matrix of the variables other than the age of the groups are highly correlated. We may want to look at the difference between age_youngest and age_oldest to see if the creates a new, predictive variable. The fact that the variables are generally not correlated will help in our modeling process if we decide to utilize regressions. Next, we examine any relationships between the predictors and the seven response variables. We produced a table that shows the percent frequency for each coverage option by each predictor. **Table 6** below shows the top 20 relationships.

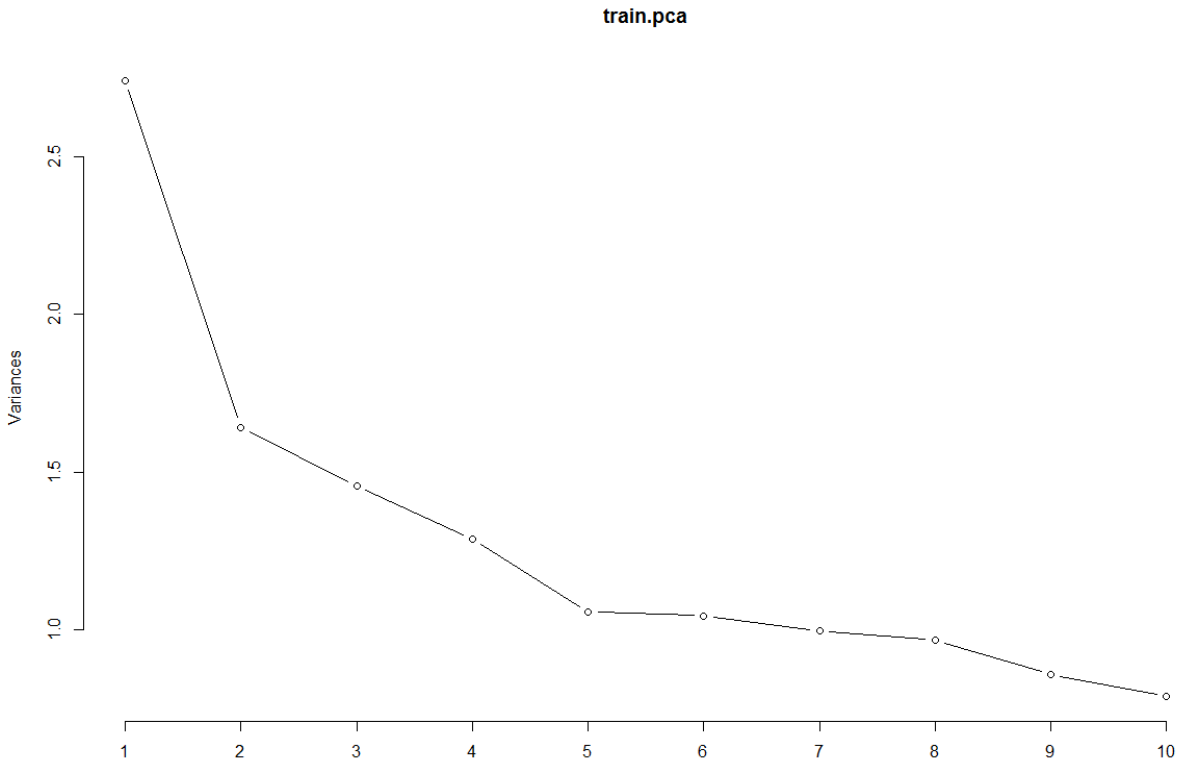
Table 6: Top 20 relationships

Coverage Option	Coverage Option Value	Variable	Variable Value	Pct Freq
D	3	married_couple	0	50
D	3	group_size	1	49
A	1	married_couple	0	48
A	1	group_size	1	47
E	0	married_couple	0	43
E	0	group_size	1	42
B	0	married_couple	0	42
B	0	group_size	1	41
D	3	homeowner	1	38
B	1	married_couple	0	37
B	1	group_size	1	37
A	1	homeowner	1	36
E	1	married_couple	0	36
E	1	group_size	1	35
D	3	C_previous	3	34
C	3	C_previous	3	31
G	2	married_couple	0	31
G	2	group_size	1	31
C	3	married_couple	0	30

The first row in the table, for example, means 50% of the D-3 coverage option was from variable married_couple with a value of 0. In other words, single people make up 50% of the D-3 coverage option. The table is helpful for categorical variables that have few values. Column age_old and age_young are examples of continuous variables in the dataset. We will compute a table that shows the average age_old for each coverage option to see any biases. The results in this table help provide a list of strong relationships between predictors and response variables.

We then ran a PCA to see find what variables have the highest influence. As shown in **Figure 6**, it takes up to five principal components to explain more than 50% of the variance.

Figure 6: Training data PCA



To determine the influence of each variable, we used the first five PCs. We ranked each variable from 1-15 for each PC. Then we took the rank and multiplied each by the proportion of variance explained. Finally, we summed up those weights for each variable and re-ranked the variables. The top five factors are displayed in **Table 7** below.

Table 7: Top 5 Risk Factors

Risk factors
risk_factor
age_oldest
age_youngest
state_num
married_couple

Predictive Modeling: Methods and Results

Data Preparation

Before we start building models, we need to handle missing values and outliers present in data sets, as well as create training, validation, and test data sets.

Imputing Missing Values

As mentioned previously, risk_factor, C_previous, and duration_previous have missing values. Missing values are imputed using the logic specified in **Table 8**.

Table 8: Missing Values Impute Logic (Train & Test data sets)

Data set	Variable	Imputed Variable	Missing Count	Impute Logic
train	risk_factor	Risk_factor_imp	240,418	Based on formula If age_oldest >= 58 then risk_factor = 1 If age_oldest >= 22 then risk_factor = 4 Else 3
	C_previous	C_previous_imp	18,711	NAs are replaced with 0 as per the variable definition
	duration_previous	duration_previous_imp	18,711	NAs are replaced with 0 as corresponding C_previous is 0.
test	risk_factor	Risk_factor_imp	75,487	Based on formula If age_oldest >= 58 then risk_factor = 1 If age_oldest >= 22 then risk_factor = 4 Else 3
	C_previous	C_previous_imp	9,769	NAs are replaced with 0 as per the variable definition
	duration_previous	duration_previous_imp	9,769	NAs are replaced with 0 as corresponding C_previous is 0.
	location	location_imp	678	Highest frequency location for corresponding state.

Training and Validation Data Sets

The Allstate data set was provided in two different data sets: train and test. A sample from train data set was created for the customers who purchased the policy (train.purchase.m). This data was further divided in trainSubset and validSubset in a 75/25 ratio for model build and validation. **Table 9** below shows the observation counts for each data set.

Table 9: Data set information

Data set	Count	Comments
train	665,249	provided
test	198,856	provided
train.purchase.m	97,009	Sample of customers who purchased policy
trainsubset	72,757	75% of train.purchase.m
validsubset	24,252	25% of train.purchase.m

Models

Because a policy is composed of seven policy options and greater than 1500 unique policies were purchased, modeling each policy option (A-G) separately was explored. A baseline model was fit using the last quoted option for each customer and will be leveraged as a benchmark for prediction methods. One possible method for predicting a complete policy is to insert a predicted option and leverage the last quote for the remaining options. For example, if we let (A_Q, B_Q, C_Q, …, G_Q) represent the last quoted value before purchase for each option and (**A_P, B_P ,C_P, …, G_P**) represented a predicted option value generated by a predictive model. A final policy prediction may consist of a combination of last quoted and predicted values (e.g. (A_Q, B_Q, C_Q, **D_p**, E_Q, F_Q, ***G_P***)). The decision to use predicted option values vs. the last quoted option value can be assessed through model performance metrics for each individual option. The scope of this document does not include the final policy prediction and will only include individual option predictions (A-G).

Analysis to identify which option allowed for the most improvement over the baseline prediction was performed. **Table 10** identifies option G as the option most often changed from its last quote; therefore, the first and most detailed modeling suite will focus on predicting option G. (A_Q, B_Q, C_Q, D_Q, E_Q, F_Q, ***G_P***).

Table 10: Options Change Frequencies from Last Quotes

A_change	B_change	C_change	D_change	E_change	F_change	G_change
6894	6392	6716	4927	6067	7039	13174
7.11%	6.59%	6.92%	5.08%	6.25%	7.26%	13.58%

We will use the out-of-sample subset of the data to validate and assess the performance of each model. The misclassification rate for the baseline model, which uses the last quote for a purchase prediction, is displayed in **Table 11**. A prediction model that performs better than the respective baseline model will be considered for the policy prediction.

Table 11: Baseline Misclassification Rates for Each Option

Option_A	Option_B	Option_C	Option_D	Option_E	Option_F	Option_G
1767	1610	1674	1174	1520	1774	3334
7.29%	6.64%	6.90%	4.84%	6.27%	7.31%	13.75%

After describing the modeling process for option G in detail, we will follow a similar model discovery and fitting process for the other policy options using the same methods (LDA, Random Forest, Boosted Trees, SVM, and KNN) but will only report the results from the best models in each method in the Model Comparison section in order to determine the method and model with the best predictive accuracy.

Option G Models

Linear Discriminant Analysis

Often considered an alternative to logistic regression and popular when having more than 2 cases, as we do with many of our policy options, Linear Discriminant Analysis models the predictors of each policy option so that we can find the linear combination of them that best separates the response groups.

Modeling for the policy options proved somewhat difficult as models involving all available predictors ran up against computational memory constraints in R statistical software. Therefore, a number of subset models were employed which left out some of the less interpretable but highly dimensional variables such as location and customer_ID. Examples of these models for option G can be seen in **Table 12** below.

Table 12: Various model specifications and misclassification errors for Option G.

Model	Misclassification Error on Training set	Misclassification Error on Validation set	Model specification
Full Model	0.1352	0.1375	model.lda0.g<- lda(G ~ lastQuoted_G + risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day + shopping_pt + state + Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4)
Only last quoted	0.1352	0.1375	model.lda1.g <- lda(G ~ lastQuoted_G)
Full model minus last quoted	0.1667	0.1686	model.lda2.g <- lda(G ~ risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day + shopping_pt + state + Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4)
Full model minus last quoted and second to last quoted	0.2093	0.2074	model.lda3.g <- lda(G ~ risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day + shopping_pt + state + Quoted_G_minus3 + Quoted_G_minus4)

As mentioned previously, the data set was split into training and validation groups, and in **Table 12**, the misclassification error of the model can be seen for both. It is worth noting that having more predictors does not necessarily improve misclassification error. Whereas, the removal of *lastQuoted* from the model increases the misclassification error on the out-of-sample validation set from 13.75% to 16.86%. In fact, the LDA model which only contained *lastQuoted* performed equally as well as the fully specified model and better than any model without it, such as the third model in **Table 12**. The final model listed in **Table 12** shows that recent quotes other than the last one are also important predictors for option G. Through many iterations of such models, it was discovered that the recently quoted policies, specifically the last quoted, were strongly predictive of the policy class.

Table 13 and **Table 14** below show the training and validation confusion matrices for the fully specified model (*model.lda0.g*).

Table 13: Confusion matrix on training data for model.lda0.g where Option G is 1,2,3, or 4 (%ages)

Linear Discriminant Analysis		Option G			
		1	2	3	4
Prediction	1	0.812	0.132	0.041	0.012
	2	0.065	0.857	0.059	0.017
	3	0.011	0.027	0.924	0.036
	4	0.021	0.061	0.093	0.823

Table 14: Confusion matrix on validation data for model.lda0.g where Option G is 1,2,3, or 4 (%ages)

Linear Discriminant Analysis		Option G			
		1	2	3	4
Prediction	1	0.804	0.136	0.047	0.010
	2	0.062	0.862	0.057	0.017
	3	0.014	0.030	0.914	0.040
	4	0.026	0.050	0.100	0.822

These same results were found across variables: models which included cost and the previous 4 policy quotes were the most predictive for all policies. In fact, the inclusion of other variables outside of lastQuoted, Quoted_minus2, Quoted_minus3, and Quoted_minus4 for each policy did not improve the misclassification error of the models.

While linear discriminant analysis seemed to offer some benefits, it is a relatively simple model, and we could find benefit, especially for option G, in moving to a more complex model in order to improve over the baseline model of simply using last quoted as a predictor for what will be purchased.

Random Forest

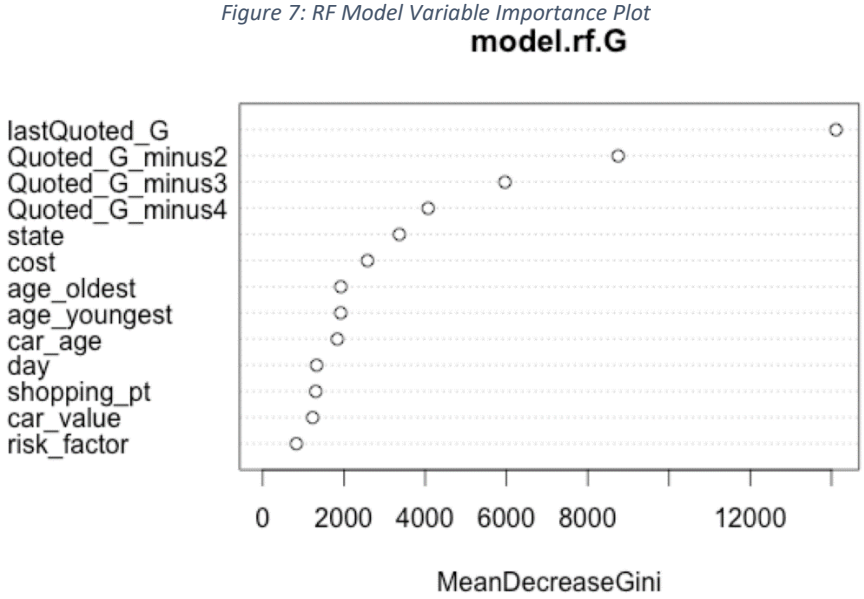
A Random Forest model was created to predict option G based on the four previous quotes for each customer, as well as their state, age, risk factor, car age, car value, cost of policy, day they shopped and the purchase shopping point. Cross validation methods were leveraged to tune the number of variables considered at each split (4) and the number of trees considered (500). **Table 15** shows the predictions. The tuned Random Forest model returned an out-of-bag (OOB) error rate of 13.61% and an out-of-sample validation error rate of 13.87%, which was not an improvement on the baseline model out-of-sample validation error of 13.75%.

Table 15: Random Forest Option G Predictions (%ages)

Random Forest		Option G			
		1	2	3	4
Prediction	1	0.805	0.137	0.046	0.011
	2	0.062	0.863	0.056	0.018
	3	0.018	0.036	0.905	0.042
	4	0.020	0.042	0.098	0.839

In **Figure 7**, the importance measured by a decrease in the Gini Index displays the importance of the last quoted option G value, along with the importance of a quote decreasing as the quote moves away from the purchase point. We can also see the variable state has a relative high importance, which implies there are variations in the G option selected across states, which aligns with the insight discovered during the initial EDA.

Figure 7: RF Model Variable Importance Plot



Additional rule-based analysis was selected based on insights discovered during the EDA. Consumers in the state of Florida did not select class 1 or 2 for option G. Any prediction for Florida residents which resulted in class 2 were adjusted to class 3 manually. Further decision tree analysis can be applied on this subset of predictions.

Boosted Tree Model

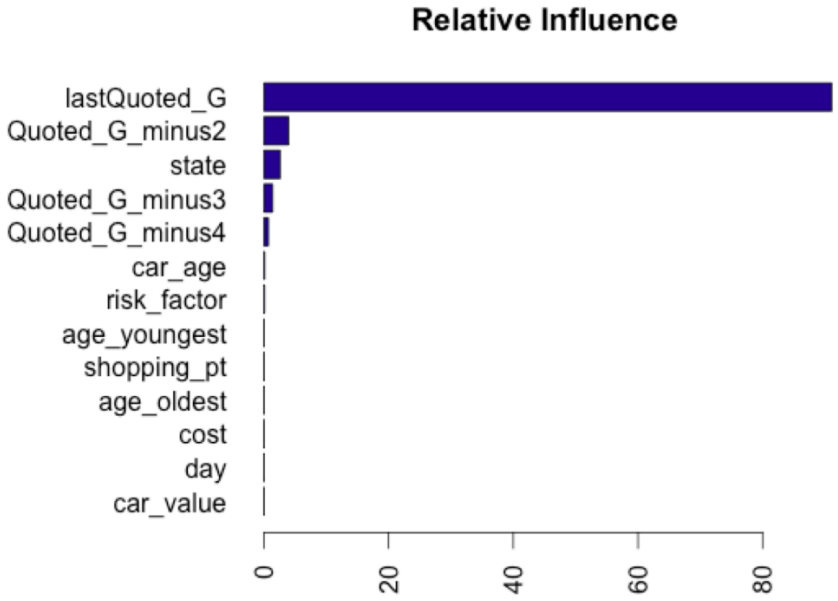
A Boosted Trees model was fit to predict option G based on the four previous quotes for each customer, as well as their state, age, risk_factor, car_age, car_value, cost, day they shopped, and the purchase shopping_point. An interaction depth of 4, shrinkage parameter of 0.01 , and number of trees equal to 1000 was selected through cross validation tuning. **Table 16** below shows the predictions.

Table 16: GBM Option G Predictions (%ages)

GBM		Option G			
		1	2	3	4
Prediction	1	0.807	0.134	0.048	0.012
	2	0.063	0.864	0.055	0.018
	3	0.016	0.033	0.909	0.042
	4	0.021	0.042	0.096	0.840

The variable relative influence, shown in **Figure 8**, is similar to the insight derived from the random forest model in that the most recent quoted G value had the most influence, decreasing as the shopper moved farther away from the purchase point. However, the consumer's state had more importance in the boosted model. The GBM returned a validation error of 13.64%, a decrease from the baseline model validation error of 13.75%.

Figure 8: Relative Influence



Supported Vector Machine (SVM)

Support Vector Machine classification models were fit using linear kernel functions. Cross validation (CV) was leveraged to select the parameters cost and gamma, which contribute to the model flexibility and influence the bias-variance trade off. The results of the repeated 10-fold cross validation are presented in **Table 17**. An array of gamma values was explored; it was found that a gamma value of one produced the lowest error. An SVM with a linear kernel, gamma parameter of 1.0, and cost parameter of 0.10 was selected through assessment of the CV error.

Table 17: SVM Cross Validation

	Cost	Gamma	Error	Dispersion
SVM_1	0.010	1.000	0.1352	0.004
SVM_2	0.100	1.000	0.1336	0.004
SVM_3	0.500	1.000	0.1337	0.004
SVM_4	1.000	1.000	0.1338	0.004

The tuned SVM model was fit to predict option G based on the four previous quotes for each customer, as well as their state, age, risk factor, car age, car value, cost, day they shopped and the purchase shopping point. **Table 18** below shows the predictions on out of sample validation set. The SVM returned a validation error of 13.63%, a decrease from the baseline model validation error of 13.75%.

Table 18: SVM Option G Predictions (%ages)

SVM		Option G			
		1	2	3	4
Prediction	1	0.805	0.137	0.048	0.011
	2	0.063	0.866	0.055	0.017
	3	0.014	0.030	0.915	0.041
	4	0.026	0.051	0.100	0.823

k-Nearest Neighbor (KNN)

The advantage of KNN is that there is only one parameter to consider (the *k*, which serves as a tuning parameter). Although a KNN model was initially fitted, the computational power required to run the algorithm was not sufficient. To solve this issue a sample of the full training dataset was used for analysis purposes. A 25% sample of the 97,009 training observations was used, allowing the KNN algorithm to work on a dataset consisting of 24,252 observations. The seed was set to 1 and the explanatory variables used were the following: group_size, homeowner, car_age, risk_factor, age_oldest, age_youngest, married_couple, C_previous, and duration_previous, for a total of nine explanatory variables. The target variable was G. A *k* of 9 was selected and the following contingency table (**Table 19**) was generated. Using the nine explanatory variables only, the prediction group 2 correctly predicted the outcome (compared to the other groups).

Table 19: KNN Option G Predictions (%ages)

KNN		Option G			
		1	2	3	4
Prediction	1	0.828	0.121	0.039	0.011
	2	0.089	0.826	0.064	0.020
	3	0.015	0.046	0.873	0.065
	4	0.015	0.036	0.089	0.850

Comparison of Results

Having described the model discovery and fitting process for option G, we will now use misclassification rates on a validation set in order to compare the best models from each modeling method and determine which method and model has the best predictive accuracy for option G.

Following the model comparison for option G, we will compare the best models from each method for the other policy options to see if those models can beat the baseline model. For the sake of brevity, we will not go into detail on the modeling process for each policy as we did for option G but will only be comparing the different modeling methods using misclassification rate.

Option G Model Comparison

Five models were fit in an attempt to outperform the baseline model for option G (as shown in **Table 20**), which leveraged the last quote as its only predictor variable and had an out-of-sample validation error rate of 0.1352. The LDA model had the exact same prediction as the baseline model, implying the model placed too much emphasis on the lastQuoted_G variable. The Random Forest model performed well on the training set but poorly on the out-of-sample validation set, suggesting the model may have overfit the data. The Boosted Trees and SVM models performed best, with a validation error rate of 13.64% and 13.63%, respectively.

Table 20: Option G Model Error Rates and Kappa Metrics

Option G Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.1352	0.1375	0.8021
LDA	0.1352	0.1375	0.8021
Random Forest	0.1210	0.1387	0.8001
Boosted Trees	0.1318	0.1364	0.8034
SVM	0.1337	0.1363	0.8038
KNN	0.1694	0.1560	0.7729

Option A Model Comparison

First, a baseline model was fit using lastQuoted_A as the only predictor of A. Then Random Forest was used to select the most important predictors to use in the other models. Factor variables with more than 53 levels, such as time and location, were excluded from the variable selection process. The Random Forest model revealed that homeowner, married_couple, and group_size were not very important, so they were excluded from subsequent models. The Boosted Trees model performed the best, while the KNN model performed the worst. **Table 21** shows the error rates and Kappa values for all fitted models for Option A.

Table 21: Option A Model Error Rates and Kappa Metrics

Option A Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0705	0.0729	0.8646
LDA	0.0705	0.0729	0.8646
Random Forest	0.0531	0.0705	0.8691
Boosted Trees	0.0647	0.0689	0.8720
SVM	0.0706	0.0723	0.8652
KNN	0.0698	0.0741	0.8604

Option B Model Comparison

The baseline model performed the best. As with most of the other plan options, the best predictor is lastQuoted_B. This variable has by far the most influence on each of the models. As seen in the results of the validation set, the Random Forest and Boosted Trees methods performed equally as well as the baseline method. Because the baseline model is simplistic compared to the tree methods, the baseline model is preferable. One other note is the Random Forest model had the lowest training error rate. **Table 22** shows the error rates and Kappa values for all fitted models for Option B.

Table 22: Option B Model Error Rates and Kappa Metrics

Option B Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0657	0.0664	0.9074
LDA	0.0657	0.0664	0.8667
Random Forest	0.0553	0.0682	0.8630
Boosted Trees	0.0657	0.0664	0.8667
SVM	0.0657	0.0664	0.8667
KNN	0.0934	0.0947	0.8096

Option C Model Comparison

Five models were fit in an attempt to outperform the baseline model for option C (as shown in **Table 23**), which leveraged the last quote as its only predictor variable and had an out-of-sample validation error rate of 0.069. Variable selection was performed using the mean decrease Gini index derived from a Random Forest model fit with all predictor variables and the four quotes before purchase for option C, along with cross validation to tune each model's parameters. The KNN model performed the worst of the models for option C. The LDA and Random Forest models had the exact same predictions as the baseline model, implying that these models placed too much emphasis on the lastQuoted_C variable. The SVM and Boosted Trees models performed a little worse than the baseline model. The LDA model is chosen as final prediction for option C over Random Forest based on its smaller Train Error Rate and execution speed.

Table 23: Option C Model Error Rates and Kappa Metrics

Option C Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0690	0.0690	0.9018
LDA	0.0690	0.0690	0.9018
Random Forest	0.0693	0.0690	0.9018
Boosted Trees	0.0691	0.0693	0.9014
SVM	0.0693	0.0692	0.9015
KNN	0.1091	0.0984	0.8591

Option D Model Comparison

For option D, another five models (shown in **Table 24**) were fit to try to outperform the baseline model, which leveraged the last quote as its only predictor variable. The baseline model had an out-of-sample validation error rate of 0.0484. The mean decrease Gini index derived from a Random Forest model fit with all predictor variables and the four quotes before purchase for option D helped with variable selection. Cross validation was used to tune each model's parameters. The KNN model was the only model that performed worse than the baseline model. The LDA model had the exact same prediction as the baseline model, implying the model placed too much emphasis on the lastQuoted_D variable. The Random Forest, SVM, and Boosted Trees model all outperformed the baseline model, but it was the Boosted Trees model that performed best and will be selected as the final prediction for option D.

Table 24: Option D Model Error Rates and Kappa Metrics

Option D Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0516	0.0484	0.9074
LDA	0.0516	0.0484	0.9074
Random Forest	0.0463	0.0480	0.9080
Boosted Trees	0.0512	0.0476	0.9091
SVM	0.0516	0.0482	0.9078
KNN	0.0775	0.0751	0.8535

Option E Model Comparison

Again, five different modeling techniques were utilized and compared to the baseline model that used the lastQuoted item. Much like the other options, the predicted model results were not better than the lastQuoted variable. **Table 25** shows the error rates and Kappa values for all fitted models for Option E.

Table 25: Option E Model Error Rates and Kappa Metrics

Option E Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0627	0.0627	0.8831
LDA	0.0625	0.0627	0.8737
Random Forest	0.0627	0.0673	0.8642
Boosted Trees	0.0621	0.0627	0.8747
SVM	0.0627	0.0628	0.8737
KNN	0.0811	0.0741	0.8604

Option F Model Comparison

Using five different models compared to the baseline of lastQuoted, the predicted models for F struggled to beat the baseline. This likely says more about the relationship between the lastQuoted and purchased variables than the power of the models. However, one modeling technique, Boosted Trees, showed some improvement over the baseline performance, beating training misclassification (0.0724 versus 0.0708) and test misclassification (0.0731 versus 0.0724). **Table 26** shows the error rates and Kappa values for all fitted models for Option F.

Table 26: Option F Model Error Rates and Kappa Metrics

Option F Model	Train Error Rate	Validation Error Rate	Validation Kappa
Baseline	0.0724	0.0731	0.8941
LDA	0.0724	0.0731	0.8941
Random Forest	0.0622	0.0725	0.8950
Boosted Trees	0.0708	0.0724	0.8951
SVM	0.0724	0.0739	0.8941
KNN	0.1090	0.1103	0.8392

Conclusions

In this report, we developed multiple machine-learning models, including LDA, KNN, Boosted Trees, Random Forest, and SVM, to predict which of seven Allstate insurance policy options a customer purchased. We used each customer's shopping and quote history in order to make our predictions. We found that the last quoted plan variable was most predictive of which policy option the customer chose. Although a full policy prediction would involve a vector of option predictions from each of the seven classification problems, we focused instead on developing potential methods for creating a prediction containing all seven policy options.

Overall, our models resulted in good quality predictions, outperforming the baseline model for four of the seven options. The validation error rates were low and the validation kappa were relatively high, suggesting that our models perform well. However, we recognize that our models can certainly improve. An avenue that we would like to explore in the future is developing models that generate predictions earlier in the shopping window. While the last quoted plan is a strong predictor, we think that we are capable of developing more nuanced models that predict the policy option sooner. The sooner we are able to predict the customer's policy preference, the more likely we are to retain the customer's

business. This is something we think worth researching and exploring if we have the opportunity to work with this data set more in the future. In the meantime, we have produced a suite of models that produce reliable predictions for the Allstate insurance policy options.

Bibliography

Shah, S. A., & Saeed, M. (2014). Predicting Purchased Policy for Customers in Allstate Purchase Prediction Challenge on Kaggle. *Proceeding of the International Conference on Artificial Intelligence and Computer Science,* 120-128. Retrieved March 11, 2017, from https://worldconferences.net/proceedings/aics2014/PAPER%20AICS/A049%20-%20PREDICTING%20PURCHASED%20POLICY%20-%20SABA%20ARSLAN.pdf.

Appendices

Appendix A: R Code

```
#PRED 454 Advanced Modeling

#Allstate Purchase Prediction Challenge


# Install Packages if they don't currently exist
list.of.packages <- c("doBy"
                      ,"lazyeval"
                      ,"psych"
                      ,"lars"
                      ,"GGally"
                      ,"ggplot2"
                      ,"grid"
                      ,"gridExtra"
                      ,"corrgram"
                      ,"corrplot"
                      ,"leaps"
                      ,"glmnet"
                      ,"MASS"
                      ,"gbm"
                      ,"tree"
                      ,"rpart"
                      ,"rpart.plot"
                      ,"rattle"
                      ,"gam"
                      ,"class"
                      ,"e1071"
                      ,"randomForest"
                      ,"doParallel"
                      ,"iterators"
                      ,"foreach"
                      ,"parallel"
                      ,"lattice"
                      ,"caret"
                      ,"data.table"
                      ,"plyr"
                      ,"maps"
                      ,"reshape"
                      ,"reshape2"
                      ,"nnet"
                      ,"mlogit"
                      ,"e1071"
                      ,"gmodels")


new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)


# Load all packages
lapply(list.of.packages, require, character.only = TRUE)


#data.table cheat sheet.
#https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf


#####################################
## LOADING DATA ##
#####################################
```

```
# set to your local directory. We will each have to edit this line of code.
path <- "C:/Users/elfty/Desktop/Sherman/MSPA/P454/Project/" #shermanpath
#path <- "/Users/paulbertucci/Desktop/MSPA/PRED454_AdvancedModeling/FinalProject/AllState" #paulpath
path <- "/Users/annie/Desktop/Northwestern/PREDICT_454/Allstate" #anniepath
setwd("/Users/annie/Desktop/Northwestern/PREDICT_454/Allstate")


#load the train and the test data
train <- read.csv(file.path(path,"train.csv"), stringsAsFactors=TRUE)
test <- read.csv(file.path(path,"test_v2.csv"), stringsAsFactors=TRUE)


# Explore the training data -- how big is it, what types of variables included, distributions and missing
values.
class(train) # data.frame
dim(train) # 665249     25
names(train)
str(train) # all integer except time, state, and car_value are factor
summary(train) # some missing data


# Check for NAs
colSums(is.na(train))[colSums(is.na(train)) > 0]
# <---Ahmar: Start of missing values Impute ---------------->
# Check for NAs %ages
round(colSums(is.na(train))[colSums(is.na(train)) > 0] * 100/ dim(train)[1],2)


# variables with missing data:
# risk_factor   C_previous    duration_previous
# 240418        18711         18711
# variable with missing data in %age:
#risk_factor        C_previous duration_previous
#36.14              2.81              2.81


# Finding association between C-previous & duration_previous
table(train$C_previous,train$duration_previous, exclude =NULL)


table(train$C_previous, exclude=NULL)
table(train$risk_factor, exclude=NULL)


# Finding association between risk_factor and other variables.
set.seed(1)
tree.x <- rpart(risk_factor ~ car_value + homeowner+married_couple+car_age+age_oldest + age_youngest +
duration_previous+group_size    , data = train, method = "class")
tree.x # splits on age_oldest
prp(tree.x)
fancyRpartPlot(tree.x,main="Risk Factor Association", sub = "   ", tweak=1, palettes=c( "YlOrRd")) #
unreadable


# Imputing missing values
# risk_factor
train$risk_factor_imp = train$risk_factor
train$risk_factor_imp[is.na(train$risk_factor_imp) & train$age_oldest >=58] <-
1;table(train$risk_factor_imp, exclude=NULL)
train$risk_factor_imp[is.na(train$risk_factor_imp) & train$age_oldest>=22] <-
4;table(train$risk_factor_imp, exclude=NULL)
train$risk_factor_imp[is.na(train$risk_factor_imp) & train$age_oldest<22] <-
3;table(train$risk_factor_imp, exclude=NULL)


# C_previous
train$C_previous_imp <- train$C_previous
train$C_previous_imp[is.na(train$C_previous_imp)] <- 0;table(train$C_previous_imp, exclude=NULL)
```

```
# duration_previous
train$duration_previous_imp <- train$duration_previous
train$duration_previous_imp[is.na(train$duration_previous_imp)] <- 0;table(train$duration_previous_imp,
exclude=NULL)


###   Test Data ################
# Check for NAs
colSums(is.na(test))[colSums(is.na(test)) > 0]
# <---Ahmar: Start of missing values Impute -------------->
# Check for NAs %ages
round(colSums(is.na(test))[colSums(is.na(test)) > 0] * 100/ dim(test)[1],2)


# variables with missing data:
#location       risk_factor        C_previous duration_previous
#678             75487              9769              9769
# variable with missing data in %age:
#location       risk_factor        C_previous duration_previous
#0.34             37.96              4.91              4.91


# Finding association between C-previous & duration_previous
table(test$C_previous,test$duration_previous, exclude =NULL)


table(test$C_previous, exclude=NULL)
table(test$risk_factor, exclude=NULL)
table(test$location, exclude=NULL)


# Finding association between risk_factor and other variables.
set.seed(1)
tree.x <- rpart(risk_factor ~ car_value + homeowner+married_couple+car_age+age_oldest + age_youngest +
duration_previous+group_size    , data = test, method = "class")
tree.x # splits on age_oldest
prp(tree.x)
fancyRpartPlot(tree.x,main="Risk Factor Association", sub = "   ", tweak=1, palettes=c( "YlOrRd")) #
unreadable


# state and location
st_loc = data.frame(state=test$state, loc=test$location )


#st_loc
dim(st_loc) #198856      2


colSums(is.na(st_loc))[colSums(is.na(st_loc)) > 0] # 678


# Remove NA locations
st_loc = st_loc[which(!is.na(st_loc$loc)),]


dim(st_loc) #198178      2


# location frequency by state
st_loc = ddply(test, .(state,location), summarize, frequency = (length(state)))


st_loc = st_loc[which(!is.na(st_loc$location)),]


# Maximum location frequency by State
```

```
dfsl = setkeyv(setDT(st_loc), c("state","location"))[,list(mxloc=max(frequency)), by=list(state)]


# Location, maximum frequency by location
st_loc = merge(st_loc, dfsl, by.x=c("state","frequency"), by.y = c("state", "mxloc"))


# Remove duplicate locations by State based on frequency - taking highest location id
st_loc = setkeyv(setDT(st_loc), c("state","frequency"))[,list(mxloc=max(location)),
by=list(state,frequency)]


# Temp df and merge by state
test2 = test #198856 25var
test2 = merge(test2, st_loc, by.x=c("state"), by.y = c("state"), all.x=TRUE)


colSums(is.na(test))[colSums(is.na(test)) > 0]



# Imputing missing values
# risk_factor
test$risk_factor_imp = test$risk_factor
test$risk_factor_imp[is.na(test$risk_factor_imp) & test$age_oldest >=58] <- 1;table(test$risk_factor_imp,
exclude=NULL)
test$risk_factor_imp[is.na(test$risk_factor_imp) & test$age_oldest>=22] <- 4;table(test$risk_factor_imp,
exclude=NULL)
test$risk_factor_imp[is.na(test$risk_factor_imp) & test$age_oldest<22] <- 3;table(test$risk_factor_imp,
exclude=NULL)


# C_previous
test$C_previous_imp <- test$C_previous
test$C_previous_imp[is.na(test$C_previous_imp)] <- 0;table(test$C_previous_imp, exclude=NULL)


# duration_previous
test$duration_previous_imp <- test$duration_previous
test$duration_previous_imp[is.na(test$duration_previous_imp)] <- 0;table(test$duration_previous_imp,
exclude=NULL)



# Replace missing locations from State Level high frequency locations.
test$location_imp <- test$location
test$location_imp[which(is.na(test$location))] <- test2$mxloc[which(is.na(test$location))]


colSums(is.na(test))[colSums(is.na(test)) > 0]


# <----------------- End of Missing Impute ---------------->


# setting variable types, please feel free to change if you think this is incorrect. #PB
# AHM_FV: added C_previous_imp : Need to uncomment this part but make sure other code doesnt break.
#names <-
c('day','location','car_value','A','B','C','D','E','F','G','record_type','homeowner','married_couple','C_p
revious', 'C_previous_imp')
names <-
c('day','location','car_value','A','B','C','D','E','F','G','record_type','homeowner','married_couple','C_p
revious')


train[,names] <- lapply(train[,names] , factor)
test[,names] <- lapply(test[,names] , factor)
```

```
str(train)
summary(train)
#Creating planCombo variable for plan total #PB
train$planCombo<-paste(train$A,train$B,train$C,train$D,train$E,train$F,train$G,sep="")


#creating a dataframe of purchased records #PB
train.purchase<-train[which(train$record_type==1),]
head(train.purchase)


####################################
## EDA ##
####################################


#freq table for each purchase option #PB
apply(train[18:24],2,FUN = table)


#####Begin Data Description - DT 1/22/17
#Graphing policy shoppers by state


#load state codes csv file # Did someone create a csv? I can't find an existing file on Github or Kaggle #
Annie
state_codes <- read.csv(file.choose())
#head(state_codes)


state_df <- as.data.frame(table(train$state)) #turn state data into DF for manipulating purposes
colnames(state_df)[1] <- "postal_code" #rename column for easy merging below for state_combo
#head(state_df)


#merge state data frame with frequency counts and postal codes to get state names
state_combo <- merge(state_codes, state_df, by="postal_code", all=TRUE)
#head(state_combo)


#merge state_combo with state plotting data
names(state_combo)[names(state_combo)=="state"] <- "region" #renaming to do merge for state_total
state_combo$region<-tolower(state_combo$region) #done for merging on region to work


all_states <- map_data("state") #ggplot2 package, load state graphing df


state_total <- merge(all_states,state_combo, by="region", all=TRUE) #combine state graphing df with
frequency data
#head(state_total)


#construct map graph
state_total <- state_total[order(state_total$order),]
p <- ggplot()
p <- p + geom_polygon(data=state_total, aes(x=long, y=lat, group = group,
fill=state_total$Freq),colour="grey70"
) + scale_fill_continuous(low = "aliceblue", high = "darkblue", na.value="ivory",guide="colorbar")
P1 <- p + theme_bw()  + labs(fill = "Number of Records"
                             ,title = "Policy Shoppers by State", x="", y="")
P1 + scale_y_continuous(breaks=c()) + scale_x_continuous(breaks=c()) + theme(panel.border =
element_blank())


###
#Graphical Distributions of Variables
# Disable scientific notation for labeling plots
options(scipen = 10)
```

```
par.default <- par() #save in case needed later
par(mfrow=c(1,2)) #fit more graphs
#barplots for predictor variables # AB: Added train$ since none worked after we removed attach()
par(mfrow=c(3,3)); ylab.box <- "Number of Records"; col.bar = "dodgerblue4"
barplot(table(train$car_value),main="Frequency by Car Values (New)",ylab=ylab.box,xlab="Car Value
Categories",col=col.bar)
barplot(table(train$day),main="Frequency of Site Visits by Day",ylab=ylab.box,xlab="Day
(0=Monday,...,6=Sunday)",col=col.bar)
barplot(table(train$C_previous),main="Frequency by Policy Option C",ylab=ylab.box,xlab="Policy Option C
Choices (0=nothing)",col=col.bar)
barplot(table(train$record_type),main="Frequency of Record Types",ylab=ylab.box,xlab="0 = Shopping Point,
1 = Purchase Point",col=col.bar)
barplot(table(train$homeowner),main="Frequency by Homeowner", ylab=ylab.box,xlab="0 = Not Homeowner, 1 =
Homeowner",col=col.bar)
barplot(table(train$married_couple),main="Frequency by Marital Status",ylab=ylab.box,xlab="0 = Not
Married, 1 = Married",col=col.bar)
barplot(table(train$risk_factor),main="Frequency by Risk Factor",ylab=ylab.box,xlab="Risk Factor
Levels",col=col.bar)
barplot(table(train$shopping_pt),main="Frequency by Shopping Point",ylab=ylab.box,xlab="Shopping Point
Depth",col=col.bar)
barplot(table(train$time),main="Frequency of Site Visits by Time of Day",ylab=ylab.box,xlab="Time
(HH:MM)",col=col.bar,border=NA)


#histograms for predictor variables
col.hist = "dodgerblue4"
par(mfrow=c(1,5))
hist(train$car_age,xlab="Age of Car (Years)", main="Frequency by Age of Car",col=col.hist)
hist(train$age_oldest,xlab="Age of Oldest Person in Group", main="Frequency by Oldest Age",col=col.hist)
hist(train$age_youngest,xlab="Age of Youngest Person in Group", main="Frequency by Youngest
Age",col=col.hist)
hist(train$cost,xlab="Cost (Dollars)", main="Frequency by Cost of Coverage Options",col=col.hist)
hist(train$duration_previous,xlab="Duration (Years)", main="Previous Insurance Issuer
Duration",col=col.hist)


# Plots for Checkpoint 2 Data Check
colplots <- "dodgerblue4"
ylab.box <- "Number of Records"
par(mfrow=c(2,4))
hist(train$car_age,xlab="Age of Car (Years)", main="Frequency by Age of Car",col=colplots)
barplot(table(train$shopping_pt),main="Frequency by Shopping Point",ylab=ylab.box,xlab="Shopping Point
Depth",col=colplots)
hist(train$duration_previous,xlab="Duration (Years)", main="Previous Insurance Issuer
Duration",col=colplots)
barplot(table(train$day),main="Frequency of Site Visits by Day",ylab=ylab.box,xlab="Day
(0=Monday,...,6=Sunday)",col=colplots)
barplot(table(train$car_value),main="Frequency by Car Values (New)",ylab=ylab.box,xlab="Car Value
Categories",col=colplots)
hist(train$age_oldest,xlab="Age of Oldest Person in Group", main="Frequency by Oldest Age",col=colplots)
hist(train$age_youngest,xlab="Age of Youngest Person in Group", main="Frequency by Youngest
Age",col=colplots)


#barplots for response variables
xlab.policy = "Policy Options"; par(mfrow=c(1,4))


barplot(table(train$A),main="Frequency for Option A",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
barplot(table(train$B),main="Frequency for Option B",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
barplot(table(train$C),main="Frequency for Option C",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
barplot(table(train$D),main="Frequency for Option D",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
par(mfrow=c(1,3))
barplot(table(train$E),main="Frequency for Option E",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
barplot(table(train$F),main="Frequency for Option F",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
barplot(table(train$G),main="Frequency for Option G",ylab=ylab.box,xlab=xlab.policy,col=col.bar)
```

```
#look at 5 number summaries
lapply(train, summary)


#consider outliers in select variables
col.box = "dodgerblue4"
bxp.shopping_pt <- boxplot(train$shopping_pt,main="Box Plot of Shopping Points", ylab="Shopping Point",
col=col.box)
bxp.car_age <- boxplot(train$car_age,main="Box Plot of Car Age", ylab="Car Age (Years)", col=col.box)
bxp.cost <- boxplot(train$cost,main="Box Plot of Policy Option Cost", ylab="Cost (Dollars)", col=col.box)
# Reset plot display
par(mfrow=c(1,1))


#####End Data Description - DT 1/22/17



# summary statistics
summary(train)
# risk_factor 240418 NAs
# C_previous & duration_previous - 18711 NAs


# of unique Customers in train set - All purchase a plan
length(unique(train$customer_ID))
#97009


# of unique plans shopped in train set
length(unique(train$planCombo))
#1809


# of unique plans purchased in train set
length(unique(train.purchase$planCombo))
#1522


#top 10 plans purchased.
top10plans<-data.frame("count"=(sort(table(train.purchase$planCombo),decreasing=TRUE)[1:10]))
top10plans


# distribution of shopping point for purchased plans
hist(as.numeric((train.purchase$shopping_pt)),col="dodgerblue4",breaks=20,xlab = "Purchase Shopping
Point",
     main = "Purchase Shopping Point - Training Set")
maxShoppingPoint.test<-aggregate(test$shopping_pt, by = list(test$customer_ID), max)
#head(maxShoppingPoint.test)
hist(as.numeric((maxShoppingPoint.test$x)),col="dodgerblue4",breaks=20,xlab = "Max Shopping Point",
     main = "Max Shopping Point - Test Set")



#Exploring the components of a plan against variables
mysettings <- trellis.par.get()
mysettings$strip.background$col <- c("lightgrey", "darkgrey")
trellis.par.set(mysettings)
histogram(~ C | car_value, data = train.purchase,
          col = "dodgerblue4", main = "Insurance Plan C v. Car Value", xlab = "Insurance Plan C")
histogram(~ A+B+C+D+E+F+G | car_value, data = train.purchase,
          col = "dodgerblue4", main = "Insurance Plan Options v. Car Value", xlab = "Insurance Options A,
B, C, D, E, F, and G")
```

```
# histogram(~ A+B+C +D+E+F+G| homeowner, data = train.purchase)


#Frequency of policy Option by state
A_Freq<-prop.table(table(train.purchase$state,train.purchase$A),1)
B_Freq<-prop.table(table(train.purchase$state,train.purchase$B),1)
C_Freq<-prop.table(table(train.purchase$state,train.purchase$C),1)
D_Freq<-prop.table(table(train.purchase$state,train.purchase$D),1)
E_Freq<-prop.table(table(train.purchase$state,train.purchase$E),1)
F_Freq<-prop.table(table(train.purchase$state,train.purchase$F),1)
G_Freq<-prop.table(table(train.purchase$state,train.purchase$G),1)
#Stack bar graph for interesting relationships
plot(A_Freq,col=c("blue","red","yellow","green"))


plot(F_Freq,col=c("blue","red","yellow","green"))
plot(G_Freq,col=c("blue","red","yellow","green"))



#can't get the below function to work, trying to plot hist for each variable for each purchase option
#my_hist2<-function(variable)
#{
  #x <- get(variable)
#  ggplot(train.purchase,aes(x=day))+geom_bar()+facet_grid(~variable)
  #h<-hist(x,breaks=seq(from=-.5,to=4.5,by=1),col="red",main=variable)
#}
#apply(X = array(names(train.purchase)[18:24]),MARGIN =1,FUN = my_hist2)


#find numeric columns #SC
nums <- sapply(train.purchase, is.numeric)
train.purchase[ , nums]
str(train.purchase)
ggplot(train.purchase,aes(x=day)) + theme_bw() + facet_grid(~A) + geom_bar(color =I("black"), fill =
I("dodgerblue4")) + ggtitle("Insurance Option A") + theme(plot.title = element_text(hjust = 0.5))


#graph of predictor vs response
ggplot(train.purchase,aes(x=age_oldest))+geom_bar()+facet_grid(~A)
ggplot(train.purchase,aes(x=train.purchase[,paste("day")]))+geom_bar()+facet_grid(paste("~","A"))


forLoopGraph <- function(x) {
  for (i in 1:7) {
    t =
ggplot(train.purchase,aes(x=train.purchase[,paste(x)]))+geom_bar()+facet_grid(paste("~",names(train.purcha
se)[17+1]))
  }
  return(t)
}


forLoopGraph("car_value")
dfgraph = apply(X=array(names(train.purchase)[c(4,8:17)]), MARGIN = 1, FUN = forLoopGraph)
dfgraph[2]
#histtable of each predictor for each response #SC


pctTot <- function(x) {
  length(x) / nrow(train.purchase) * 100
}


forLoopFunc <- function(x) {
  print(x)
  histtableEDAtemp = list()
  for (i in 1:7) {
```

```
    #print(myFunction2(train.purchase, names(train.purchase)[17+i], x))
    df = melt(cast(train.purchase, paste(names(train.purchase)[17+i],x, sep = "~"), pctTot))
    df$col1 = names(df)[1]
    df$col2 = names(df)[3]
    names(df)[1] = "cat1"
    names(df)[3] = "cat2"
    histtableEDAtemp[[i]] = df
  }
  histtableEDA = do.call(rbind, histtableEDAtemp)
  return(histtableEDA)
}


#EDA for continous variables (average by coverage option) #SC
ddply(train.purchase, .(A), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(B), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(C), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(D), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(E), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(F), summarize, avg=mean(age_oldest))
ddply(train.purchase, .(G), summarize, avg=mean(age_oldest))


ddply(train.purchase, .(A), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(B), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(C), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(D), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(E), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(F), summarize, avg=mean(age_youngest))
ddply(train.purchase, .(G), summarize, avg=mean(age_youngest))


ddply(train.purchase, .(A), summarize, avg=mean(car_age))
ddply(train.purchase, .(B), summarize, avg=mean(car_age))
ddply(train.purchase, .(C), summarize, avg=mean(car_age))
ddply(train.purchase, .(D), summarize, avg=mean(car_age))
ddply(train.purchase, .(E), summarize, avg=mean(car_age))
ddply(train.purchase, .(F), summarize, avg=mean(car_age))
ddply(train.purchase, .(G), summarize, avg=mean(car_age))


dcast(train.purchase, age_oldest~A, mean)
histtable = apply(X=array(names(train.purchase)[c(4,8:17)]), MARGIN = 1, FUN = forLoopFunc)
write.csv(rbind.fill(newdf[1]), "c:/histtbl1.csv")
write.csv(rbind.fill(newdf[2]), "c:/histtbl2.csv")
write.csv(rbind.fill(newdf[3]), "c:/histtbl3.csv")
write.csv(rbind.fill(newdf[4]), "c:/histtbl4.csv")
write.csv(rbind.fill(newdf[5]), "c:/histtbl5.csv")
write.csv(rbind.fill(newdf[6]), "c:/histtbl6.csv")
write.csv(rbind.fill(newdf[7]), "c:/histtbl7.csv")
write.csv(rbind.fill(newdf[8]), "c:/histtbl8.csv")
write.csv(rbind.fill(newdf[9]), "c:/histtbl9.csv")
write.csv(rbind.fill(newdf[10]), "c:/histtbl10.csv")
write.csv(rbind.fill(newdf[11]), "c:/histtbl11.csv")


##uniquechar
train.uniquechar = unique(train[c("customer_ID","state",
"group_size","homeowner","car_age","car_value","risk_factor","age_oldest",
                                  "age_youngest","married_couple","C_previous","duration_previous")])


#add numeric factors in place of categorical for correlation analysis #SC
#correlation matrix for numeric variables #SC -- AB: had to modify since we changed some from integer to
factors
#code works as of 2/4/17 #SC
train_cp = train
```

```
state_factor = as.factor(train[,c("state")])
state_ranks <- rank(-table(state_factor), ties.method="first")
train_cp$state_num <- data.frame(category=state_factor, rank=state_ranks[as.character(state_factor)])$rank


car_value_factor = as.factor(train[,c("car_value")])
car_value_ranks <- rank(-table(car_value_factor), ties.method="first")
train_cp$car_value_num <- data.frame(category=car_value_factor,
rank=car_value_ranks[as.character(car_value_factor)])$rank


sapply(train_cp, class)
# 2, 8, 10, 12:14, 17, 25, 27, 28
#cormat = cor(train_cp[c(2:4,7:10,12:17,27:28)], use="na.or.complete")
cormat = cor(train_cp[c(2, 8, 10, 12:14, 17, 25, 27, 28)], use="na.or.complete") # AB: This doesn't work
either yet.
cormat_table <- as.data.frame(as.table(cormat))
cormat_table <- cormat_table[order(abs(cormat_table$Freq),decreasing = TRUE),]
write.csv(cormat_table, "cormat_table.csv")


#PCA #SC -- AB: This no longer works since 'x' must be numeric
#train.pca <- prcomp(na.omit(train_cp[c(2:4,7:10,12:17,27:28)]),center = TRUE,scale. = TRUE)
#print(train.pca)
#summary(train.pca)
#plot(train.pca, type="l")


#write.csv(train.pca$rotation,"pca.csv")


############ EDA Naive Models ############
# shopping_pt + day + time + state + location + group_size + homeowner + car_age + car_value + risk_factor
+ age_oldest + age_youngest + married_couple + C_previous + duration_previous + cost


# Create tree model
#tree.x <- rpart(A ~ shopping_pt + day + location + group_size + homeowner + car_value +
                  #risk_factor + age_oldest + age_youngest + married_couple + C_previous +
                  #duration_previous + cost, data = train, method = "anova", control=
rpart.control(maxdepth= 3))
#tree.x # splits on cost and location
#prp(tree.x)
#fancyRpartPlot(tree.x,sub = "") # unreadable


########## END EDA ############


# Create LDA model--
#model.lda <- lda(A ~ shopping_pt + day + homeowner, data = train)
#+ day + time + state + location + group_size + homeowner + car_age +
#  car_value + risk_factor + age_oldest + age_youngest + married_couple + C_previous +
#  duration_previous + cost


#plot(model.lda, main = "LDA Model", cex = 0.90)


# Use backward subset selection on model.lda--crashes RStudio
#model.lda.bwd <- regsubsets(A ~ shopping_pt + day + state + location + group_size + homeowner + car_age +
#                              car_value + risk_factor + age_oldest + age_youngest + married_couple +
C_previous +
#                              duration_previous + cost, data = train, nvmax=5, method="backward")
#summary(model.lda.bwd)


# Create second LDA model using top selected variables
#model.lda2 <- lda(A ~ , data = train)
```

```
#plot(model.lda2)


####################################
## Data manipulation for Model Build ##
####################################
#PB


train.purchase.m<-train.purchase


#Adding previous quoted plans to train.purchase.m data frame #PB
train.purchase.m$purchaseMinus_1<-train.purchase.m$shopping_pt-1
train.purchase.m$purchaseMinus_2<-train.purchase.m$shopping_pt-2
train.purchase.m$purchaseMinus_3<-train.purchase.m$shopping_pt-3
train.purchase.m$purchaseMinus_4<-train.purchase.m$shopping_pt-4


#purchase minus 1 quote
lookup<-train[c("customer_ID","shopping_pt","planCombo")]
train.purchase.m<-
merge(x=train.purchase.m,y=lookup,by.x=c("purchaseMinus_1","customer_ID"),by.y=c("shopping_pt","customer_I
D"))
#cleaning up train.purchase.m dataframe
names(train.purchase.m)[names(train.purchase.m)=='planCombo.x']<-"planCombo"
names(train.purchase.m)[names(train.purchase.m)=='planCombo.y']<-"lastQuotedPlan"


#purchase minus 2 quote
train.purchase.m<-
merge(x=train.purchase.m,y=lookup,by.x=c("purchaseMinus_2","customer_ID"),by.y=c("shopping_pt","customer_I
D"),all.x=TRUE)
#cleaning up train.purchase.m dataframe
names(train.purchase.m)[names(train.purchase.m)=='planCombo.x']<-"planCombo"
names(train.purchase.m)[names(train.purchase.m)=='planCombo.y']<-"QuoteMinus_2"


#purchase minus 3 quote
train.purchase.m<-
merge(x=train.purchase.m,y=lookup,by.x=c("purchaseMinus_3","customer_ID"),by.y=c("shopping_pt","customer_I
D"),all.x=TRUE)
#cleaning up train.purchase.m dataframe
names(train.purchase.m)[names(train.purchase.m)=='planCombo.x']<-"planCombo"
names(train.purchase.m)[names(train.purchase.m)=='planCombo.y']<-"QuoteMinus_3"


#purchase minus 4 quote
train.purchase.m<-
merge(x=train.purchase.m,y=lookup,by.x=c("purchaseMinus_4","customer_ID"),by.y=c("shopping_pt","customer_I
D"),all.x=TRUE)
#cleaning up train.purchase.m dataframe
names(train.purchase.m)[names(train.purchase.m)=='planCombo.x']<-"planCombo"
names(train.purchase.m)[names(train.purchase.m)=='planCombo.y']<-"QuoteMinus_4"


#Cleaning up the train.purchsase data frame
train.purchase.m$QuoteMinus_3[is.na(train.purchase.m$QuoteMinus_3)]<-"XXXXXXX"
train.purchase.m$QuoteMinus_4[is.na(train.purchase.m$QuoteMinus_4)]<-"XXXXXXX"
train.purchase.m<-train.purchase.m[order(train.purchase.m$customer_ID),]
#removing unneccessary variables
train.purchase.m$purchaseMinus_1 <- NULL
train.purchase.m$purchaseMinus_2 <- NULL
train.purchase.m$purchaseMinus_3 <- NULL
train.purchase.m$purchaseMinus_4 <- NULL


#table of who bought the last plan shopped
table(train.purchase.m$planCombo==train.purchase.m$lastQuotedPlan)
```

```
#adding quoting history to model data frame #PB
planOptions<-c("A","B","C","D","E","F","G")
for (ii in 1:7)  {
  train.purchase.m[paste("lastQuoted_",planOptions[ii],sep="")] <-
as.factor(substring(train.purchase.m$lastQuotedPlan,first=ii,last=ii))
  train.purchase.m[paste("Quoted_",planOptions[ii],"_minus2",sep="")] <-
as.factor(substring(train.purchase.m$QuoteMinus_2,first=ii,last=ii))
  train.purchase.m[paste("Quoted_",planOptions[ii],"_minus3",sep="")] <-
as.factor(substring(train.purchase.m$QuoteMinus_3,first=ii,last=ii))
  train.purchase.m[paste("Quoted_",planOptions[ii],"_minus4",sep="")] <-
as.factor(substring(train.purchase.m$QuoteMinus_4,first=ii,last=ii))
}



#looking at how often options change from purchase to last quote #PB
quoteChange_df<-data.frame(matrix(0,nrow=dim(train.purchase)[1],ncol=1))
planOptions<-c("A","B","C","D","E","F","G")
for (ii in 1:7)  {
  quoteChange_df[paste(planOptions[ii],"_change",sep="")] <-
as.numeric((get(paste("lastQuoted_",planOptions[ii],sep=""),train.purchase.m))!=(get(planOptions[ii],train
.purchase.m)))
}
quoteChange_df[1]<-NULL
colSums(quoteChange_df,2)
# A_change B_change C_change D_change E_change F_change G_change
# 6894     6392     6716     4927     6067     7039    13174


# study the changes G records
changeG<-(subset(train.purchase.m,train.purchase.m$lastQuoted_G!=train.purchase.m$G))




quoteChange_df<-data.frame(matrix(0,nrow=dim(train.purchase.m[validSubset,])[1],ncol=1))
planOptions<-c("A","B","C","D","E","F","G")
for (ii in 1:7)  {
  quoteChange_df[paste(planOptions[ii],"_change",sep="")] <-
as.numeric((get(paste("lastQuoted_",planOptions[ii],sep=""),train.purchase.m[validSubset,]))!=(get(planOpt
ions[ii],train.purchase.m[validSubset,])))
}
quoteChange_df[1]<-NULL
colSums(quoteChange_df,2)
colSums(quoteChange_df,2)/dim(train.purchase.m[validSubset,])[1]


1-mean(train.purchase.m$lastQuoted_G[validSubset]==train.purchase.m$G[validSubset])
#0.1374732



#####################################
## Train/Validation Split ##
#####################################


# 75/25 train/validation split            #PB
n <-dim(train.purchase.m)[1] # sample size = 97009 customers purchase a policy
set.seed(1233) # set random number generator seed to enable
# repeatability of results
valid <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m$part<-0
```

```
train.purchase.m$part[-valid] <-"train"
train.purchase.m$part[valid]  <-"valid"
table(train.purchase.m$part)
trainSubset<-(which(train.purchase.m$part=="train"))
validSubset<-(which(train.purchase.m$part=="valid"))


# add variable "part" to the full training data set
lookup<-train.purchase.m[c("customer_ID","part")]
train<-merge(x=train,y=lookup,by="customer_ID")



####################################
## Impute missing values ##
####################################
#we could use a decision tree to impute missing values. I am using the median to get the models working.
Please feel free to change #PB
# Ahmar: I have added code in the Data Quality section to identify relationship using tree and also
created new variables
# with name like _imp for 3 variables. If we want to stick with original names of variables then following
code will work.
#train.purchase.m$risk_factor[is.na(train.purchase.m$risk_factor)]<-
median(train.purchase.m$risk_factor[!is.na(((train.purchase.m$risk_factor)))])
#train.purchase.m$C_previous[is.na(train.purchase.m$C_previous)]<-
as.factor((which.max(table(train.purchase.m$C_previous))))
#train.purchase.m$duration_previous[is.na(train.purchase.m$duration_previous)]<-
median(train.purchase.m$duration_previous[!is.na(((train.purchase.m$duration_previous)))])


#train.purchase.m$risk_factor <- train.purchase.m$risk_factor_imp
#train.purchase.m$C_previous <- train.purchase.m$C_previous_imp
#train.purchase.m$duration_previous <- train.purchase.m$duration_previous_imp


summary(train.purchase.m)


####################################
## Model Build ##
####################################


#############################################################################
## Option *G* Models ##
#############################################################################



####################
#LDA G
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda0.g <- lda(G ~ (lastQuoted_G) + risk_factor + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4,
                        data = train.purchase.m,
                        subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#   user  system elapsed
#   3.17    0.24    3.42
```

```
#classification accuracy for training data
post.train.lda0.g <- predict(object=model.lda0.g, newdata = train.purchase.m[trainSubset,])
plot(model.lda0.g, col = as.integer(train.purchase.m$G[-validSubset]), dimen = 2) #scatterplot with colors
table(post.train.lda0.g$class, train.purchase.m$G[trainSubset]) #confusion matrix
mean(post.train.lda0.g$class==train.purchase.m$G[trainSubset]) #what percent did we predict successfully?
plot(train.purchase.m$G[trainSubset], post.train.lda0.g$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda0.g <- predict(object=model.lda0.g, newdata = train.purchase.m[validSubset,])
plot(model.lda0.g, col = as.integer(train.purchase.m$G[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda0.g$class, train.purchase.m$G[validSubset]) #confusion matrix
mean(post.valid.lda0.g$class==train.purchase.m$G[validSubset]) #what percent did we predict successfully?
plot(train.purchase.m$G[validSubset], post.valid.lda0.g$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?


###Other less good models mentioned in paper
#Reduced model - only lastQuoted_G
model.lda1.g <- lda(G ~ (lastQuoted_G),
                    data = train.purchase.m,
                    subset = trainSubset)
#Reduced model - everything but lastQuoted_G
model.lda2.g <- lda(G ~ (risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day +
shopping_pt + state +
    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4),
                    data = train.purchase.m,
                    subset = trainSubset)
#Reduced model - everything but lastQuoted_G and Quoted_G_minus2
model.lda3.g <- lda(G ~ (risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day +
shopping_pt + state +
     Quoted_G_minus3 + Quoted_G_minus4),
                    data = train.purchase.m,
                    subset = trainSubset)



####################
# K-Nearest Neighbors -- 2/4/17 FP --- Edited AM 3/12/17
####################
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


### KNN SAMPLING CODE (need 'train' and 'valid' in part column)   #FP
n <-dim(train.purchase.m)[1]
knn.sample <- sample(n, round(.25*n))
train.purchase.m.knn<-train.purchase.m[knn.sample,]
dim(train.purchase.m.knn)
View(train.purchase.m.knn)


train.purchase.m.knn$Quoted_G_minus3 = as.numeric(train.purchase.m.knn$Quoted_G_minus3)
train.purchase.m.knn$Quoted_G_minus2 = as.numeric(train.purchase.m.knn$Quoted_G_minus2)
train.purchase.m.knn$Quoted_G_minus4 = as.numeric(train.purchase.m.knn$Quoted_G_minus4)
train.purchase.m.knn$lastQuoted_G = as.numeric(train.purchase.m.knn$lastQuoted_G)
train.purchase.m.knn$C_previous_imp = as.numeric(train.purchase.m.knn$C_previous_imp)
train.purchase.m.knn$married_couple = as.numeric(train.purchase.m.knn$married_couple)
train.purchase.m.knn$car_value = as.numeric(train.purchase.m.knn$car_value)
train.purchase.m.knn$homeowner = as.numeric(train.purchase.m.knn$homeowner)
train.purchase.m.knn$state = as.numeric(train.purchase.m.knn$state)
```

```
train.purchase.m.knn$day = as.numeric(train.purchase.m.knn$day)


set.seed(1)
library(class)
dim(train.purchase.m.knn)
table(train.purchase.m.knn$part)
### 24,252 observations | 18,221 train | 6,031 valid


ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(G ~ (lastQuoted_G) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                  Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4 + group_size + homeowner +
married_couple
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)


knnFit


# k   Accuracy   Kappa
# 5  0.8256630  0.7467639
# 7  0.8283022  0.7504890
# 9  0.8306936  0.7538245
# 11  0.8296217  0.7520257
# 13  0.8296217  0.7519470
#
# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was k = 9.


knnPredict <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knnPredict


### Define KNN training and test sets
knn.training <- train.purchase.m.knn[train.purchase.m.knn$part=="train",
c(2,4,6,8,9,10,11,13,14,15,25,26,58,59,60,61)]
knn.test <- train.purchase.m.knn[train.purchase.m.knn$part=="valid",
c(2,4,6,8,9,10,11,13,14,15,25,26,58,59,60,61)]
View(knn.training)
knn.trainLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="train", 24]
knn.testLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", 24]


colSums(is.na(knn.training))[colSums(is.na(knn.training)) > 0]
colSums(is.na(knn.test))[colSums(is.na(knn.test)) > 0]


table(knn.trainLabels)
table(knn.testLabels)


### Building classifier
knn.fit <- knn(train= knn.training, test= knn.test, cl = knn.trainLabels, k=3, prob=TRUE) #, l=0,
prob=TRUE, use.all = TRUE)


knn.fit


knn.valid <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knn.valid
```

```
plot(knn.fit)


library(gmodels)
CrossTable(x = knn.testLabels, y = knn.fit, prop.chisq=FALSE)


table(knnPredict, knn.testLabels)


#Check the misclassification rate
error.knn.C <- round(mean(knn.valid!=knn.testLabels),4)
error.knn.C
# 0.1560



confusionMatrix(knn.valid,knn.testLabels)
# Kappa 0.7729


####################
# RandomForest G
####################
set.seed(1)


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


model.rf.G <- randomForest(G ~ (lastQuoted_G) + risk_factor + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4 ,
  data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock


#RunTime
#user  system elapsed
#730.899   6.797 742.187


#model summary,Var importance stats and plot
model.rf.G
randomForest::importance(model.rf.G)
randomForest::varImpPlot(model.rf.G)


# Predict random forest on validation set
post.valid.rf.G <- predict(model.rf.G, train.purchase.m[validSubset,])
length(post.valid.rf.G)


#Create a simple confusion matrix
table(post.valid.rf.G,train.purchase.m$G[validSubset])


#Check the misclassification rate
error.rf.G <- round(mean(post.valid.rf.G!=train.purchase.m$G[validSubset]),4)
error.rf.G


#Compare against the misclassification rate for the base model
error.rf.G.base <-
round(mean(train.purchase.m$lastQuoted_G[validSubset]!=train.purchase.m$G[validSubset]),4)
error.rf.G.base
```

```
# Fit Metrics
confusionMatrix(post.valid.rf.G,train.purchase.m$G[validSubset],)
```

```
####################
# Boosting Model G
####################
set.seed(1)

#### Use this code to tune the GBM model. ###

# library(caret)
# myTuneGrid <- expand.grid(n.trees = 500,interaction.depth = c(6,7),shrinkage =
c(.001,.01,.1),n.minobsinnode=10)
# fitControl <- trainControl(method = "repeatedcv", number = 3,repeats = 1, verboseIter =
FALSE,returnResamp = "all")
# myModel <- train(G ~ (lastQuoted_G)+ risk_factor + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
#                    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4  ,
#                    data=train.purchase.m[trainSubset,],
#                      method = "gbm",
#                      trControl = fitControl,
#                      tuneGrid = myTuneGrid)
```

```
ptm <- proc.time() # Start the clock!
set.seed(1)
model.boost.G=gbm(G ~ (lastQuoted_G)+ risk_factor + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state +
    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4  ,
  data=train.purchase.m[trainSubset,],
  distribution="multinomial",
  n.trees=1000,
  interaction.depth=4,
  shrinkage = .01)

proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#283.950   4.094 302.237

#relative influence statistics & plot.
summary(model.boost.G)
summaryBoost<-summary(model.boost.G)

# Predict GBM on validation set
post.valid.boost.prob.G <- predict(model.boost.G,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.G<-apply(post.valid.boost.prob.G, 1, which.max)
length(post.valid.boost.G)
head(post.valid.boost.G)

#Create a simple confusion matrix
table(post.valid.boost.G,train.purchase.m$G[validSubset])
```

```
#Check the misclassification rate
error.boost.G <- round(mean(post.valid.boost.G!=train.purchase.m$G[validSubset]),4)
error.boost.G


#Compare against the misclassification rate for the base model
error.boost.G.base <-
round(mean(train.purchase.m$lastQuoted_G[validSubset]!=train.purchase.m$G[validSubset]),4)
error.boost.G.base


# Fit Metrics
confusionMatrix(post.valid.boost.G,train.purchase.m$G[validSubset],)


#plot relative influence of variables
summaryBoost<-summaryBoost[order(summaryBoost$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoost$rel.inf),names.arg = summaryBoost$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)



###################
# SVM G
###################
set.seed(1)


### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)






# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a radial kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 5,cross = 5)
# tune.out = tune(
#   svm,G ~ (lastQuoted_G) + risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day +
shopping_pt + state +
#     Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4  ,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(1)),
#   tunecontrol = control
# )
# summary(tune.out)



#Fit a linear SVM Model
ptm <- proc.time() # Start the clock!
svmfit.G=svm(G ~ (lastQuoted_G) + risk_factor + car_age + car_value + cost + age_oldest + age_youngest +
day + shopping_pt + state +
    Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4  ,
  data=train.purchase.m.svm,
  kernel="linear",
```

```
  gamma=.01,
  cost=1,
  probability =TRUE)
proc.time() - ptm # Stop the clock


#Summary statitics
summary(svmfit.G)


#RunTime
# user  system elapsed
# 156.099   1.407 158.419


# Predict SVM on validation set
post.valid.svm.G<-predict(svmfit.G,train.purchase.m[validSubset,])
length(post.valid.svm.G)


#Create a simple confusion matrix
table(post.valid.svm.G,train.purchase.m$G[validSubset])


#Check the misclassification rate
error.svm.G <- round(mean(post.valid.svm.G!=train.purchase.m$G[validSubset]),4)
error.svm.G
# [1] 0.1363


#Compare against the misclassification rate for the base model
error.svm.G.base <-
round(mean(train.purchase.m$lastQuoted_G[validSubset]!=train.purchase.m$G[validSubset]),4)
error.svm.G.base


##############################################################################
## Option *A* Models ##
##############################################################################


# Create naive RF model to select predictors for rest of models
ptm <- proc.time() # Start the clock!
set.seed(1)
model.RF.naive.A <- randomForest(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost +
age_oldest + age_youngest + day + shopping_pt + state +
                                        Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + homeowner
+ married_couple + group_size + C_previous_imp + duration_previous_imp,
                                 data = train.purchase.m, ntree =500)
proc.time() - ptm # Stop the clock
#   user  system elapsed
#748.162  18.492 805.550


varImpPlot(model.RF.naive.A, main = "Random Forest Model: \n Variable Importance")
importance(model.RF.naive.A)
#homeowner + married_couple + group_size don't appear as important--will leave out for other models


# Base Model
ptm <- proc.time() # Start the clock!
model.A.base <- lda(A ~ (lastQuoted_A),
                    data = train.purchase.m,
                    subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#   user  system elapsed
#0.119   0.021   0.142
```

```
error.A.base.train <-
round(mean(train.purchase.m$lastQuoted_A[trainSubset]!=train.purchase.m$A[trainSubset]),4)
error.A.base.train
#0.0705


error.A.base <- round(mean(train.purchase.m$lastQuoted_A[validSubset]!=train.purchase.m$A[validSubset]),4)
error.A.base
#0.0729


post.valid.base.A <- predict(object=model.A.base, newdata = train.purchase.m[validSubset,])
post.valid.base.A


confusionMatrix(post.valid.base.A$class,train.purchase.m$A[validSubset],)


####################
#LDA *A*
####################
set.seed(1)


##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda0.a <- lda(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                      Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + C_previous_imp +
duration_previous_imp,
                    data = train.purchase.m,
                    subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#   user  system elapsed
#   2.768   0.422   3.437


#classification accuracy for training data
post.train.lda0.a <- predict(object=model.lda0.a, newdata = train.purchase.m[trainSubset,])
plot(model.lda0.a, col = as.integer(train.purchase.m$A[-validSubset]), dimen = 2) #scatterplot with colors
table(post.train.lda0.a$class, train.purchase.m$A[trainSubset]) #confusion matrix
#0     1     2
#0 14633   777   276
#1  1116 43070  1828
#2   286   844  9927


(mean(post.train.lda0.a$class!=train.purchase.m$A[trainSubset]))
#0.07046745
plot(train.purchase.m$A[trainSubset], post.train.lda0.a$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda0.a <- predict(object=model.lda0.a, newdata = train.purchase.m[validSubset,])
plot(model.lda0.a, col = as.integer(train.purchase.m$A[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda0.a$class, train.purchase.m$A[validSubset]) #confusion matrix
#0     1     2
#0  4922   254    94
#1   325 14349   710
#2    96   288  3214


#Check the misclassification rate
error.lda.A <- round(mean(post.valid.lda0.a$class!=train.purchase.m$A[validSubset]),4)
```

```
error.lda.A
#0.0729


#Compare against the misclassification rate for the base model
error.lda.A.base <-
round(mean(train.purchase.m$lastQuoted_A[validSubset]!=train.purchase.m$A[validSubset]),4)
error.lda.A.base
# 0.0729



plot(train.purchase.m$A[validSubset], post.valid.lda0.a$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?


confusionMatrix(post.valid.lda0.a$class,train.purchase.m$A[validSubset],)
#Confusion Matrix and Statistics
#Reference
#Prediction     0     1     2
#0  4922   254    94
#1   325 14349   710
#2    96   288  3214


#Overall Statistics


#Accuracy : 0.9271
#95% CI : (0.9238, 0.9304)
#No Information Rate : 0.614
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.8646
#Mcnemar's Test P-Value : < 2.2e-16


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9212   0.9636   0.7999
#Specificity            0.9816   0.8894   0.9810
#Pos Pred Value         0.9340   0.9327   0.8933
#Neg Pred Value         0.9778   0.9389   0.9611
#Prevalence             0.2203   0.6140   0.1657
#Detection Rate         0.2030   0.5917   0.1325
#Detection Prevalence   0.2173   0.6343   0.1484
#Balanced Accuracy      0.9514   0.9265   0.8905



####################
# K-Nearest Neighbors *A*
####################
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
set.seed(1)
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


### KNN SAMPLING CODE (need 'train' and 'valid' in part column)   #FP
set.seed(1)
```

```
n <-dim(train.purchase.m)[1]
knn.sample <- sample(n, round(.25*n))
train.purchase.m.knn<-train.purchase.m[knn.sample,]
dim(train.purchase.m.knn)
#View(train.purchase.m.knn)


train.purchase.m.knn$Quoted_A_minus3 = as.numeric(train.purchase.m.knn$Quoted_A_minus3)
train.purchase.m.knn$Quoted_A_minus2 = as.numeric(train.purchase.m.knn$Quoted_A_minus2)
train.purchase.m.knn$Quoted_A_minus4 = as.numeric(train.purchase.m.knn$Quoted_A_minus4)
train.purchase.m.knn$lastQuoted_C = as.numeric(train.purchase.m.knn$lastQuoted_A)
train.purchase.m.knn$C_previous_imp = as.numeric(train.purchase.m.knn$C_previous_imp)
train.purchase.m.knn$married_couple = as.numeric(train.purchase.m.knn$married_couple)
train.purchase.m.knn$car_value = as.numeric(train.purchase.m.knn$car_value)
train.purchase.m.knn$homeowner = as.numeric(train.purchase.m.knn$homeowner)
train.purchase.m.knn$state = as.numeric(train.purchase.m.knn$state)
train.purchase.m.knn$day = as.numeric(train.purchase.m.knn$day)


set.seed(1)
library(class)
dim(train.purchase.m.knn)
table(train.purchase.m.knn$part)
### 24,252 observations | 18,260 train | 5,992 valid


ptm <- proc.time() # Start the clock!
ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                  Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + C_previous_imp +
duration_previous_imp
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)
proc.time() - ptm # Stop the clock


knnFit
#k-Nearest Neighbors


#24252 samples
#15 predictor
#3 classes: '0', '1', '2'


#Pre-processing: centered (16), scaled (16)
#Resampling: Cross-Validated (10 fold, repeated 1 times)
#Summary of sample sizes: 21825, 21827, 21827, 21825, 21827, 21828, ...
#Resampling results across tuning parameters:

#  k   Accuracy   Kappa
#5  0.9255318  0.8608074
#7  0.9270986  0.8636649
#9  0.9278821  0.8651100
#11  0.9280882  0.8655147
#13  0.9278822  0.8651222


#Accuracy was used to select the optimal model using  the largest value.
#The final value used for the model was k = 11.


knnPredict <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knnPredict


### Define KNN training and test sets
```

```
knn.training <- train.purchase.m.knn[train.purchase.m.knn$part=="train",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,34,35,36,37)]
knn.test <- train.purchase.m.knn[train.purchase.m.knn$part=="valid",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,34,35,36,37)]
#View(knn.training)
knn.trainLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="train", 18]
knn.testLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", 18]


colSums(is.na(knn.training))[colSums(is.na(knn.training)) > 0]
colSums(is.na(knn.test))[colSums(is.na(knn.test)) > 0]


table(knn.trainLabels)
table(knn.testLabels)


### Building classifier
knn.fit <- knn(train= knn.training, test= knn.test, cl = knn.trainLabels, k=11, prob=TRUE) #, l=0,
prob=TRUE, use.all = TRUE)


knn.fit


knn.valid <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knn.valid


knn.train <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="train",])
knn.train


plot(knn.fit)


library(gmodels)
CrossTable(x = knn.testLabels, y = knn.fit, prop.chisq=FALSE)


table(knnPredict, knn.testLabels)
#knn.testLabels
#knnPredict    0    1    2
#0 1175   56   21
#1   79 3592  192
#2   24   68  785


#Check the misclassification rate
error.knn.train.A <- round(mean(knn.train!=knn.trainLabels),4)
error.knn.train.A
# 0.0698


#Check the misclassification rate
error.knn.A <- round(mean(knn.valid!=knn.testLabels),4)
error.knn.A
# 0.0741


confusionMatrix(knn.valid,knn.testLabels)
#Confusion Matrix and Statistics


#Reference
#Prediction    0    1    2
#0 1171   56   20
#1   83 3592  193
#2   24   68  785
```

```
#Overall Statistics


#Accuracy : 0.9259
#95% CI : (0.919, 0.9324)
#No Information Rate : 0.6202
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.8604
#Mcnemar's Test P-Value : 3.971e-14


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9163   0.9666   0.7866
#Specificity            0.9839   0.8787   0.9816
#Pos Pred Value         0.9391   0.9286   0.8951
#Neg Pred Value         0.9774   0.9416   0.9584
#Prevalence             0.2133   0.6202   0.1666
#Detection Rate         0.1954   0.5995   0.1310
#Detection Prevalence   0.2081   0.6455   0.1464
#Balanced Accuracy      0.9501   0.9227   0.8841


####################
# RandomForest *A*
####################
set.seed(1)
ptm <- proc.time() # Start the clock!
model.rf.A <- randomForest(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
                             Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + C_previous_imp +
duration_previous_imp,
                           data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock
#   user  system elapsed
#468.374  10.999 495.126


#model summary,Var importance stats and plot
model.rf.A
randomForest::importance(model.rf.A)
randomForest::varImpPlot(model.rf.A)


post.train.rf.A <- predict(model.rf.A, train.purchase.m[trainSubset,])


# Predict random forest on validation set
post.valid.rf.A <- predict(model.rf.A, train.purchase.m[validSubset,])
length(post.valid.rf.A)


#Create a simple confusion matrix
table(post.valid.rf.A,train.purchase.m$A[validSubset])
#post.valid.rf.A     0     1     2
#              0  4974   241    81
#              1   277 14360   728
#              2    92   290  3209


#Check the misclassification rate
```

```
error.train.rf.A <- round(mean(post.train.rf.A!=train.purchase.m$A[trainSubset]),4)
error.train.rf.A
#0.0531


#Check the misclassification rate
error.rf.A <- round(mean(post.valid.rf.A!=train.purchase.m$A[validSubset]),4)
error.rf.A
#0.0705


#Compare against the misclassification rate for the base model
error.rf.A.base <-
round(mean(train.purchase.m$lastQuoted_A[validSubset]!=train.purchase.m$A[validSubset]),4)
error.rf.A.base
#0.0729


#Compare against the misclassification rate for the base model
error.A.base <- round(mean(train.purchase.m$lastQuoted_A[trainSubset]!=train.purchase.m$A[trainSubset]),4)
error.A.base
#0.0705


# Fit Metrics
confusionMatrix(post.valid.rf.A,train.purchase.m$A[validSubset],)
#Confusion Matrix and Statistics
#Reference
#Prediction     0     1     2
#0  4974   241    81
#1   277 14360   728
#2    92   290  3209


#Overall Statistics
#Accuracy : 0.9295
#95% CI : (0.9262, 0.9327)
#No Information Rate : 0.614
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.8691
#Mcnemar's Test P-Value : < 2.2e-16


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9309   0.9643   0.7987
#Specificity            0.9830   0.8926   0.9811
#Pos Pred Value         0.9392   0.9346   0.8936
#Neg Pred Value         0.9805   0.9402   0.9608
#Prevalence             0.2203   0.6140   0.1657
#Detection Rate         0.2051   0.5921   0.1323
#Detection Prevalence   0.2184   0.6336   0.1481
#Balanced Accuracy      0.9570   0.9285   0.8899



####################
# Boosting Model *A*
####################


ptm <- proc.time() # Start the clock!
set.seed(1)
```

```
model.boost.A=gbm(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                    Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + C_previous_imp +
duration_previous_imp,
                  data=train.purchase.m[trainSubset,],
                  distribution="multinomial",
                  n.trees=1000,
                  interaction.depth=4,
                  shrinkage = .01)


proc.time() - ptm # Stop the clock
#user  system elapsed
#221.621   4.041 245.252


#relative influence statistics & plot.
summary(model.boost.A)
#var      rel.inf
#lastQuoted_A                   lastQuoted_A 92.309232045
#Quoted_A_minus2             Quoted_A_minus2  3.031761469
#car_age                           car_age  1.436593186
#cost                                 cost  1.109095993
#Quoted_A_minus3             Quoted_A_minus3  0.808908619
#state                               state  0.628317815
#Quoted_A_minus4             Quoted_A_minus4  0.374617269
#age_youngest                   age_youngest  0.128697563
#risk_factor_imp             risk_factor_imp  0.060140494
#age_oldest                       age_oldest  0.044539081
#duration_previous_imp duration_previous_imp  0.036031368
#C_previous_imp               C_previous_imp  0.017624987
#shopping_pt                     shopping_pt  0.007892330
#car_value                         car_value  0.004329008
#day                                     day  0.002218774
#summaryBoostA<-summary(model.boost.A)



post.train.boost.prob.A <- predict(model.boost.A,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.A<-apply(post.train.boost.prob.A, 1, which.max) - 1


train.boost.A <- round(mean(post.train.boost.A!=train.purchase.m$A[trainSubset]),4)
train.boost.A
#0.0647


#post.train.boost.A <- predict(model.boost.A, train.purchase.m[trainSubset,])
train.error.boost.A <- round(mean(post.train.boost.A!=train.purchase.m$A[trainSubset]),4)
train.error.boost.A


# Predict ABM on validation set
post.valid.boost.prob.A <- predict(model.boost.A,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.A<-apply(post.valid.boost.prob.A, 1, which.max)-1
length(post.valid.boost.A)
head(post.valid.boost.A)


#Create a simple confusion matrix
table(post.valid.boost.A,train.purchase.m$A[validSubset])
#post.valid.boost.A     0     1     2
#0  4996   228    78
#1   267 14373   728
#2    80   290  3212
```

```
#Compare against the misclassification rate for the base model
error.train.boost.A.base <-
round(mean(train.purchase.m$lastQuoted_A[trainSubset]!=train.purchase.m$A[trainSubset]),4)
error.train.boost.A.base
#0.0705


train.error.boost.A <- round(mean(post.train.boost.A!=train.purchase.m$A[trainSubset]),4)
train.error.boost.A
#0.965


#Check the misclassification rate
error.boost.A <- round(mean(post.valid.boost.A!=train.purchase.m$A[validSubset]),4)
error.boost.A
# 0.0689


#Compare against the misclassification rate for the base model
error.boost.A.base <-
round(mean(train.purchase.m$lastQuoted_A[validSubset]!=train.purchase.m$A[validSubset]),4)
error.boost.A.base
# 0.0729


# Fit Metrics
confusionMatrix(post.valid.boost.A,train.purchase.m$A[validSubset])
#Confusion Matrix and Statistics
#Reference
#Prediction     0     1     2
#0  4996   228    78
#1   267 14373   728
#2    80   290  3212


#Overall Statistics


#Accuracy : 0.9311
#95% CI : (0.9278, 0.9343)
#No Information Rate : 0.614
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.872
#Mcnemar's Test P-Value : < 2.2e-16


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9351   0.9652   0.7994
#Specificity            0.9838   0.8937   0.9817
#Pos Pred Value         0.9423   0.9353   0.8967
#Neg Pred Value         0.9817   0.9417   0.9610
#Prevalence             0.2203   0.6140   0.1657
#Detection Rate         0.2060   0.5927   0.1324
#Detection Prevalence   0.2186   0.6337   0.1477
#Balanced Accuracy      0.9594   0.9295   0.8906


#plot relative influence of variables
summaryBoostA<-summaryBoostA[order(summaryBoostA$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoostA$rel.inf),names.arg = summaryBoostA$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)
```

```
###################
# SVM *A*
###################
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)
#18189    62



# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a radial kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 5,cross = 5)
# tune.out = tune(
#   svm,A ~ (lastQuoted_A) + risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day +
shopping_pt + state +
#     Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4  ,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(1)),
#   tunecontrol = control
# )
# summary(tune.out)



#Fit a linear SVM Model
ptm <- proc.time() # Start the clock!
svmfit.A=svm(A ~ (lastQuoted_A) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state +
               Quoted_A_minus2 + Quoted_A_minus3 + Quoted_A_minus4 + C_previous_imp +
duration_previous_imp,
             data=train.purchase.m.svm,
             kernel="linear",
             gamma=.01,
             cost=1,
             probability =TRUE)
proc.time() - ptm # Stop the clock


#Summary statitics
summary(svmfit.A)
#Parameters:
#  SVM-Type:  C-classification
#SVM-Kernel:  linear
#cost:  1
#gamma:  0.01


#Number of Support Vectors:  3699
#( 1908 1054 737 )
#Number of Classes:  3
#Levels:
#  0 1 2


#RunTime
# user  system elapsed
```

```
# 49.066   0.533  52.545



#Train Error Rate
post.train.svm.A<-predict(svmfit.A,train.purchase.m[trainSubset,])
train.svm.A <- round(mean(post.train.svm.A!=train.purchase.m$A[trainSubset]),4)
train.svm.A
#0.0706


# Predict SVM on validation set
post.valid.svm.A<-predict(svmfit.A,train.purchase.m[validSubset,])
length(post.valid.svm.A) #24252


#Create a simple confusion matrix
table(post.valid.svm.A,train.purchase.m$A[validSubset])
#post.valid.svm.A     0     1     2
#               0  4910   224    77
#               1   360 14386   738
#               2    73   281  3203


#Check the misclassification rate
error.svm.A <- round(mean(post.valid.svm.A!=train.purchase.m$A[validSubset]),4)
error.svm.A
#0.0723


#Compare against the misclassification rate for the base model
error.svm.A.base <-
round(mean(train.purchase.m$lastQuoted_A[validSubset]!=train.purchase.m$A[validSubset]),4)
error.svm.A.base
#0.0729


# Fit Metrics
confusionMatrix(post.valid.svm.A,train.purchase.m$A[validSubset],)
#Confusion Matrix and Statistics


#Reference
#Prediction     0     1     2
#0  4910   224    77
#1   360 14386   738
#2    73   281  3203


#Overall Statistics


#Accuracy : 0.9277
#95% CI : (0.9244, 0.9309)
#No Information Rate : 0.614
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.8652
#Mcnemar's Test P-Value : < 2.2e-16


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9190   0.9661   0.7972
#Specificity            0.9841   0.8827   0.9825
```

```
#Pos Pred Value         0.9422   0.9291   0.9005
#Neg Pred Value         0.9773   0.9424   0.9606
#Prevalence             0.2203   0.6140   0.1657
#Detection Rate         0.2025   0.5932   0.1321
#Detection Prevalence   0.2149   0.6385   0.1467
#Balanced Accuracy      0.9515   0.9244   0.8898
```

```
################################################################################
## Option *B* Models ##
################################################################################
```

```
####################
#LDA *B*
####################
set.seed(1)
##Initial Full model
```

```
ptm <- proc.time() # Start the clock!
model.lda.B <- lda(B ~ (lastQuoted_B) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                     Quoted_B_minus2 + Quoted_B_minus3 + Quoted_B_minus4 + C_previous_imp +
duration_previous_imp  + group_size + homeowner + married_couple,
                   data = train.purchase.m,
                   subset = trainSubset)
proc.time() - ptm # Stop the clock
# #RunTime
#user  system elapsed
#2.04    0.18    2.23
```

```
#classification accuracy for training data
post.train.lda.B <- predict(object=model.lda.B, newdata = train.purchase.m[trainSubset,])
# plot(model.lda.D, col = as.integer(train.purchase.m$D[-validSubset]), dimen = 2) #scatterplot with
colors
table(post.train.lda.B$class, train.purchase.m$B[trainSubset]) #confusion matrix
mean(post.train.lda.B$class!=train.purchase.m$B[trainSubset]) #what percent did we predict successfully?
#0.06572563
#compare against misclassification rate for base model of training set
round(mean(train.purchase.m$lastQuoted_B[trainSubset]!=train.purchase.m$B[trainSubset]),4)
#0.0657
```

```
# plot(train.purchase.m$D[trainSubset], post.train.lda.D$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?
```

```
#classification accuracy for validation data
post.valid.lda.B <- predict(object=model.lda.B, newdata = train.purchase.m[validSubset,])$class
length(post.valid.lda.B)
# plot(model.lda.D, col = as.integer(train.purchase.m$D[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda.B, train.purchase.m$B[validSubset]) #confusion matrix
# plot(train.purchase.m$D[validSubset], post.valid.lda.D$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?
```

```
#Check the misclassification rate
error.lda.B <- round(mean(post.valid.lda.B!=train.purchase.m$B[validSubset]),4)
error.lda.B
# 0.0664
```

```
#Compare against the misclassification rate for the base model
```

```
error.lda.B.base <-
round(mean(train.purchase.m$lastQuoted_B[validSubset]!=train.purchase.m$B[validSubset]),4)
error.lda.B.base
# 0.0664


confusionMatrix(post.valid.lda.B,train.purchase.m$B[validSubset],)
# Kappa 0.8667


####################
# K-Nearest Neighbors *B*
####################
set.seed(1)


#knn second method
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(B ~ (lastQuoted_B) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                  Quoted_B_minus2 + Quoted_B_minus3 + Quoted_B_minus4 + C_previous_imp +
duration_previous_imp  + group_size + homeowner + married_couple
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)



# Resampling results across tuning parameters:
#
#   k   Accuracy   Kappa      Accuracy SD  Kappa SD
# 5  0.9207772  0.8450820  0.005632213  0.011662121
# 7  0.9225365  0.8484299  0.007032948  0.014358392
# 9  0.9216569  0.8463100  0.005811135  0.011981198
# 11  0.9205566  0.8437952  0.004690705  0.009812055
# 13  0.9205023  0.8434500  0.005354910  0.011164977
#
# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was k = 7.


knnPredict <- predict(knnFit,newdata = train.purchase.m[validSubset,])


knn.trainLabels <- train.purchase.m.knn[,c('B')]
knn.testLabels <- train.purchase.m[validSubset,c('B')]


summary(knn.testLabels)
### Building classifier
knn_pred <- knn(train = knn.training, test = knn.test, cl = knn.trainLabels, k=3)
knn_pred


library(gmodels)
CrossTable(x = knn.testLabels, y = knn_pred, prop.chisq=FALSE)


#train error rate
knnPredict_train <- predict(knnFit,newdata = train.purchase.m[trainSubset,])
```

```
train.knn.B <- round(mean(knnPredict_train!=train.purchase.m$B[trainSubset]),4)
train.knn.B


#validation error rate
error.knn.B <- round(mean(knnPredict!=train.purchase.m$B[validSubset]),4)
error.knn.B


confusionMatrix(knnPredict,train.purchase.m$B[validSubset],)


####################
# RandomForest *B*
####################
set.seed(1)


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


model.rf.B <- randomForest(B ~ (lastQuoted_B) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
                             Quoted_B_minus2 + Quoted_B_minus3 + Quoted_B_minus4 + C_previous_imp +
duration_previous_imp  + group_size + homeowner + married_couple,
                           data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock


#RunTime
#user  system elapsed
#730.899   6.797 742.187


#model summary,Var importance stats and plot
model.rf.B
randomForest::importance(model.rf.B)
randomForest::varImpPlot(model.rf.B)


post.train.rf.B <- predict(model.rf.B, train.purchase.m[trainSubset,])


# Predict random forest on validation set
post.valid.rf.B <- predict(model.rf.B, train.purchase.m[validSubset,])
length(post.valid.rf.B)


#Create a simple confusion matrix
table(post.valid.rf.B,train.purchase.m$B[validSubset])
# post.valid.rf.D     1     2     3
# 1  2977   186   204
# 2    94  5042   407
# 3    75   198 15069




#Check the misclassification rate
error.train.rf.B <- round(mean(post.train.rf.B!=train.purchase.m$B[trainSubset]),4)
error.train.rf.B
# 0.0553



#Check the misclassification rate
```

```
error.rf.B <- round(mean(post.valid.rf.B!=train.purchase.m$B[validSubset]),4)
error.rf.B
# 0.0682


#Compare against the misclassification rate for the base model
error.rf.B.base <-
round(mean(train.purchase.m$lastQuoted_B[validSubset]!=train.purchase.m$B[validSubset]),4)
error.rf.B.base
# 0.0664


# Fit Metrics
confusionMatrix(post.valid.rf.B,train.purchase.m$B[validSubset],)
#
# Accuracy : 0.9318
# Kappa : 0.863




# Class: 1 Class: 2 Class: 3
# Sensitivity            0.9463   0.9292   0.9610
# Specificity            0.9815   0.9734   0.9682
# Pos Pred Value         0.8842   0.9096   0.9822
# Neg Pred Value         0.9919   0.9795   0.9314
# Prevalence             0.1297   0.2237   0.6465
# Detection Rate         0.1228   0.2079   0.6214
# Detection Prevalence   0.1388   0.2286   0.6326
# Balanced Accuracy      0.9639   0.9513   0.9646


####################
# Boosting Model *B*
####################
set.seed(1)


str(train.purchase.m)
ptm <- proc.time() # Start the clock!


model.boost.B=gbm(B ~ (lastQuoted_B)+ risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state + C_previous_imp + duration_previous_imp +
                    Quoted_B_minus2 + Quoted_B_minus3 + Quoted_B_minus4  ,
                  data=train.purchase.m[trainSubset,],
                  distribution="multinomial",
                  n.trees=1000,
                  interaction.depth=2,
                  shrinkage = .01)


proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#151.54    0.47  153.98


#train error rate
post.train.boost.prob.B <- predict(model.boost.B,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.B<-apply(post.train.boost.prob.B, 1, which.max) - 1
train.boost.B <- round(mean(post.train.boost.B!=train.purchase.m$B[trainSubset]),4)
train.boost.B


#relative influence statistics & plot.
summary(model.boost.B)
summaryBoost<-summary(model.boost.B)
```

```
# Predict GBM on validation set
post.valid.boost.prob.B <- predict(model.boost.B,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.B<-apply(post.valid.boost.prob.B, 1, which.max) - 1
length(post.valid.boost.B)
head(post.valid.boost.B)


#Create a simple confusion matrix
table(post.valid.boost.B,train.purchase.m$B[validSubset])


#Check the misclassification rate
error.boost.B <- round(mean(post.valid.boost.B!=train.purchase.m$B[validSubset]),4)
error.boost.B


#Compare against the misclassification rate for the base model
error.boost.B.base <-
round(mean(train.purchase.m$lastQuoted_B[validSubset]!=train.purchase.m$B[validSubset]),4)
error.boost.B.base


# Fit Metrics
confusionMatrix(post.valid.boost.B,train.purchase.m$B[validSubset])


#plot relative influence of variables
summaryBoost<-summaryBoost[order(summaryBoost$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoost$rel.inf),names.arg = summaryBoost$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)



###################
# SVM *B*
###################
set.seed(1)


### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)


# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a radial kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 5,cross = 5)
# tune.out = tune(
#   svm,G ~ (lastQuoted_G) + risk_factor + car_age + car_value + cost + age_oldest + age_youngest + day +
shopping_pt + state +
#     Quoted_G_minus2 + Quoted_G_minus3 + Quoted_G_minus4  ,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(1)),
#   tunecontrol = control
# )
# summary(tune.out)
```

```
#Fit a linear SVM Model
ptm <- proc.time() # Start the clock!
svmfit.B=svm(B ~ (lastQuoted_B) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state + C_previous_imp + duration_previous_imp +
               Quoted_B_minus2 + Quoted_B_minus3 + Quoted_B_minus4  ,
             data=train.purchase.m.svm,
             kernel="linear",
             gamma=.01,
             cost=1,
             probability =TRUE)
proc.time() - ptm # Stop the clock


#Summary statitics
summary(svmfit.B)


#RunTime
# user  system elapsed
# 156.099   1.407 158.419


#Train Error Rate
post.train.svm.B<-predict(svmfit.B,train.purchase.m[trainSubset,])
train.svm.B <- round(mean(post.train.svm.B!=train.purchase.m$B[trainSubset]),4)
train.svm.B


# Predict SVM on validation set
post.valid.svm.B<-predict(svmfit.B,train.purchase.m[validSubset,])
length(post.valid.svm.B)


#Create a simple confusion matrix
table(post.valid.svm.B,train.purchase.m$B[validSubset])


#Check the misclassification rate
error.svm.B <- round(mean(post.valid.svm.B!=train.purchase.m$B[validSubset]),4)
error.svm.B
# [1] 0.1363


#Compare against the misclassification rate for the base model
error.svm.B.base <-
round(mean(train.purchase.m$lastQuoted_B[validSubset]!=train.purchase.m$B[validSubset]),4)
error.svm.B.base


confusionMatrix(post.valid.svm.B,train.purchase.m$B[validSubset])
##############################################################################
## Option *C* Models ##
##############################################################################
####################
#Baseline Model *C*
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda.C <- lda(C ~ (lastQuoted_C),
                   data = train.purchase.m,
                   subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
# user  system elapsed
```

```
# 0.13    0.00    2.24


#classification accuracy for training data
post.train.lda.C <- predict(object=model.lda.C, newdata = train.purchase.m[trainSubset,])
plot(model.lda.C, col = as.integer(train.purchase.m$C[-validSubset]), dimen = 2) #scatterplot with colors
table(post.train.lda.C$class, train.purchase.m$C[trainSubset]) #confusion matrix
mean(post.train.lda.C$class==train.purchase.m$C[trainSubset]) #what percent did we predict successfully?
plot(train.purchase.m$C[trainSubset], post.train.lda.C$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda.C <- predict(object=model.lda.C, newdata = train.purchase.m[validSubset,])
plot(model.lda.C, col = as.integer(train.purchase.m$C[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda.C$class, train.purchase.m$C[validSubset]) #confusion matrix
mean(post.valid.lda.C$class==train.purchase.m$C[validSubset]) #what percent did we predict successfully?
plot(train.purchase.m$C[validSubset], post.valid.lda.C$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?



confusionMatrix(post.valid.lda.C$class,train.purchase.m$C[validSubset])


modelCompare.df <- My.ModelCompare("LDA", "C",
post.valid.lda.C$class,train.purchase.m$C[validSubset]);modelCompare.df


#Check the misclassification rate
error.lda.C <- round(mean(post.valid.lda.C$class!=train.purchase.m$C[validSubset]),4)
error.lda.C
# 0.069


#Compare against the misclassification rate for the base model
error.lda.C.base <-
round(mean(train.purchase.m$lastQuoted_C[validSubset]!=train.purchase.m$C[validSubset]),4)
error.lda.C.base
# 0.069


confusionMatrix(post.valid.lda.C$class,train.purchase.m$C[validSubset])
# Kappa 0.9018


####################
#LDA *C*
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda.C <- lda(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                     Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 +
                     C_previous_imp + duration_previous_imp,
                   data = train.purchase.m,
                   subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#1.89    0.28    3.50


##############################################################################
## Option *C* Models ##
```

```
#############################################################################
####################
#LDA *C*
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda.C <- lda(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                     Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 +
                     C_previous_imp + duration_previous_imp,
                   data = train.purchase.m,
                   subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#1.89    0.28    3.50


#classification accuracy for training data
post.train.lda.C <- predict(object=model.lda.C, newdata = train.purchase.m[trainSubset,])
plot(model.lda.C, col = as.integer(train.purchase.m$C[-validSubset]), dimen = 2) #scatterplot with colors
table(post.train.lda.C$class, train.purchase.m$C[trainSubset]) #confusion matrix
mean(post.train.lda.C$class==train.purchase.m$C[trainSubset]) #what percent did we predict successfully?
plot(train.purchase.m$C[trainSubset], post.train.lda.C$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda.C <- predict(object=model.lda.C, newdata = train.purchase.m[validSubset,])
plot(model.lda.C, col = as.integer(train.purchase.m$C[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda.C$class, train.purchase.m$C[validSubset]) #confusion matrix
mean(post.valid.lda.C$class==train.purchase.m$C[validSubset]) #what percent did we predict successfully?
plot(train.purchase.m$C[validSubset], post.valid.lda.C$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?



confusionMatrix(post.valid.lda.C$class,train.purchase.m$C[validSubset])


modelCompare.df <- My.ModelCompare("LDA", "C",
post.valid.lda.C$class,train.purchase.m$C[validSubset]);modelCompare.df


#Check the misclassification rate
error.lda.C <- round(mean(post.valid.lda.C$class!=train.purchase.m$C[validSubset]),4)
error.lda.C
# 0.069


#Compare against the misclassification rate for the base model
error.lda.C.base <-
round(mean(train.purchase.m$lastQuoted_C[validSubset]!=train.purchase.m$C[validSubset]),4)
error.lda.C.base
# 0.069


confusionMatrix(post.valid.lda.C$class,train.purchase.m$C[validSubset])
# Kappa 0.9018


####################
# K-Nearest Neighbors *C*
####################
### Use this code to create a sample of the training data to fit a model   #PB
```

```
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


### KNN SAMPLING CODE (need 'train' and 'valid' in part column)   #FP
n <-dim(train.purchase.m)[1]
knn.sample <- sample(n, round(.25*n))
train.purchase.m.knn<-train.purchase.m[knn.sample,]
dim(train.purchase.m.knn)
View(train.purchase.m.knn)


train.purchase.m.knn$Quoted_C_minus3 = as.numeric(train.purchase.m.knn$Quoted_C_minus3)
train.purchase.m.knn$Quoted_C_minus2 = as.numeric(train.purchase.m.knn$Quoted_C_minus2)
train.purchase.m.knn$Quoted_C_minus4 = as.numeric(train.purchase.m.knn$Quoted_C_minus4)
train.purchase.m.knn$lastQuoted_C = as.numeric(train.purchase.m.knn$lastQuoted_C)
train.purchase.m.knn$C_previous_imp = as.numeric(train.purchase.m.knn$C_previous_imp)
train.purchase.m.knn$married_couple = as.numeric(train.purchase.m.knn$married_couple)
train.purchase.m.knn$car_value = as.numeric(train.purchase.m.knn$car_value)
train.purchase.m.knn$homeowner = as.numeric(train.purchase.m.knn$homeowner)
train.purchase.m.knn$state = as.numeric(train.purchase.m.knn$state)
train.purchase.m.knn$day = as.numeric(train.purchase.m.knn$day)


set.seed(1)
library(class)
dim(train.purchase.m.knn)
table(train.purchase.m.knn$part)
### 24,252 observations | 18,209 train | 6,043 valid


ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                  Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 + C_previous_imp +
duration_previous_imp  + group_size + homeowner + married_couple
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)


knnFit


# k   Accuracy   Kappa
# 5  0.8876797  0.8390917
# 7  0.8889162  0.8407584
# 9  0.8904425  0.8428784
# 11  0.8909782  0.8436213
# 13  0.8904832  0.8427947
#
# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was k = 11.


knnPredict <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knnPredict


### Define KNN training and test sets
knn.training <- train.purchase.m.knn[train.purchase.m.knn$part=="train",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,42,43,44,45)]
knn.test <- train.purchase.m.knn[train.purchase.m.knn$part=="valid",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,42,43,44,45)]
View(knn.training)
knn.trainLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="train", 20]
knn.testLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", 20]
```

```
colSums(is.na(knn.training))[colSums(is.na(knn.training)) > 0]
colSums(is.na(knn.test))[colSums(is.na(knn.test)) > 0]


table(knn.trainLabels)
table(knn.testLabels)


### Building classifier
knn.fit <- knn(train= knn.training, test= knn.test, cl = knn.trainLabels, k=11, prob=TRUE) #, l=0,
prob=TRUE, use.all = TRUE)


knn.fit


knn.valid <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knn.valid



plot(knn.fit)


library(gmodels)
CrossTable(x = knn.testLabels, y = knn.fit, prop.chisq=FALSE)


table(knnPredict, knn.testLabels)


#Check the misclassification rate
error.knn.C <- round(mean(knn.valid!=knn.testLabels),4)
error.knn.C
# 0.0984



confusionMatrix(knn.valid,knn.testLabels)
# Kappa 0.8591



####################
# RandomForest *C*
####################
set.seed(1)


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


model.rf.C <- randomForest(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
                             Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 +
                             C_previous_imp + duration_previous_imp + group_size + homeowner +
married_couple,
                           data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock


#RunTime
```

```
#user  system elapsed
#342.17    1.17  345.84


#model summary,Var importance stats and plot
model.rf.C
randomForest::importance(model.rf.C)
randomForest::varImpPlot(model.rf.C)


# Predict random forest on validation set
post.valid.rf.C <- predict(model.rf.C, train.purchase.m[validSubset,])
length(post.valid.rf.C)
#str(post.valid.rf.C)
#str(train.purchase.m$C[validSubset])
#Create a simple confusion matrix
table(post.valid.rf.C,train.purchase.m$C[validSubset])


#Check the misclassification rate
error.rf.C <- round(mean(post.valid.rf.C!=train.purchase.m$C[validSubset]),4)
error.rf.C
# 0.0693


#Compare against the misclassification rate for the base model
error.rf.C.base <-
round(mean(train.purchase.m$lastQuoted_C[validSubset]!=train.purchase.m$C[validSubset]),4)
error.rf.C.base
# 0.069


# Fit Metrics
confusionMatrix(post.valid.rf.C,train.purchase.m$C[validSubset])


# Kappa : 0.9018


####################
# Boosting Model *C*
####################
set.seed(1)


ptm <- proc.time() # Start the clock!
set.seed(1)
model.boost.C=gbm(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                    Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 + C_previous_imp +
duration_previous_imp  + group_size + homeowner + married_couple,
                  data=train.purchase.m[trainSubset,],
                  distribution="multinomial",
                  n.trees=1000,
                  interaction.depth=2,
                  shrinkage = .01)


proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#194.00    0.91  203.04


#relative influence statistics & plot.
summary(model.boost.C)
summaryBoost<-summary(model.boost.C)
```

```
# Predict DBM on validation set
post.train.boost.prob.C <- predict(model.boost.C,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.C<-apply(post.train.boost.prob.C, 1, which.max)


post.valid.boost.prob.C <- predict(model.boost.C,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.C<-apply(post.valid.boost.prob.C, 1, which.max)
length(post.valid.boost.C)
head(post.valid.boost.C)


#Create a simple confusion matrix
table(post.valid.boost.C,train.purchase.m$C[validSubset])


#Compare against the misclassification rate for the base model
error.train.boost.C.base <-
round(mean(train.purchase.m$lastQuoted_C[trainSubset]!=train.purchase.m$C[trainSubset]),4)
error.train.boost.C.base
#0.0693


train.error.boost.C <- round(mean(post.train.boost.C!=train.purchase.m$C[trainSubset]),4)
train.error.boost.C
#0.0691


#Check the misclassification rate
error.boost.C <- round(mean(post.valid.boost.C!=train.purchase.m$C[validSubset]),4)
error.boost.C
# 0.0693


#Compare against the misclassification rate for the base model
error.boost.C.base <-
round(mean(train.purchase.m$lastQuoted_C[validSubset]!=train.purchase.m$C[validSubset]),4)
error.boost.C.base
# 0.069


# Fit Metrics
confusionMatrix(post.valid.boost.C,train.purchase.m$C[validSubset])


#plot relative influence of variables
summaryBoost<-summaryBoost[order(summaryBoost$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoost$rel.inf),names.arg = summaryBoost$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)
#                Reference
#Prediction    1    2    3    4
#1          6801  228  161   25
#2           239 4563  211   21
#3           157  291 8983  198
#4            13   23  113 2225
# Accuracy : 0.9307
# Kappa : 0.9014


###################
# SVM *C*
###################
set.seed(1)


### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
```

```
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)
```

```
# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a linear kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 1,cross = 5)
# tune.out = tune(
#   svm,C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest +
day + shopping_pt + state +
#     Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 + C_previous_imp + duration_previous_imp  +
group_size,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(.01,.1,1)),
#   tunecontrol = control
# )
# summary(tune.out)


# - best parameters:
#   cost gamma
# 0.1  0.01
#
# - best performance: 0.06861316
#
# - Detailed performance results:
#   cost gamma      error  dispersion
# 1  0.01  0.01 0.06866815 0.004401212
# 2  0.10  0.01 0.06861316 0.004323864
# 3  0.50  0.01 0.06866815 0.004401212
# 4  1.00  0.01 0.06888807 0.004482941
# 5  0.01  0.10 0.06866815 0.004401212
# 6  0.10  0.10 0.06861316 0.004323864
# 7  0.50  0.10 0.06866815 0.004401212
# 8  1.00  0.10 0.06888807 0.004482941
# 9  0.01  1.00 0.06866815 0.004401212
# 10 0.10  1.00 0.06861316 0.004323864
# 11 0.50  1.00 0.06866815 0.004401212
# 12 1.00  1.00 0.06888807 0.004482941
```

```
#Fit a linear SVM Model
ptm <- proc.time() # Start the clock!
svmfit.C=svm(C ~ (lastQuoted_C) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state +
               Quoted_C_minus2 + Quoted_C_minus3 + Quoted_C_minus4 + C_previous_imp +
duration_previous_imp  + group_size , #+ homeowner + married_couple,
             data=train.purchase.m.svm,
             kernel="linear",
             gamma=.01,
             cost=.10,
             probability =TRUE)
proc.time() - ptm # Stop the clock
# user  system elapsed
# 30.56    0.25   33.93
```

```
#Summary statitics
summary(svmfit.C)
```

```
# Predict SVM on validation set
post.train.svm.C<-predict(svmfit.C,train.purchase.m[trainSubset,])
post.valid.svm.C<-predict(svmfit.C,train.purchase.m[validSubset,])
length(post.valid.svm.C)


#Create a simple confusion matrix
table(post.valid.svm.C,train.purchase.m$C[validSubset])
# post.valid.svm.C    1    2    3    4
# 1                6797  229  162   24
# 2                 242 4559  209   21
# 3                 157  291 8995  203
# 4                  14   26  102 2221
#Check the misclassification rate


#Compare against the misclassification rate for the base model
error.train.svm.C.base <-
round(mean(train.purchase.m$lastQuoted_C[trainSubset]!=train.purchase.m$C[trainSubset]),4)
error.train.svm.C.base
# 0.0693


train.error.svm.C <- round(mean(post.train.svm.C!=train.purchase.m$C[trainSubset]),4)
train.error.svm.C
# 0.0693


error.svm.C <- round(mean(post.valid.svm.C!=train.purchase.m$C[validSubset]),4)
error.svm.C
# 0.0692


#Compare against the misclassification rate for the base model
error.svm.C.base <-
round(mean(train.purchase.m$lastQuoted_C[validSubset]!=train.purchase.m$C[validSubset]),4)
error.svm.C.base
# 0.069


confusionMatrix(post.valid.svm.C,train.purchase.m$C[validSubset])
# Accuracy : 0.9308
# Kappa : 0.9015




################################################################################
## Option *D* Models ##
################################################################################


####################
#LDA *D*
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda.D <- lda(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
    Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size + homeowner + married_couple,
  data = train.purchase.m,
  subset = trainSubset)
```

```
proc.time() - ptm # Stop the clock
# #RunTime
# user  system elapsed
# 8.056   1.232  10.878


#classification accuracy for training data
post.train.lda.D <- predict(object=model.lda.D, newdata = train.purchase.m[trainSubset,])
# plot(model.lda.D, col = as.integer(train.purchase.m$D[-validSubset]), dimen = 2) #scatterplot with
colors
table(post.train.lda.D$class, train.purchase.m$D[trainSubset]) #confusion matrix
mean(post.train.lda.D$class!=train.purchase.m$D[trainSubset]) #what percent did we predict successfully?
# plot(train.purchase.m$D[trainSubset], post.train.lda.D$class, col=c("blue","red","yellow","green"),main
="Training Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda.D <- predict(object=model.lda.D, newdata = train.purchase.m[validSubset,])$class
length(post.valid.lda.D)
# plot(model.lda.D, col = as.integer(train.purchase.m$D[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda.D, train.purchase.m$D[validSubset]) #confusion matrix
# plot(train.purchase.m$D[validSubset], post.valid.lda.D$class, col=c("blue","red","yellow","green"),main
="Validation Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?


#Check the misclassification rate
error.lda.D <- round(mean(post.valid.lda.D!=train.purchase.m$D[validSubset]),4)
error.lda.D
# 0.0484


#Compare against the misclassification rate for the base model
error.lda.D.base <-
round(mean(train.purchase.m$lastQuoted_D[validSubset]!=train.purchase.m$D[validSubset]),4)
error.lda.D.base
# 0.0484


confusionMatrix(post.valid.lda.D,train.purchase.m$D[validSubset],)
# Kappa 0.9074


####################
# K-Nearest Neighbors *D*
####################
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)
library(class)


ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
    Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size + homeowner + married_couple
  , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess = c("center","scale"),
tuneLength = 5)



# Resampling results across tuning parameters:
#
#   k   Accuracy   Kappa      Accuracy SD  Kappa SD
```

```
# 5  0.9207772  0.8450820  0.005632213  0.011662121
# 7  0.9225365  0.8484299  0.007032948  0.014358392
# 9  0.9216569  0.8463100  0.005811135  0.011981198
# 11  0.9205566  0.8437952  0.004690705  0.009812055
# 13  0.9205023  0.8434500  0.005354910  0.011164977
#
# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was k = 7.


knnPredict <- predict(knnFit,newdata = train.purchase.m[validSubset,])




knn.trainLabels <- train.purchase.m.knn[,c('D')]
knn.testLabels <- train.purchase.m[validSubset,c('D')]


summary(knn.testLabels)
### Building classifier
knn_pred <- knn(train = knn.training, test = knn.test, cl = knn.trainLabels, k=3)
knn_pred


library(gmodels)
CrossTable(x = knn.testLabels, y = knn_pred, prop.chisq=FALSE)


####################
# RandomForest *D*
####################
set.seed(1)


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


model.rf.D <- randomForest(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
    Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size + homeowner + married_couple,
  data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock


#RunTime
#user  system elapsed
#730.899   6.797 742.187


#model summary,Var importance stats and plot
model.rf.D
randomForest::importance(model.rf.D)
randomForest::varImpPlot(model.rf.D)


post.train.rf.D <- predict(model.rf.D, train.purchase.m[trainSubset,])


# Predict random forest on validation set
post.valid.rf.D <- predict(model.rf.D, train.purchase.m[validSubset,])
length(post.valid.rf.D)
```

```
#Create a simple confusion matrix
table(post.valid.rf.D,train.purchase.m$D[validSubset])
# post.valid.rf.D     1     2     3
# 1  2977   186   204
# 2    94  5042   407
# 3    75   198 15069



#Check the misclassification rate
error.train.rf.D <- round(mean(post.train.rf.D!=train.purchase.m$D[trainSubset]),4)
error.train.rf.D
# 0.0463



#Check the misclassification rate
error.rf.D <- round(mean(post.valid.rf.D!=train.purchase.m$D[validSubset]),4)
error.rf.D
# 0.048


#Compare against the misclassification rate for the base model
error.rf.D.base <-
round(mean(train.purchase.m$lastQuoted_D[validSubset]!=train.purchase.m$D[validSubset]),4)
error.rf.D.base
# 0.0484


# Fit Metrics
confusionMatrix(post.valid.rf.D,train.purchase.m$D[validSubset],)
#
# Accuracy : 0.952
# Kappa : 0.908



# Class: 1 Class: 2 Class: 3
# Sensitivity            0.9463   0.9292   0.9610
# Specificity            0.9815   0.9734   0.9682
# Pos Pred Value         0.8842   0.9096   0.9822
# Neg Pred Value         0.9919   0.9795   0.9314
# Prevalence             0.1297   0.2237   0.6465
# Detection Rate         0.1228   0.2079   0.6214
# Detection Prevalence   0.1388   0.2286   0.6326
# Balanced Accuracy      0.9639   0.9513   0.9646


####################
# Boosting Model *D*
####################
set.seed(1)


#### Use this code to tune the GBM model. ###


# library(caret)
# myTuneGrid <- expand.grid(n.trees = 500,interaction.depth = c(2,3),shrinkage = c(01),n.minobsinnode=10)
# fitControl <- trainControl(method = "repeatedcv", number = 3,repeats = 1, verboseIter =
FALSE,returnResamp = "all")
# myModel <- train(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
# Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size + homeowner + married_couple,
#                   data=train.purchase.m[trainSubset,],
```

```
#                     method = "gbm",
#                     trControl = fitControl,
#                     tuneGrid = myTuneGrid)
#


ptm <- proc.time() # Start the clock!
set.seed(1)
model.boost.D=gbm(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
    Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size + homeowner + married_couple,
  data=train.purchase.m[trainSubset,],
  distribution="multinomial",
  n.trees=1000,
  interaction.depth=2,
  shrinkage = .01)


proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#283.950   4.094 302.237


#relative influence statistics & plot.
summary(model.boost.D)
summaryBoost<-summary(model.boost.D)


# Predict DBM on validation set
post.train.boost.prob.D <- predict(model.boost.D,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.D<-apply(post.train.boost.prob.D, 1, which.max)


post.valid.boost.prob.D <- predict(model.boost.D,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.D<-apply(post.valid.boost.prob.D, 1, which.max)
length(post.valid.boost.D)
head(post.valid.boost.D)


#Create a simple confusion matrix
table(post.valid.boost.D,train.purchase.m$D[validSubset])


#Compare against the misclassification rate for the base model
error.train.boost.D.base <-
round(mean(train.purchase.m$lastQuoted_D[trainSubset]!=train.purchase.m$D[trainSubset]),4)
error.train.boost.D.base
#0.0516


train.error.boost.D <- round(mean(post.train.boost.D!=train.purchase.m$D[trainSubset]),4)
train.error.boost.D
#0.0512


#Check the misclassification rate
error.boost.D <- round(mean(post.valid.boost.D!=train.purchase.m$D[validSubset]),4)
error.boost.D
# 0.0474


#Compare against the misclassification rate for the base model
error.boost.D.base <-
round(mean(train.purchase.m$lastQuoted_D[validSubset]!=train.purchase.m$D[validSubset]),4)
error.boost.D.base
```

```
# 0.0484


# Fit Metrics
confusionMatrix(post.valid.boost.D,train.purchase.m$D[validSubset],)


#plot relative influence of variables
summaryBoost<-summaryBoost[order(summaryBoost$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoost$rel.inf),names.arg = summaryBoost$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)
# 1  2985   182   202
# 2    94  5043   404
# 3    67   201 15074
# Accuracy : 0.9526
# Kappa : 0.9091


###################
# SVM *D*
###################
set.seed(1)


### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)









# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a linear kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 1,cross = 5)
# tune.out = tune(
#   svm,D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest +
day + shopping_pt + state +
#     Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(.01,.1,1)),
#   tunecontrol = control
# )
# summary(tune.out)




#Fit a linear SVM Model
ptm <- proc.time() # Start the clock!
svmfit.D=svm(D ~ (lastQuoted_D) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state +
    Quoted_D_minus2 + Quoted_D_minus3 + Quoted_D_minus4 + C_previous_imp + duration_previous_imp  +
group_size,
  data=train.purchase.m.svm,
  kernel="linear",
  gamma=.01,
  cost=1,
  probability =TRUE)
proc.time() - ptm # Stop the clock
```

```
#Summary statitics
summary(svmfit.D)


#RunTime
# user  system elapsed
# 156.099   1.407 158.419


# Predict SVM on validation set
post.train.svm.D<-predict(svmfit.D,train.purchase.m[trainSubset,])
post.valid.svm.D<-predict(svmfit.D,train.purchase.m[validSubset,])
length(post.valid.svm.D)


#Create a simple confusion matrix
table(post.valid.svm.D,train.purchase.m$D[validSubset])
# post.valid.svm.D     1     2     3
# 1  2987   183   224
# 2    94  5049   408
# 3    65   194 15048
#Check the misclassification rate


#Compare against the misclassification rate for the base model
error.train.svm.D.base <-
round(mean(train.purchase.m$lastQuoted_D[trainSubset]!=train.purchase.m$D[trainSubset]),4)
error.train.svm.D.base
# 0.0516


train.error.svm.D <- round(mean(post.train.svm.D!=train.purchase.m$D[trainSubset]),4)
train.error.svm.D
# 0.0515


error.svm.D <- round(mean(post.valid.svm.D!=train.purchase.m$D[validSubset]),4)
error.svm.D
# 0.0482


#Compare against the misclassification rate for the base model
error.svm.D.base <-
round(mean(train.purchase.m$lastQuoted_D[validSubset]!=train.purchase.m$D[validSubset]),4)
error.svm.D.base
# 0.0484


confusionMatrix(post.valid.svm.D,train.purchase.m$D[validSubset])
# Accuracy : 0.9518
# Kappa : 0.9078




##############################################################################
## Option *E* Models ##
##############################################################################


# Create naive RF model to select predictors for rest of models
ptm <- proc.time() # Start the clock!
set.seed(1)
model.RF.naive.E <- randomForest(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost +
age_oldest + age_youngest + day + shopping_pt + state +
                                 Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + homeowner +
married_couple + group_size + C_previous_imp + duration_previous_imp,
```

```
                                data = train.purchase.m, ntree =500)
proc.time() - ptm # Stop the clock
# user  system elapsed
# 226.74    3.32  250.17


varImpPlot(model.RF.naive.E, main = "Random Forest Model: \n Variable Importance")
importance(model.RF.naive.E)
#homeowner + married_couple + group_size don't appear as important--will leave out for other models


####################
#LDA *E*
####################
set.seed(1)


##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda0.e <- lda(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                      Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp +
duration_previous_imp,
                    data = train.purchase.m,
                    subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
# user  system elapsed
# 2.42    0.45    4.68


#classification accuracy for training data
post.train.lda0.e <- predict(object=model.lda0.e, newdata = train.purchase.m[trainSubset,])
plot(model.lda0.e, col = as.integer(train.purchase.m$E[-validSubset]), dimen = 2) #scatterplot with colors
table(post.train.lda0.e$class, train.purchase.m$E[trainSubset]) #confusion matrix
# 0     1
# 0 37213  2681
# 1  1866 30997


(mean(post.train.lda0.e$class!=train.purchase.m$E[trainSubset]))
#0.0624957
plot(train.purchase.m$E[trainSubset], post.train.lda0.e$class, col=c("blue","orange"),main ="Training
Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict trainSubset?


#classification accuracy for validation data
post.valid.lda0.e <- predict(object=model.lda0.e, newdata = train.purchase.m[validSubset,])
plot(model.lda0.e, col = as.integer(train.purchase.m$E[validSubset]), dimen = 2) #scatterplot with colors
table(post.valid.lda0.e$class, train.purchase.m$E[validSubset]) #confusion matrix
# 0     1
# 0 12433   900
# 1   620 10299


(mean(post.valid.lda0.e$class!=train.purchase.m$E[validSubset])) #what percent did we predict
successfully?
#0.06267524
plot(train.purchase.m$E[validSubset], post.valid.lda0.e$class, col=c("blue","orange"),main ="Validation
Set", xlab = "Actual Choice", ylab="Predicted Choice") #how well did we predict validSubset?


confusionMatrix(post.valid.lda0.e$class,train.purchase.m$E[validSubset],)
# Confusion Matrix and Statistics
#
# Reference
# Prediction     0     1
```

```
# 0 12433   900
# 1   620 10299
#
# Accuracy : 0.9373
# 95% CI : (0.9342, 0.9403)
# No Information Rate : 0.5382
# P-Value [Acc > NIR] : < 2.2e-16
#
# Kappa : 0.8737
# Mcnemar's Test P-Value : 0.0000000000008294
#
# Sensitivity : 0.9525
# Specificity : 0.9196
# Pos Pred Value : 0.9325
# Neg Pred Value : 0.9432
# Prevalence : 0.5382
# Detection Rate : 0.5127
# Detection Prevalence : 0.5498
# Balanced Accuracy : 0.9361
#
# 'Positive' Class : 0



####################
# K-Nearest Neighbors *E*  -- can't get KNN to work
####################
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
set.seed(1)
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


### KNN SAMPLING CODE (need 'train' and 'valid' in part column)   #FP
set.seed(1)
n <-dim(train.purchase.m)[1]
knn.sample <- sample(n, round(.25*n))
train.purchase.m.knn<-train.purchase.m[knn.sample,]
dim(train.purchase.m.knn)
#View(train.purchase.m.knn)


train.purchase.m.knn$Quoted_E_minus3 = as.numeric(train.purchase.m.knn$Quoted_E_minus3)
train.purchase.m.knn$Quoted_E_minus2 = as.numeric(train.purchase.m.knn$Quoted_E_minus2)
train.purchase.m.knn$Quoted_E_minus4 = as.numeric(train.purchase.m.knn$Quoted_E_minus4)
train.purchase.m.knn$lastQuoted_C = as.numeric(train.purchase.m.knn$lastQuoted_E)
train.purchase.m.knn$C_previous_imp = as.numeric(train.purchase.m.knn$C_previous_imp)
train.purchase.m.knn$married_couple = as.numeric(train.purchase.m.knn$married_couple)
train.purchase.m.knn$car_value = as.numeric(train.purchase.m.knn$car_value)
train.purchase.m.knn$homeowner = as.numeric(train.purchase.m.knn$homeowner)
train.purchase.m.knn$state = as.numeric(train.purchase.m.knn$state)
train.purchase.m.knn$day = as.numeric(train.purchase.m.knn$day)


set.seed(1)
library(class)
dim(train.purchase.m.knn)
table(train.purchase.m.knn$part)
### 24,252 observations | 18,260 train | 5,992 valid


ptm <- proc.time() # Start the clock!
ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
```

```
                  Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp +
duration_previous_imp
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)
proc.time() - ptm # Stop the clock


knnFit
# k-Nearest Neighbors
#
# 24252 samples
# 15 predictor
# 2 classes: '0', '1'
#
# Pre-processing: centered (15), scaled (15)
# Resampling: Cross-Validated (10 fold, repeated 1 times)
# Summary of sample sizes: 21828, 21827, 21826, 21826, 21826, 21827, ...
# Resampling results across tuning parameters:
#
#   k   Accuracy   Kappa
# 5  0.9301493  0.8593003
# 7  0.9310565  0.8611145
# 9  0.9317162  0.8624164
# 11  0.9310977  0.8611740
# 13  0.9308916  0.8607528
#
# Accuracy was used to select the optimal model using  the largest value.
# The final value used for the model was k = 9.


knnPredict <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knnPredict


### Define KNN training and test sets
knn.training <- train.purchase.m.knn[train.purchase.m.knn$part=="train",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,34,35,36,37)]
knn.test <- train.purchase.m.knn[train.purchase.m.knn$part=="valid",
c(2,4,6,8,9,10,11,13,14,15,25,26,27,28,34,35,36,37)]
#View(knn.training)
knn.trainLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="train", 18]
knn.testLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", 18]


colSums(is.na(knn.training))[colSums(is.na(knn.training)) > 0]
colSums(is.na(knn.test))[colSums(is.na(knn.test)) > 0]


table(knn.trainLabels)
table(knn.testLabels)


### Building classifier
knn.fit <- knn(train= knn.training, test= knn.test, cl = knn.trainLabels, k=11, prob=TRUE) #, l=0,
prob=TRUE, use.all = TRUE)


knn.fit


knn.valid <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knn.valid


plot(knn.fit)


library(gmodels)
CrossTable(x = knn.testLabels, y = knn.fit, prop.chisq=FALSE)
```

```
table(knnPredict, knn.testLabels)
# knn.testLabels
# knnPredict    0    1    2
# 0 1220 1594  504
# 1   58 2122  494


#Check the misclassification rate
error.knn.E <- round(mean(knn.valid!=knn.testLabels),4)
error.knn.E
# 0.0741


confusionMatrix(knn.valid,knn.testLabels)
#Confusion Matrix and Statistics


#Reference
#Prediction    0    1    2
#0 1171   56   20
#1   83 3592  193
#2   24   68  785


#Overall Statistics


#Accuracy : 0.9259
#95% CI : (0.919, 0.9324)
#No Information Rate : 0.6202
#P-Value [Acc > NIR] : < 2.2e-16


#Kappa : 0.8604
#Mcnemar's Test P-Value : 3.971e-14


#Statistics by Class:


#Class: 0 Class: 1 Class: 2
#Sensitivity            0.9163   0.9666   0.7866
#Specificity            0.9839   0.8787   0.9816
#Pos Pred Value         0.9391   0.9286   0.8951
#Neg Pred Value         0.9774   0.9416   0.9584
#Prevalence             0.2133   0.6202   0.1666
#Detection Rate         0.1954   0.5995   0.1310
#Detection Prevalence   0.2081   0.6455   0.1464
#Balanced Accuracy      0.9501   0.9227   0.8841


####################
# RandomForest *E*
####################
set.seed(1)
ptm <- proc.time() # Start the clock!
model.rf.E <- randomForest(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
                             Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp +
duration_previous_imp,
                           data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock
# user  system elapsed
# 126.42    2.05  144.27
```

```
#model summary,Var importance stats and plot
model.rf.E
randomForest::importance(model.rf.E)
randomForest::varImpPlot(model.rf.E)


# Predict random forest on validation set
post.valid.rf.E <- predict(model.rf.E, train.purchase.m[validSubset,])
length(post.valid.rf.E)


#Create a simple confusion matrix
table(post.valid.rf.E,train.purchase.m$E[validSubset])
# post.valid.rf.E     0     1
# 0 12509  1088
# 1   544 10111


#Check the misclassification rate
error.rf.E <- round(mean(post.valid.rf.E!=train.purchase.m$E[validSubset]),4)
error.rf.E
#0.0673


#Compare against the misclassification rate for the base model
error.rf.E.base <-
round(mean(train.purchase.m$lastQuoted_E[validSubset]!=train.purchase.m$E[validSubset]),4)
error.rf.E.base
#0.0627


# Fit Metrics
confusionMatrix(post.valid.rf.E,train.purchase.m$E[validSubset],)
# Confusion Matrix and Statistics
#
# Reference
# Prediction     0     1
# 0 12509  1088
# 1   544 10111
#
# Accuracy : 0.9327
# 95% CI : (0.9295, 0.9358)
# No Information Rate : 0.5382
# P-Value [Acc > NIR] : < 2.2e-16
#
# Kappa : 0.8642
# Mcnemar's Test P-Value : < 2.2e-16
#
#             Sensitivity : 0.9583
#             Specificity : 0.9028
#          Pos Pred Value : 0.9200
#          Neg Pred Value : 0.9489
#              Prevalence : 0.5382
#          Detection Rate : 0.5158
#    Detection Prevalence : 0.5607
#       Balanced Accuracy : 0.9306
#
#        'Positive' Class : 0



####################
# Boosting Model *E*
####################


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

```
set.seed(1)
model.boost.E=gbm(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                    Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp +
duration_previous_imp,
                  data=train.purchase.m[trainSubset,],
                  distribution="multinomial",
                  n.trees=1000,
                  interaction.depth=4,
                  shrinkage = .01)


proc.time() - ptm # Stop the clock
#user  system elapsed
# 208.92    1.59  288.30


#relative influence statistics & plot.
summary(model.boost.E)
#var      rel.inf
# lastQuoted_E                   lastQuoted_E 95.343839429
# Quoted_E_minus2             Quoted_E_minus2  1.388196071
# cost                                   cost  0.824369609
# car_age                             car_age  0.763640385
# state                                 state  0.445824881
# Quoted_E_minus3             Quoted_E_minus3  0.412431425
# Quoted_E_minus4             Quoted_E_minus4  0.242647120
# age_youngest                   age_youngest  0.240936151
# age_oldest                       age_oldest  0.140383532
# shopping_pt                     shopping_pt  0.083447996
# C_previous_imp               C_previous_imp  0.034988286
# risk_factor_imp             risk_factor_imp  0.031847367
# car_value                         car_value  0.022677371
# duration_previous_imp duration_previous_imp  0.018561585
# day                                   day  0.006208792
#summaryBoostE<-summary(model.boost.E)


# Predict ABM on validation set
post.train.boost.prob.E <- predict(model.boost.E,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.E<-apply(post.train.boost.prob.E, 1, which.max)


post.valid.boost.prob.E <- predict(model.boost.E,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.E<-apply(post.valid.boost.prob.E, 1, which.max)-1
length(post.valid.boost.E)
head(post.valid.boost.E)


#Create a simple confusion matrix
table(post.valid.boost.E,train.purchase.m$E[validSubset])
# post.valid.boost.E     0     1
# 0 12457   911
# 1   596 10288


#Compare against the misclassification rate for the base model
error.train.boost.E.base <-
round(mean(train.purchase.m$lastQuoted_E[trainSubset]!=train.purchase.m$E[trainSubset]),4)
error.train.boost.E.base
#0.0625


train.error.boost.E <- round(mean(post.train.boost.E!=train.purchase.m$E[trainSubset]),4)
train.error.boost.E
#0.9629
```

```
#Check the misclassification rate
error.boost.E <- round(mean(post.valid.boost.E!=train.purchase.m$E[validSubset]),4)
error.boost.E
# 0.0621


#Compare against the misclassification rate for the base model
error.boost.E.base <-
round(mean(train.purchase.m$lastQuoted_E[validSubset]!=train.purchase.m$E[validSubset]),4)
error.boost.E.base
# 0.0627


# Fit Metrics
# confusionMatrix(post.valid.boost.E,train.purchase.m$E[validSubset])
# Confusion Matrix and Statistics
#
# Reference
# Prediction     0     1
# 0 12457   911
# 1   596 10288
#
# Accuracy : 0.9379
# 95% CI : (0.9347, 0.9409)
# No Information Rate : 0.5382
# P-Value [Acc > NIR] : < 2.2e-16
#
# Kappa : 0.8747
# Mcnemar's Test P-Value : 6.036e-16
#
#             Sensitivity : 0.9543
#             Specificity : 0.9187
#          Pos Pred Value : 0.9319
#          Neg Pred Value : 0.9452
#              Prevalence : 0.5382
#          Detection Rate : 0.5136
#    Detection Prevalence : 0.5512
#       Balanced Accuracy : 0.9365
#
#        'Positive' Class : 0



#plot relative influence of variables
summaryBoostE<-summaryBoostE[order(summaryBoostE$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoostE$rel.inf),names.arg = summaryBoostE$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)



###################
# SVM *A*
###################
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)
#18189    62



#Fit a linear SVM Model
```

```
ptm <- proc.time() # Start the clock!
svmfit.E=svm(E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest
+ day + shopping_pt + state +
               Quoted_E_minus2 + Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp +
duration_previous_imp,
             data=train.purchase.m.svm,
             kernel="linear",
             gamma=.01,
             cost=1,
             probability =TRUE)
proc.time() - ptm # Stop the clock


#Summary statitics
summary(svmfit.E)
# svm(formula = E ~ (lastQuoted_E) + risk_factor_imp + car_age + car_value +
#       cost + age_oldest + age_youngest + day + shopping_pt + state + Quoted_E_minus2 +
#       Quoted_E_minus3 + Quoted_E_minus4 + C_previous_imp + duration_previous_imp,
#     data = train.purchase.m.svm, kernel = "linear", gamma = 0.01, cost = 1,
#     probability = TRUE)
#
#
# Parameters:
#   SVM-Type:  C-classification
# SVM-Kernel:  linear
# cost:  1
# gamma:  0.01
#
# Number of Support Vectors:  3427
#
# ( 1633 1794 )
#
#
# Number of Classes:  2
#
# Levels:
#   0 1


#RunTime
# user  system elapsed
# 65.39    0.28   97.41


# Predict SVM on validation set
post.valid.svm.E<-predict(svmfit.E,train.purchase.m[validSubset,])
length(post.valid.svm.E) #24252


#Create a simple confusion matrix
table(post.valid.svm.E,train.purchase.m$E[validSubset])
# post.valid.svm.E     0     1
# 0 12433   900
# 1   620 10299


#Check the misclassification rate
error.svm.E <- round(mean(post.valid.svm.E!=train.purchase.m$E[validSubset]),4)
error.svm.E
# [1] 0.0627


#Compare against the misclassification rate for the base model
error.svm.E.base <-
round(mean(train.purchase.m$lastQuoted_E[validSubset]!=train.purchase.m$E[validSubset]),4)
error.svm.E.base #0.0627


# Fit Metrics
```

```
confusionMatrix(post.valid.svm.E,train.purchase.m$E[validSubset],)
# Confusion Matrix and Statistics
#
# Reference
# Prediction     0     1
# 0 12433   900
# 1   620 10299
#
# Accuracy : 0.9373
# 95% CI : (0.9342, 0.9403)
# No Information Rate : 0.5382
# P-Value [Acc > NIR] : < 2.2e-16
#
# Kappa : 0.8737
# Mcnemar's Test P-Value : 0.0000000000008294
#
#             Sensitivity : 0.9525
#             Specificity : 0.9196
#          Pos Pred Value : 0.9325
#          Neg Pred Value : 0.9432
#              Prevalence : 0.5382
#          Detection Rate : 0.5127
#    Detection Prevalence : 0.5498
#       Balanced Accuracy : 0.9361
#
#        'Positive' Class : 0
```

```
################################################################################
## Option *F* Models ##
################################################################################
set.seed(1)


####################
#LDA *F*
####################
set.seed(1)
##Initial Full model


ptm <- proc.time() # Start the clock!
model.lda0.f <- lda(F ~ (lastQuoted_F) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                      Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4 + group_size + homeowner +
married_couple,
                    data = train.purchase.m,
                    subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#   user  system elapsed
#   3.48    0.15    3.69


#classification accuracy for training data
post.train.lda0.f <- predict(model.lda0.f, newdata=train.purchase.m[trainSubset,])
table(post.train.lda0.f$class, train.purchase.m[trainSubset,]$F) #confusion matrix
mean(post.train.lda0.f$class!=train.purchase.m[trainSubset,]$F) #misclassification percent 0.07236417


#classification accuracy for validation data
```

```
post.valid.lda0.f <- predict(model.lda0.f, newdata=train.purchase.m[validSubset,])
table(post.valid.lda0.f$class, train.purchase.m[validSubset,]$F) #confusion matrix
mean(post.valid.lda0.f$class!=train.purchase.m[validSubset,]$F) #misclassification percent 0.07314861


#confusion matrix
confusionMatrix(post.valid.lda0.f$class,train.purchase.m$F[validSubset],) #Kappa : 0.8941


ptm <- proc.time() # Start the clock!
model.lda1.f <- lda(F ~ (lastQuoted_F) +
                      Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4,
                    data = train.purchase.m,
                    subset = trainSubset)
proc.time() - ptm # Stop the clock
#RunTime
#   user  system elapsed
#   0.39    0.07    0.50


post.train.lda1.f <- predict(model.lda1.f, newdata=train.purchase.m[trainSubset,])
table(post.train.lda1.f$class, train.purchase.m[trainSubset,]$F)
mean(post.train.lda1.f$class!=train.purchase.m[trainSubset,]$F) #0.07236417


post.valid.lda1.f <- predict(model.lda1.f, newdata=train.purchase.m[validSubset,])
table(post.valid.lda1.f$class, train.purchase.m[validSubset,]$F)
mean(post.valid.lda1.f$class!=train.purchase.m[validSubset,]$F) #0.07236417


####################
# K-Nearest Neighbors *F*
####################
set.seed(1)
### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
set.seed(1)
knn.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.knn<-train.purchase.m[train.purchase.m$part=="train",][knn.sample,]
dim(train.purchase.m.knn)


### KNN SAMPLING CODE (need 'train' and 'valid' in part column)   #FP
n <-dim(train.purchase.m)[1]
knn.sample <- sample(n, round(.25*n))
train.purchase.m.knn<-train.purchase.m[knn.sample,]
dim(train.purchase.m.knn)
View(train.purchase.m.knn)


train.purchase.m.knn$Quoted_F_minus3 = as.numeric(train.purchase.m.knn$Quoted_F_minus3)
train.purchase.m.knn$Quoted_F_minus2 = as.numeric(train.purchase.m.knn$Quoted_F_minus2)
train.purchase.m.knn$Quoted_F_minus4 = as.numeric(train.purchase.m.knn$Quoted_F_minus4)
train.purchase.m.knn$lastQuoted_F = as.numeric(train.purchase.m.knn$lastQuoted_F)
train.purchase.m.knn$C_previous_imp = as.numeric(train.purchase.m.knn$C_previous_imp)
train.purchase.m.knn$married_couple = as.numeric(train.purchase.m.knn$married_couple)
train.purchase.m.knn$car_value = as.numeric(train.purchase.m.knn$car_value)
train.purchase.m.knn$homeowner = as.numeric(train.purchase.m.knn$homeowner)
train.purchase.m.knn$state = as.numeric(train.purchase.m.knn$state)
train.purchase.m.knn$day = as.numeric(train.purchase.m.knn$day)


set.seed(1)
library(class)
dim(train.purchase.m.knn)
table(train.purchase.m.knn$part)
### 24,252 observations | 18,209 train | 6,043 valid
```

```
ctrl <- trainControl(method="repeatedcv",repeats = 1) #,classProbs=TRUE,summaryFunction = twoClassSummary)
knnFit <- train(F ~ (lastQuoted_F) + risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                  Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4 + group_size + homeowner +
married_couple
                , data = train.purchase.m.knn, method = "knn", trControl = ctrl, preProcess =
c("center","scale"), tuneLength = 5)


knnFit


#  k   RMSE       Rsquared
#   5  0.4171145  0.8063309
#   7  0.4112254  0.8114590
#   9  0.4089547  0.8135273
#  11  0.4075037  0.8149351
#  13  0.4071391  0.8153405
#
#RMSE was used to select the optimal model using  the smallest value.
#The final value used for the model was k = 13.


knnPredict <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part!="valid",])
knnPredict


### Define KNN training and test sets
knn.training <- train.purchase.m.knn[train.purchase.m.knn$part=="train", c('lastQuoted_F',
'risk_factor_imp','car_age', 'car_value','cost','age_oldest','age_youngest','day','shopping_pt','state',

'Quoted_F_minus2','Quoted_F_minus3','Quoted_F_minus4','group_size','homeowner','married_couple')]
knn.test <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", c('lastQuoted_F',
'risk_factor_imp','car_age', 'car_value','cost','age_oldest','age_youngest','day','shopping_pt','state',

'Quoted_F_minus2','Quoted_F_minus3','Quoted_F_minus4','group_size','homeowner','married_couple')]


View(knn.training)
knn.trainLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="train", 23]
knn.testLabels <- train.purchase.m.knn[train.purchase.m.knn$part=="valid", 23]


colSums(is.na(knn.training))[colSums(is.na(knn.training)) > 0]
colSums(is.na(knn.test))[colSums(is.na(knn.test)) > 0]


table(knn.trainLabels)
table(knn.testLabels)


### Building classifier
knn.fit <- knn(train= knn.training, test= knn.test, cl = knn.trainLabels, k=13, prob=TRUE) #, l=0,
prob=TRUE, use.all = TRUE)


knn.fit


knn.valid <- predict(knnFit,newdata = train.purchase.m.knn[train.purchase.m.knn$part=="valid",])
knn.valid



plot(knn.fit)
```

```
library(gmodels)
CrossTable(x = knn.testLabels, y = knn.fit, prop.chisq=FALSE)


#training
table(round(knnPredict), knn.trainLabels)


#Check the misclassification rate
error.knn.F <- round(mean(knnPredict!=knn.trainLabels),4)
error.knn.F


confusionMatrix(round(knnPredict),knn.trainLabels)
# Kappa 0.8901


#validation
table(round(knn.valid), knn.testLabels)


#Check the misclassification rate
error.knn.F <- round(mean(knn.valid!=knn.testLabels),4)
error.knn.F


confusionMatrix(round(knn.valid),knn.testLabels)
# Kappa 0.8897


####################
# RandomForest *F*
####################
set.seed(1)


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


model.rf.f <- randomForest(F ~ (lastQuoted_F) + risk_factor_imp + car_age + car_value + cost + age_oldest
+ age_youngest + day + shopping_pt + state +
                             Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4 ,
                           data=train.purchase.m,subset = trainSubset,ntrees=500)


proc.time() - ptm # Stop the clock


#RunTime
#user  system elapsed
#114.52    0.93  116.36


#model summary,Var importance stats and plot
model.rf.f
randomForest::importance(model.rf.f)
randomForest::varImpPlot(model.rf.f)


# Predict random forest on validation set
post.train.rf.f <- predict(model.rf.f,train.purchase.m[trainSubset,])
post.valid.rf.f <- predict(model.rf.f, train.purchase.m[validSubset,])


table(post.train.rf.f,train.purchase.m$F[trainSubset])
table(post.valid.rf.f,train.purchase.m$F[validSubset])


#Check the misclassification rate
```

```
  #training
  round(mean(post.train.rf.f!=train.purchase.m$F[trainSubset]),4) #0.0622
  #validation
  error.rf.f <- round(mean(post.valid.rf.f!=train.purchase.m$F[validSubset]),4)
  error.rf.f #0.0725


#Compare against the misclassification rate for the base model
error.rf.f.base <-
round(mean(train.purchase.m$lastQuoted_F[validSubset]!=train.purchase.m$F[validSubset]),4)
error.rf.f.base  #0.0731


# Fit Metrics
confusionMatrix(post.valid.rf.f,train.purchase.m$F[validSubset],) #Kappa : 0.895


####################
# Boosting Model *F*
####################
set.seed(1)


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


model.boost.F=gbm(F ~ (lastQuoted_F)+ risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
                    Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4  ,
                  data=train.purchase.m[trainSubset,],
                  distribution="multinomial",
                  n.trees=1000,
                  interaction.depth=2,
                  shrinkage = .01)


proc.time() - ptm # Stop the clock
#RunTime
#user  system elapsed
#151.54    0.47  153.98


#relative influence statistics & plot.
summary(model.boost.F)
summaryBoost<-summary(model.boost.F)


#Predict GBM on training set
post.train.boost.F <-     predict(model.boost.F,
train.purchase.m[trainSubset,],type='response',n.trees=1000)
post.train.boost.F<-apply(post.train.boost.F, 1, which.max) - 1


# Predict GBM on validation set
post.valid.boost.prob.F <-predict(model.boost.F,
train.purchase.m[validSubset,],type='response',n.trees=1000)
post.valid.boost.F<-apply(post.valid.boost.prob.F, 1, which.max) - 1




#Create a simple confusion matrix
table(post.valid.boost.F,train.purchase.m$F[validSubset])
#    0    1    2    3
#0 7612   95  185   46
#1   83 5339  303   34
#2  178  407 8425  233
#3   29   36  127 1120
```

```
#Check the misclassification rate
  #training
  round(mean(post.train.boost.F!=train.purchase.m$F[trainSubset]),4) #0.0708

  #validation
  error.boost.F <- round(mean(post.valid.boost.F!=train.purchase.m$F[validSubset]),4)
  error.boost.F #0.0724


#Compare against the misclassification rate for the base model
error.boost.F.base <-
round(mean(train.purchase.m$lastQuoted_F[validSubset]!=train.purchase.m$F[validSubset]),4)
error.boost.F.base #0.0731


# Fit Metrics
confusionMatrix(post.valid.boost.F,train.purchase.m$F[validSubset]) #Kappa : 0.8951


#plot relative influence of variables
summaryBoost<-summaryBoost[order(summaryBoost$rel.inf,decreasing=FALSE),]
par(mar=c(3,10,3,3))
barplot(t(summaryBoost$rel.inf),names.arg = summaryBoost$var ,las=2,col="darkblue",main = "Relative
Influence",horiz=TRUE)




####################
# SVM *F*
####################
set.seed(1)



### Use this code to create a sample of the training data to fit a model   #PB
n <-dim(train.purchase.m[train.purchase.m$part=="train",])[1]
# repeatability of results
svm.sample <- sample(n, round(.25*n)) # randomly sample 25% test
train.purchase.m.svm<-train.purchase.m[train.purchase.m$part=="train",][svm.sample,]
dim(train.purchase.m.svm)



##was taking a Long time
# # We can perform cross-validation using tune() to select the best choice of
# # gamma and cost for an SVM with a radial kernel:
# set.seed(1)
# control <- tune.control(nrepeat = 5,cross = 5)
# tune.out = tune(
#   svm,F ~ (lastQuoted_F) + risk_factor_imp + car_age + car_value + cost + age_oldest + age_youngest +
day + shopping_pt + state +
#     Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4  ,
#   data = train.purchase.m.svm,
#   kernel = "linear",
#   ranges = list(cost = c(.01,.1,.5,1),
#     gamma = c(1)),
#   tunecontrol = control
# )
# summary(tune.out)
```

```
#Fit a linear SVM Model #Full model overfit - recently quoted did better
ptm <- proc.time() # Start the clock!
svmfit.F=svm(F ~ (lastQuoted_F) + #risk_factor_imp + car_age + car_value + cost + age_oldest +
age_youngest + day + shopping_pt + state +
               Quoted_F_minus2 + Quoted_F_minus3 + Quoted_F_minus4  ,
             data=train.purchase.m.svm,
             kernel="linear",
             gamma=.01,
             cost=1,
             probability =TRUE)
proc.time() - ptm # Stop the clock


#Summary statitics
summary(svmfit.F)


#RunTime
# user  system elapsed
# 29.89 0.39   30.40


# Predict SVM on validation set
post.train.svm.F<-predict(svmfit.F,train.purchase.m[trainSubset,])


post.valid.svm.F<-predict(svmfit.F,train.purchase.m[validSubset,])
length(post.valid.svm.F)


#Create a simple confusion matrix
table(post.valid.svm.F,train.purchase.m$F[validSubset])


#   0    1    2    3
#0 7590  106  185   44
#1   86 5333  303   34
#2  192  402 8425  225
#3   34   36  127 1130


#Check the misclassification rate
  ##training set
  round(mean(post.train.svm.F!=train.purchase.m$F[trainSubset]),4) #0.0724


  ##validation set
  error.svm.F <- round(mean(post.valid.svm.F!=train.purchase.m$F[validSubset]),4)
  error.svm.F # 0.0739


#Compare against the misclassification rate for the base model
error.svm.F.base <-
round(mean(train.purchase.m$lastQuoted_F[validSubset]!=train.purchase.m$F[validSubset]),4)
error.svm.F.base # 0.0731
  confusionMatrix(train.purchase.m$lastQuoted_F[validSubset],train.purchase.m$F[validSubset]) #Kappa:
0.8941




confusionMatrix(post.valid.svm.F,train.purchase.m$F[validSubset])
# Kappa : 0.8941


###################
# ## Data manipulation for test set ##
###################
#PB
```

```
#Baseline prediction using last quoted plan #.53793 on Kaggle #PB
test$planCombo<-paste(test$A,test$B,test$C,test$D,test$E,test$F,test$G,sep="")
#find the last shopping point for each customer
maxShopping_pt<-(aggregate(test$shopping_pt, by = list(test$customer_ID), max))
names(maxShopping_pt)<-c("customer_ID","lastquote")
lookup<-test[c("customer_ID","shopping_pt","planCombo")]


#SUBMIT File for baseline model - based on last quote
#merge each customer ID with their last quoted plan
lastQuoteSubmit<-
merge(x=maxShopping_pt,y=lookup,by.x=c("lastquote","customer_ID"),by.y=c("shopping_pt","customer_ID"))
names(lastQuoteSubmit)[names(lastQuoteSubmit)=="planCombo"]<-"lastQuotedPlan"
#clean up dataframe
lastQuoteSubmit$lastquote <- NULL
lastQuoteSubmit<-lastQuoteSubmit[order(lastQuoteSubmit$customer_ID),]
names(lastQuoteSubmit)<-c("customer_ID","plan")


#data frame of unique customers
test.m<-
merge(x=maxShopping_pt,y=test,by.x=c("lastquote","customer_ID"),by.y=c("shopping_pt","customer_ID"))
names(test.m)[names(test.m)=="planCombo"]<-"lastQuotedPlan"
test.m$shopping_pt<-test.m$lastquote


#Adding previous quoted plans to train.purchase.m data frame #PB
test.m$purchaseMinus_2<-test.m$lastquote-1
test.m$purchaseMinus_3<-test.m$lastquote-2
test.m$purchaseMinus_4<-test.m$lastquote-3


test.m<-
merge(x=test.m,y=lookup,by.x=c("purchaseMinus_2","customer_ID"),by.y=c("shopping_pt","customer_ID"),all.x=
TRUE)
names(test.m)[names(test.m)=='planCombo']<-"QuoteMinus_2"


test.m<-
merge(x=test.m,y=lookup,by.x=c("purchaseMinus_3","customer_ID"),by.y=c("shopping_pt","customer_ID"),all.x=
TRUE)
names(test.m)[names(test.m)=='planCombo']<-"QuoteMinus_3"


test.m<-
merge(x=test.m,y=lookup,by.x=c("purchaseMinus_4","customer_ID"),by.y=c("shopping_pt","customer_ID"),all.x=
TRUE)
names(test.m)[names(test.m)=='planCombo']<-"QuoteMinus_4"


#Cleaning up the test.m data frame
test.m$QuoteMinus_3[is.na(test.m$QuoteMinus_3)]<-"XXXXXXX"
test.m$QuoteMinus_4[is.na(test.m$QuoteMinus_4)]<-"XXXXXXX"
test.m<-test.m[order(test.m$customer_ID),]


#adding quoting history for each option to model data frame #PB
planOptions<-c("A","B","C","D","E","F","G")
for (ii in 1:7)  {
  test.m[paste("lastQuoted_",planOptions[ii],sep="")] <-
as.factor(substring(test.m$lastQuotedPlan,first=ii,last=ii))
  test.m[paste("Quoted_",planOptions[ii],"_minus2",sep="")] <-
as.factor(substring(test.m$QuoteMinus_2,first=ii,last=ii))
  test.m[paste("Quoted_",planOptions[ii],"_minus3",sep="")] <-
as.factor(substring(test.m$QuoteMinus_3,first=ii,last=ii))
  test.m[paste("Quoted_",planOptions[ii],"_minus4",sep="")] <-
as.factor(substring(test.m$QuoteMinus_4,first=ii,last=ii))
```

```
}


test.m<-test.m[order(test.m$customer_ID),]
head(test.m)



####################################
## Impute missing values ##
####################################
#Used decision tree to impute missing values for risk_factor.
#test.m$risk_factor <- test.m$risk_factor_imp
#test.m$C_previous <- test.m$C_previous_imp
#test.m$duration_previous <- test.m$duration_previous_imp
#test.m$location <- test.m$location_imp







#################
# PREDICT G on test set
#################


#Predict G on the test set using Random Forest #PB  0.53955
predictG <- predict(model.rf.G, test.m, type = "class")
#Predict G on the test set using Boosted Tree  #PB   0.54045
predictG <-predict(model.boost.G, test.m,type = 'response',n.trees = 1000)
predictG <- apply(predictG, 1, which.max)


predictResultsG<-data.frame(test.m$customer_ID,predictG)
test.m$predictG<-predictG


# hard coded rules
#all resdints in Florida should select 3 or 4, predictions of 2 change to 3, could use decision tree for
this #PB
predictResults$predictG[test.m$state=="FL" & test.m$predictG=="2"]<-as.factor("3")
#add option F - NY and CT rules


predict_Submit<-
merge(x=lastQuoteSubmit,y=predictResultsG,by.x=c("customer_ID"),by.y=c("test.m.customer_ID"))
head(predict_Submit)









#################
# PREDICT F on test set
#################


#Predict F on the test set using Random Forest #PB
# predictF <- predict(model.rf.F, test.m, type = "class")
#Predict F on the test set using Boosted Tree  #PB
```

```
predictF <-predict(model.boost.F, test.m,type = 'response',n.trees = 1000)
predictF <- apply(predictF, 1, which.max)-1


predictResultsF<-data.frame(test.m$customer_ID,predictF)


# hard coded rules
#add option F - NY and CT rules
predict_Submit<-
merge(x=predict_Submit,y=predictResultsF,by.x=c("customer_ID"),by.y=c("test.m.customer_ID"))
head(predict_Submit)


#replacing last quoted G with predicted G #PB
predict_Submit$predict.plan<-predictG_Submit$plan
predict_Submit$predict.plan<-
paste(substring(predict_Submit$predict.plan,first=1,last=6),predict_Submit$predictG,sep="")
head(predict_Submit)


# #replacing last quoted F with predicted F #PB
# predict_Submit$predict.plan<-
paste(substring(predict_Submit$predict.plan,first=1,last=5),predict_Submit$predictF,substring(predict_Subm
it$predict.plan,first=7,last=7),sep="")
# head(predict_Submit)



#view sample before exporting to csv #PB
predict_Submit_Final<-(predict_Submit[c('customer_ID','predict.plan')])
names(predict_Submit_Final)<- c('customer_ID','plan')
head(predict_Submit_Final)
write.csv(predict_Submit_Final, file=file.path(path,"submit_GBM_FG_2.csv"), row.names=FALSE, quote=FALSE)
```