FOREST COVER TYPE
TEAM 1

Objective

The objective of this analysis is to identify a set of predictive machine learning models to classify forest cover types into seven categories using a set of cartographic variables.

Authors:

DellaGrotte, Paul
Gyampo, Rose
Hughes, Derek
Park, Sung
Rassas, Adeline

Forest Cover Type Project

I. Introduction

The objective of this project is to develop a suite of machine learning models to estimate the classification of forest cover type. As a pure prediction problem in the military/defense scenario, we analyze data that represent forest cover types for 30x30 meter cells obtained from U.S. Forest Service (USFS) Region 2 Resource Information System (RIS). In this analysis, forest cover types are classified into seven distinct categories. We employ a set of cartographic attributes, soil types, and area designations to build a suite of statistical learning models to estimate the classification of forest cover types, thus leading into a multiclass classification problem.

In order to predict forest cover classification, we conduct a model-based exploratory data analysis (EDA) to gain an initial understanding of the explanatory variables and their associations with forest cover type class. First, using descriptive exploratory techniques, we discover pairs of quantitative variables that are correlated using principal component analysis (PCA). Using model-based EDA, we discover that ELEVATION is the most important variable for classifying COVERTYPE using a recursive decision tree. Using the random forest method, ELEVATION is also the most important.

In this analysis, we examine the following statistical learning models: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), K-nearest neighbors (KNN), random forest (RF), gradient boosting (GBM), support vector machine (SVM), and neural network (NNET). Each model type is examined using different algorithmic parameters (e.g. K-nearest neighbor values, tuning parameters, number of hidden neurons, and kernels) to identify the model that best predicts the classification of COVERTYPE. Each model is compared using test

misclassification rates (e.g. confusion matrices). The model that exhibits the lowest misclassification rate on unseen data is deemed the best.

Our best machine learning model is the random forest using an "*improve*" parameter of 0.05, which is the relative improvement in out-of-bag error for the search to continue. This model yields the highest correct classification rate of 95.52% on unseen data. Using this parameter as the tuning method, this random forest considered six predictors at each split instead of four predictors at each split.

In this paper, we state our modeling problem, followed by the description of our modeling dataset. We proceed by highlighting the most interesting statistical features of our modeling problem using a model-based EDA recursive decision tree. Further, we discuss the backward variable selection technique and the reduction of variables using principal component analysis (PCA). The predictive modeling section contains details about statistical learning models for linear discriminant analysis, quadratic discriminant analysis, K-nearest neighbors, random forest, gradient boosting, support vector machines, and neural network. Using statistical learning models and appropriate analytical methods, we compare the predictive performance on unseen (test) data and conclude our findings.

II. Modeling Problem

This statistical learning task is a multiclass classification problem. In order to predict the classification of forest cover types, we examine a set of cartographic attributes, soil types, and area designations. Forest cover types are represented using seven categories: (1) spruce/fir, (2) lodgepole pine, (3) ponderosa pine, (4) cottonwood/willow, (5) aspen, (6) douglas/fir, and (7) krummholz. Using data generated from the U.S. Geological Survey (USGS), we examine the

statistical relationships of collected variables with the classification outcome using exploratory data analysis (EDA), model-based EDA, and principal component analysis (PCA).

The overarching objective of this project is to develop an accurate statistical learning model to predict the forest cover type for military/defense purposes. The initial purpose of this dataset was to study natural resources for developing ecosystem management strategies (Blackard, 1999). However, additional application can consist of developing an accurate model for U.S. military with the goal of deploying strategic defense programs to better protect the U.S. border.

We investigate the following statistical models: (1) linear discriminant analysis, (2) quadratic discriminant analysis, (3) K-nearest neighbors, (4) random forest, (5) gradient boosting, (6) support vector machine, and (7) neural network. These modeling algorithms are the most relevant techniques for our classification problem because we are working with a large dataset that contains a variety of variables (quantitative and binary) and we seek to classify forest cover types. These algorithms are designed to minimize a loss function (e.g. misclassification rate or mean squared error) using computational optimizations for mixed datasets.

For linear discriminant analysis and quadratic discriminant analysis, quantitative explanatory variables are required to exhibit a normal distribution (Pohar, Blas & Turk, 2004). By examining the boxplot distributions of quantitative variables (see appendix), we conclude that these variables follow a fairly normal distribution, with some skewness due to the presence of extreme values. In essence, linear discriminant analysis makes more assumptions about the underlying data and works better when the normality assumptions are met (Pohar, Blas & Turk, 2004). Linear discriminant analysis, along with logistic regression, have been originally designed for classification problems. These models are estimated using maximum likelihood method. We

investigate the predictive power of linear discriminant analysis and quadratic discriminant analysis in order to evaluate their performance with traditional machine learning models.

In order to become more effective in working with complex non-linear relationships of quantitative and binary explanatory cartographic variables, we employ machine learning models. In this environment, the normality assumption is not as stringent and the overarching objective is to obtain the most accurate results on unseen data instead of making statistical inference. In essence, statistical learning models are most appropriate when we seek accurate predictions of complex, non-linear relationships of explanatory variables. The objective of statistical learning models is to increase predictive accuracy, which takes place inside a "black box." As a result, statistical inference is more challenging.

For our study, our aim is to build statistical learning models that can predict accurately the forest cover type for defense purposes and natural resource management for any private, state, or federal land management agency (Blackard, 1999). Statistical learning models can achieve high bias-high variance in the prediction process, which leads to the danger of overfitting. As a result, we employ training and testing validation process to evaluate predictive performance on unseen data.

K-nearest neighbor is defined as a lazy learning algorithm since it does not produce any specific model parameters (Lesmeister, p. 106). K-NN classifies elements by looking at the closest resemblance among training elements to determine the proper class. By defining an optimal K, which represents the number of K-nearest points, this technique identifies groups (Lesmeister, p. 106). K-NN technique employs the Euclidean Distance to identify closest points. In order to identify these groups, we scale all quantitative variables so that they have a mean of zero and a standard deviation of one. This step standardizes quantitative variables so that all

variables are on the same scale in order to prevent significant influences from large values on forest classification. Using identified groups from K-NN, we compare the predicted class with the actual forest cover type class using a confusion matrix.

Random forest is a form of ensemble learning model. In essence, it utilizes the classification results voted from many classification trees. The random forest can grow hundreds or thousands of trees (as we defined in R) and in this process, generates individual classification results that are then utilized for majority voting (e.g. classification problem). In particular, the majority votes of all decision trees are used to classify the data. Random forest is meant to ensemble weak learners (single decision trees) into a strong learner (Chiu, p. 278). Hence, we can identify the most important variables that are used to generate individual trees.

Similar to random forest, gradient boosting works as a voting algorithm. This algorithm begins with a base learner and grows additional branches in sequence, where each successive learner is built for the prediction residuals of the preceding learner (Chiu, p. 252). In other words, using residuals from previous base learners, the algorithm builds the next base learners that maximally correlates with the gradient of the loss function (e.g. misclassification) (Chiu, p.262). In this application, we employ gradient boosting in order to evaluate its predictive power compared to other machine learning models, especially with random forest since both techniques are ensemble models.

Unlike random forest and gradient boosting methods, support vector machine and neural network are considered “black box” models. It is less apparent how each individual attribute is evaluated and employed to generate predictions. Technically, support vector machines employ *kernels* (e.g. linear, polynomial, radial, or sigmoid) to find the optimal hyperplanes that separate elements into classes (Chiu, p. 188). The effectiveness of these hyperplanes is evaluated using

margins, which are the distances (space) between classified groups. Ideally, we seek hyperplanes whose margins are the largest amongst all possible separating hyperplanes (Forte, p. 166). Support vector machines are difficult to interpret since it is not apparent how predictors are evaluated to generate hyperplanes.

Finally, neural network models are also complex to interpret. These models can be represented as interconnected group of hidden nodes, as the human brain cells. In essence, brain cells fire information from one neuron to another. Neural network works in a similar manner. This model type consists of inputs, weights, and outputs. The strength of inputs are weights (Chiu, p. 207). Weights greater than zero generate an "excitation" status. Otherwise, it is an "inhabitation" status (Chiu, p. 207). The higher the weight, the stronger the activation of the input. Neural network can contain hidden nodes, which are used to sum up weights and modify these sums to minimize the misclassification rate (using a cost function). In other words, neural network can "learn" from various inputs in each neuron and attempt to minimize misclassification rates with new summed weights from activation nodes. The advantage of neural network is its ability to detect non-linear relationships between the target and explanatory variables (Chiu, p. 208). The choice of the number of hidden layers is through trial and error. In this project, we utilize resilient backpropagation learning algorithm with one hidden layer (since it is the most common method (Blackard, 1999)) and modify the learning rate of the neural network algorithm to better understand the predictive performance. Due to complex nature of this statistical learning, the predictive performance on unseen data is evaluated using confusion matrices. Human interpretation of neural network is nearly impossible. However, we can examine the weights of each neuron from one layer to the next.

The target variable, COVERTYPE, takes seven possible nominal values. In order to estimate the proposed machine learning models, datatype transformation is applied. For LDA, QDA, and K-NN models, the nominal value of COVERTYPE is retained using *as.factor()* function. Random forest, gradient boosting, and support vector machine also take COVERTYPE as a factor to estimate the final class. Unlike previous models, neural network requires COVERTYPE to be transformed into separate numeric binary target attributes (1/0).

In order to evaluate the effectiveness of statistical learning models, we generate confusion matrices and compute test MSE (or misclassification rate) on unseen dataset. In general, we select the statistical learning model with the lowest test MSE. For all modeling stages, we split the original dataset into training/testing datasets. We split 70% of the original data as "train" and 30% as "test" to evaluate the test MSE on unseen data. In order to train our models using available computing resources, we sample 2.5% of the 70% training set.

III. Data

These data represent a forest located in *Roosevelt National Forest* in northern Colorado, which is approximately 70 miles northwest of Denver, Colorado (Blackard, 1999). These attributes represent environmental and geographical characteristics of the forest cover type. These data are collected from the U.S. Geological Survey (USGS) and U.S. Forest Service (USFS) Region 2 Resource Information System (RIS). The modeling dataset contains 581,012 observations with a total of 54 attributes. These data consist of 12 measures. There are 10 quantitative variables, 4 binary wilderness areas, and 40 binary soil type variables.
In table 1, we summarize the characteristics of the modeling dataset.

Table 1: Data Definition

Variable	Data Type	Description
Elevation	Quantitative	Elevation in meters
Aspect	Quantitative	Aspect in degrees azimuth
Slope	Quantitative	Slope in degrees
Horizontal Distance to Hydrology	Quantitative	Horz distance to nearest surface water features
Vertical Distance to Hydrology	Quantitative	Vert distance to nearest surface water features
Horizontal Distance to Roadways	Quantitative	Horz distance to nearest roadway
Hillshade 9am	Quantitative	Hill shade index at 9am, summer solstice
Hillshade Noon	Quantitative	Hill shade index at noon, summer solstice
Hillshade 3pm	Quantitative	Hill shade index at 3pm, summer solstice
Horizontal Distance to Fire Points	Quantitative	Horz distance to nearest wildfire ignition points
Wilderness Area (4 binary columns)	Qualitative	Wilderness area designation 1 – Rawah Wilderness Area 2 – Neota Wildnerness Area 3 – Comanche Peak Wilderness Area 4 – Cache la Poudre Wilderness Area
Soil Type (40 binary columns)	Qualitative	Soil type designation USFS Ecological Land Type Units
Cover type (Target) (7 types)	Integer	Forest cover type designation 1 – Spruce/Fir 2 – Lodgepole Pine 3 – Ponderosa Pine 4 – Cottonwood/Willow 5 – Aspen 6 – Douglas/Fir 7 – Krummholz

Further, we display the class distribution of the target COVERTYPE variable in table 2.

Table 2: Forest Cover Type Frequency and Proportion

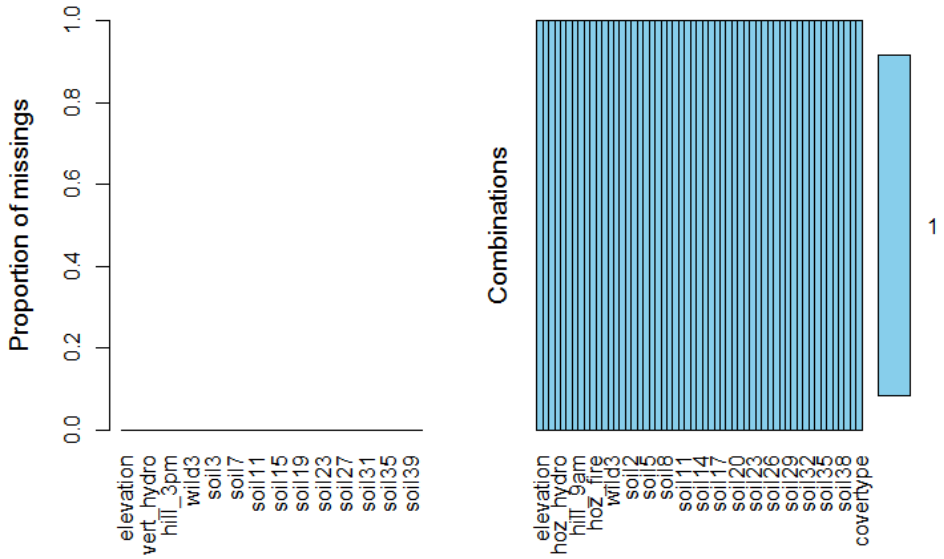
Cover Type	Frequency	Proportion
Spruce/Fir	211,840	36.46%
Lodgepole Pine	283,301	48.76%
Ponderosa Pine	35,754	6.15%
Cottonwood/Willow	2,747	0.47%
Aspen	9,493	1.63%
Douglas/Fir	17,367	2.99%
Krummholz	20,510	3.53%
Total	**581,012**	**100%**

We conclude that lodgepole pine exhibits the highest proportion with 48%, followed by spruce/fir with 36%, and ponderosa pine with 6%. We observe an imbalance among the seven classes, which will affect the classification performance of our machine learning models.

IV. Exploratory Data Analysis

In this section, we conduct exploratory data analysis (EDA) to uncover interesting statistical relationships with COVERTYPE. First, we examine the presence of missing values using *VIM()* package. This dataset does not contain any missing values, as indicated in figure 1.

Figure 1: Visual Representation of Missing Values



a. Traditional EDA

In this section, we summarize frequency for qualitative variables (table 3) and present the overall distributions of quantitative variables (table 4).

For wilderness area, area1 makes up the greatest proportion with 45%, followed by area3 with 44%. Soiltype10 dominates all other soil types with 38%. Overall, we observe a very sparse

distribution across wilderness areas and soil types, which will introduce significant noise into model estimations.

Table 3: Table of Distributions – Qualitative Variables

Variable	Yes (1)	No (0)	Proportion Yes (1)
Wilderness Area 1	260796	320216	44.89%
Wilderness Area 2	29884	551128	5.14%
Wilderness Area 3	253364	327648	43.61%
Wilderness Area 4	36968	544044	6.36%
Soil Type 1	3031	577981	0.52%
Soil Type 2	7525	573487	1.30%
Soil Type 3	4823	576189	0.83%
Soil Type 4	12396	568616	2.13%
Soil Type 5	1597	579415	0.27%
Soil Type 6	6575	574437	1.13%
Soil Type 7	105	580907	0.02%
Soil Type 8	179	580833	0.03%
Soil Type 9	1147	579865	0.20%
Soil Type 10	332634	548378	37.76%
Soil Type 11	12410	568602	2.14%
Soil Type 12	29971	551041	5.16%
Soil Type 13	17431	563581	3.00%
Soil Type 14	599	580413	0.10%
Soil Type 15	3	581009	0.00%
Soil Type 16	2845	578167	0.49%
Soil Type 17	3422	577590	0.59%
Soil Type 18	1899	579113	0.33%
Soil Type 19	4021	576991	0.69%
Soil Type 20	9259	571753	1.59%
Soil Type 21	838	580174	0.14%
Soil Type 22	33373	547639	5.74%
Soil Type 23	57752	523260	9.94%
Soil Type 24	21278	559734	3.66%
Soil Type 25	474	580538	0.08%
Soil Type 26	2589	578423	0.45%
Soil Type 27	1086	579926	0.19%

Variable	Yes (1)	No (0)	Proportion Yes (1)
Soil Type 28	946	580066	0.16%
Soil Type 29	115247	465765	19.84%
Soil Type 30	30170	550842	5.19%
Soil Type 31	25666	555346	4.42%
Soil Type 32	52519	528493	9.04%
Soil Type 33	45154	535858	7.77%
Soil Type 34	1611	579401	0.28%
Soil Type 35	1891	579121	0.33%
Soil Type 36	119	580893	0.02%
Soil Type 37	298	580714	0.05%
Soil Type 38	15573	565439	2.68%
Soil Type 39	13806	567206	2.38%
Soil Type 40	8750	572262	1.51%

In general, all quantitative variables appear to be centered at the mean and median since both values are very similar. However, ASPECT, HOZ_HYDRO, VERT_HYDRO, HOZ_ROAD, and HOZ_FIRE appear to be right-skewed, as indicated by their higher means compared to their medians. All quantitative variables contain outliers (as indicated by boxplots), which will affect the predictive performance of our machine learning models. Values that are at least two standard deviations away from the mean are considered outliers. Graphical distributions for all quantitative variables appear in the appendix.

Table 4: Table of Distributions – Quantitative Variables

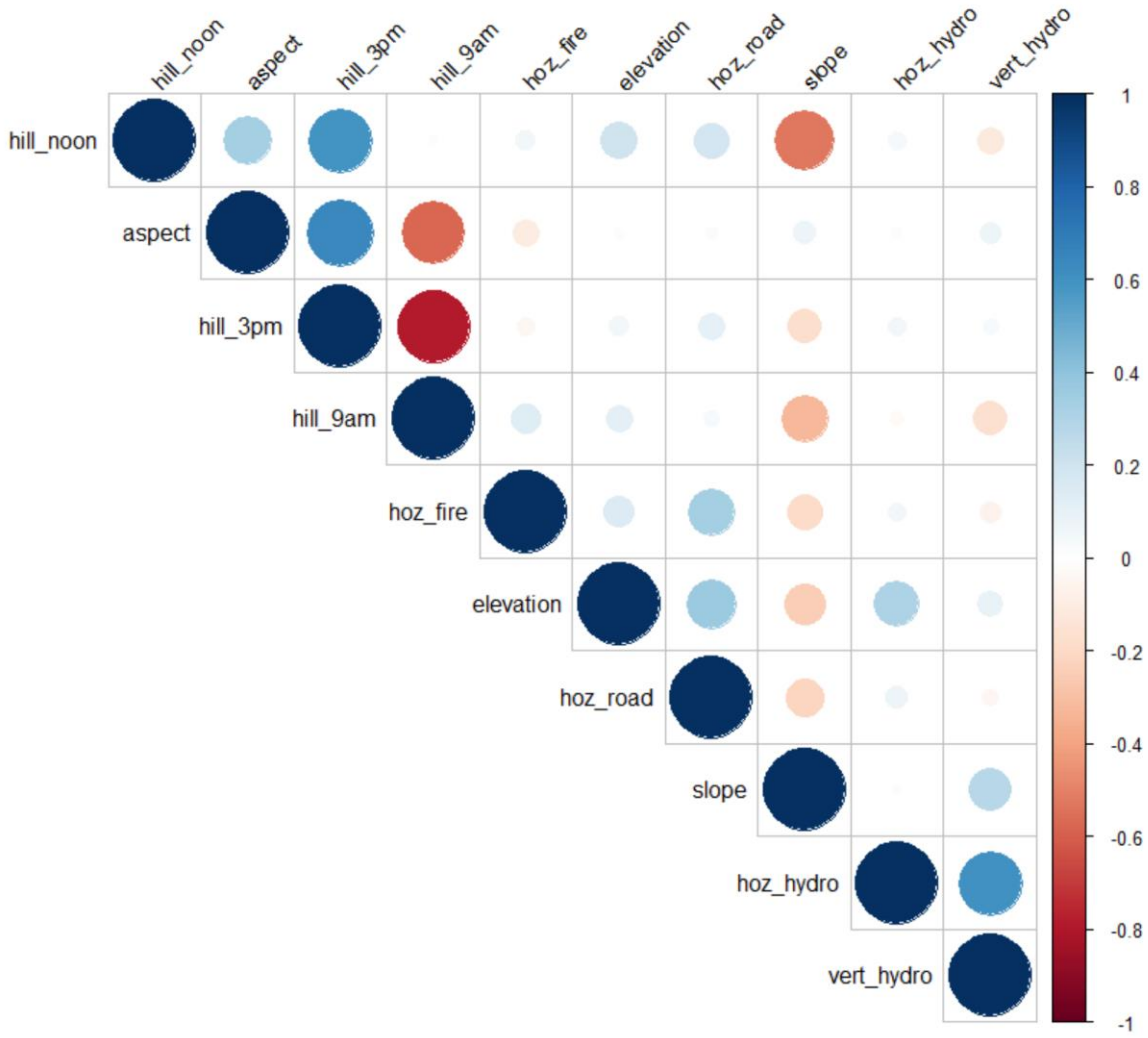
Variable	Minimum	25th Percent	50th Percent	75th Percent	Maximum	Mean
Elevation	1859	2809	2996	3163	3858	2959
Aspect	0	58	127	260	360	155.7
Slope	0	9	16	18	66	14.1
Horizontal Distance to Hydrology	0	108	218	384	1397	269.4
Vertical Distance to Hydrology	-173	7	30	69	601	46.42
Horizontal Distance to Roadways	0	1106	1997	3328	7117	2350
Hillshade 9am	0	198	218	231	254	212.10
Hillshade Noon	0	213	226	237	254	223.30

Variable	Minimum	25th Percent	50th Percent	75th Percent	Maximum	Mean
Hillshade 3pm	0	119	143	168	254	142.5
Horizontal Distance to Fire Points	0	1024	1710	2550	7173	1980

All quantitative variables appear to be legitimate. The negative value of "Vertical Distance to Hydrology" is also legitimate since it is a measure of vertical distance to nearest surface water features. A negative distance indicates a topographic feature below sea level.

In addition to traditional EDA using distribution statistics, we investigate the correlations of quantitative variables. Figure 2 displays these correlations and shows that HILLSIDE_3PM and HILLSIDE_NOON are strongly positively correlated. On the other hand, HILLSIDE_NOON and SLOPE are strongly negatively correlated. In addition, ASPECT and HILLSIDE_3PM are strongly positively correlated. HORIZONTAL_HYDRO and VERTICAL_HYDRO are also positively correlated. While we identified positively correlated pairs, HILLSIDE_3PM and HILLSIDE_9AM are strongly negatively correlated. In all, this correlation analysis reveals evidence of correlated quantitative variables, thus indicating the presence of multicollinearity. However, in theory, multicollinearity does not affect predictive performance, but will produce incorrect coefficients using conventional multinomial logistic regression.

Figure 2: Plot of Correlations for Quantitative Variables

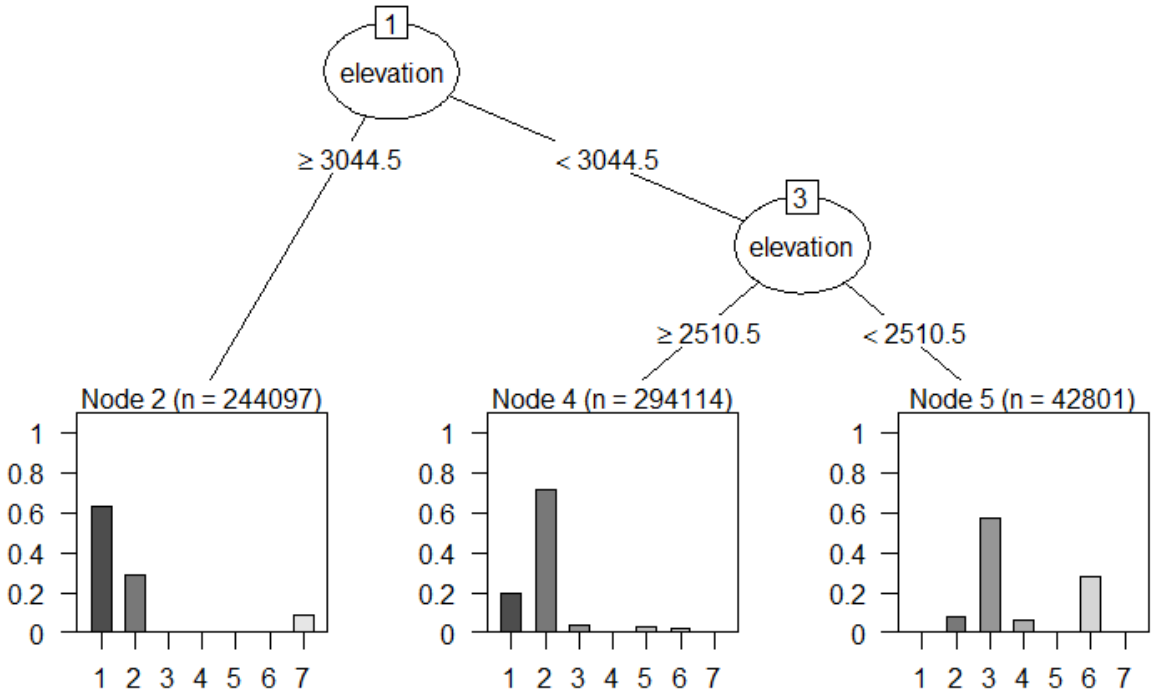


b. Model-Based EDA

In this section, we investigate the associative relationship of all exploratory variables with COVERTYPE. Using a recursive decision tree for EDA purpose (*rpart* package), we discover that the most important variable is ELEVATION. The variable importance is 59, followed by Wild4 with 17. Detailed results appear in the appendix. In figure 3, the tree diagram is presented, showing ELEVATION as the most important variable. It is clear that for elevation

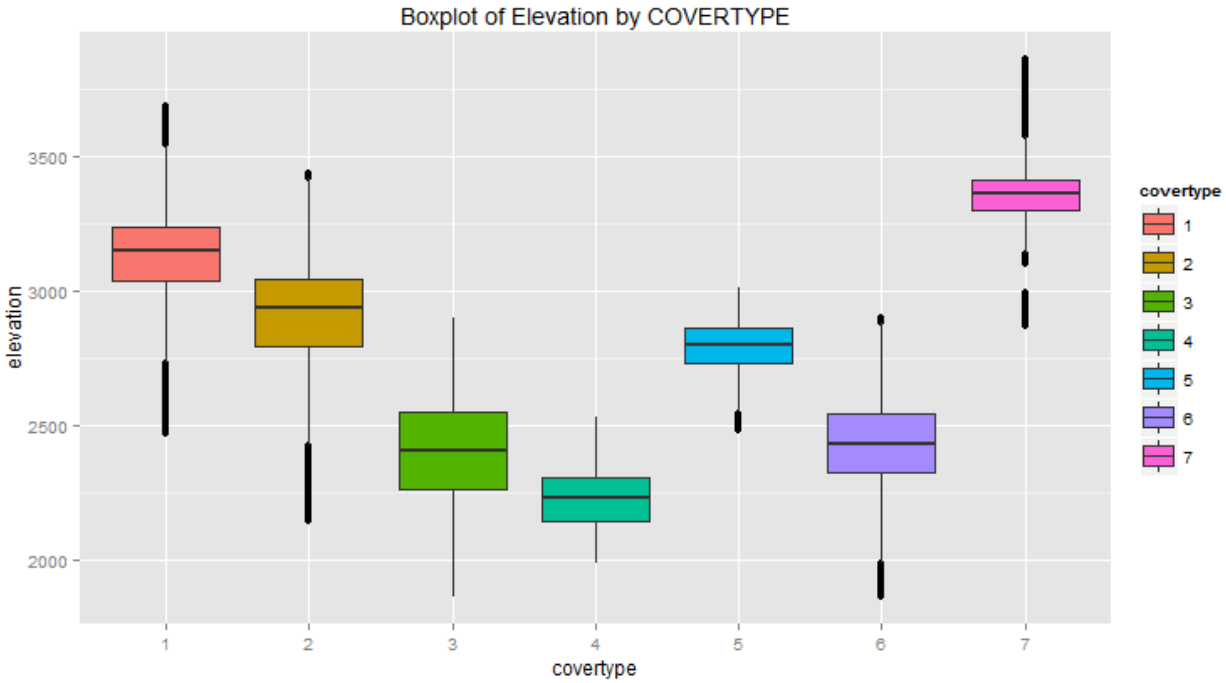
>= 3,044.50 meters, the forest cover type is primarily in category 1 When elevation < 3,044.50 meters, the cover type is category 2 and above.

Figure 3: Tree Diagram



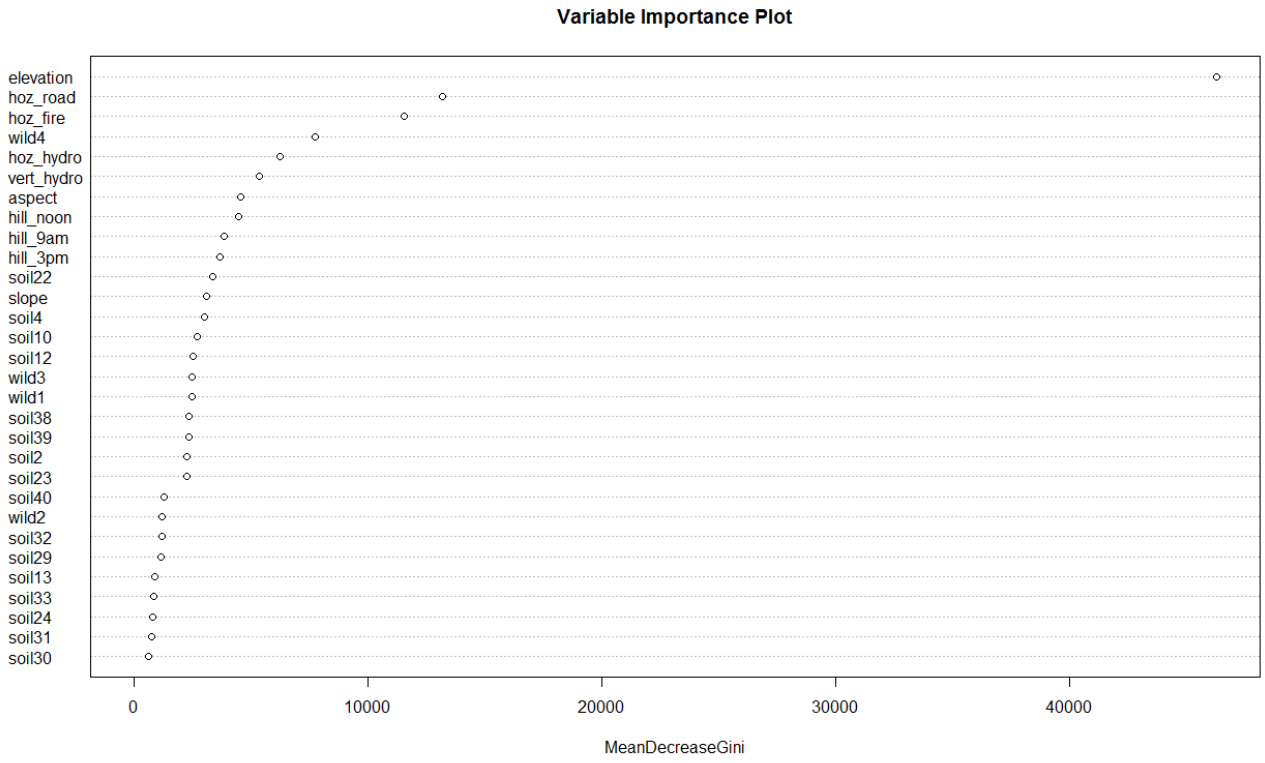
Further, we examine individual distributions of ELEVATION by COVERTYPE in figure 4. Clearly, mean ELEVATION is the highest for COVERTYPE 7, followed by COVERTYPE 1 and COVERTYPE 2. All distributions are slightly skewed to the left, as indicated by outliers at the lower percentiles.

Figure 4: Boxplot of ELEVATION BY COVERTYPE



In addition to recursive decision tree, we employ random forest to identify the most important variables for classifying COVERTYPE. We observe that ELEVATION, HOZ_ROAD, and HOZ_FIRE are the top three most important variables.

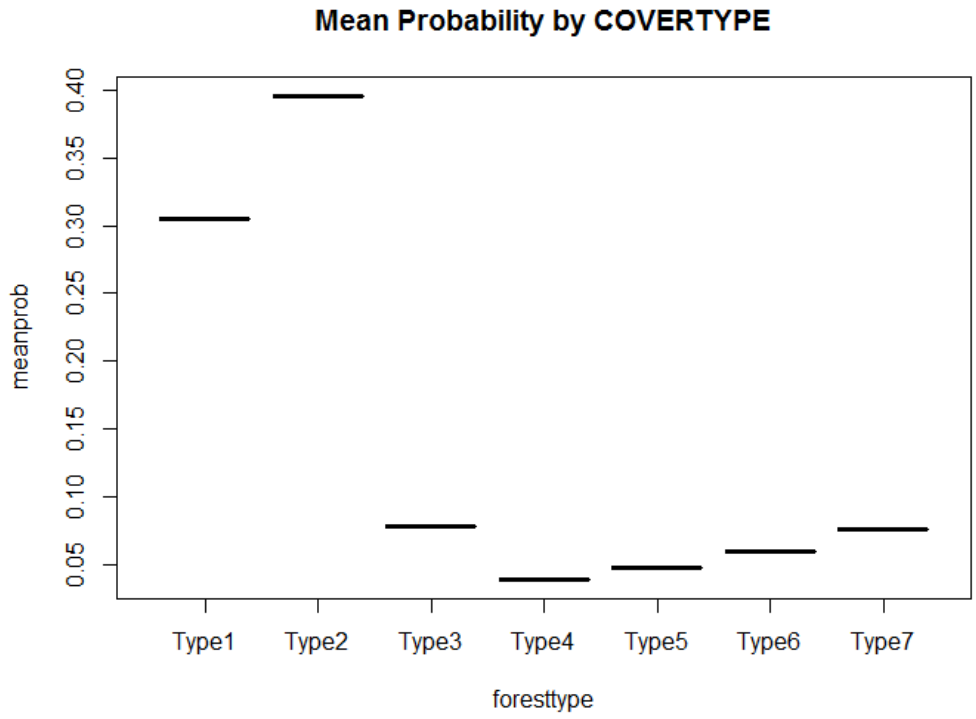
Figure 5: Variable Importance Plot



In this section, we uncovered the importance of ELEVATION in classifying COVERTYPE using model-based EDA recursive decision tree. In addition to ELEVATION, the remaining quantitative variables exhibit importance, as indicated by the random forest-based EDA. Hence, we further evaluate their importance in the "Variable Selection" section. As evidenced by this random forest, wilderness area and soil type do not contribute as much importance as quantitative variables. Hence, they are not used as explanatory variables in our models.

In addition to decision trees, we estimate class probabilities using multinomial logistic regression using all explanatory variables. Figure 6 displays the mean predicted probabilities by COVERTYPE.

Figure 6: Plot of Mean Probability by COVERTYPE



Clearly, the greatest probability of being classified as Type 2 is the highest with 40%, followed by Type 1 with 30%. The lowest probability occurs for Type 4 with less than 5%.

Variable Selection and Reduction

In this section, we further evaluate the importance of quantitative variables using a naïve statistical model (backward selection method).

A. Using Backward Selection:

By only considering quantitative variables, we employ backward selection to confirm their importance.

The following seven variables exhibit importance:

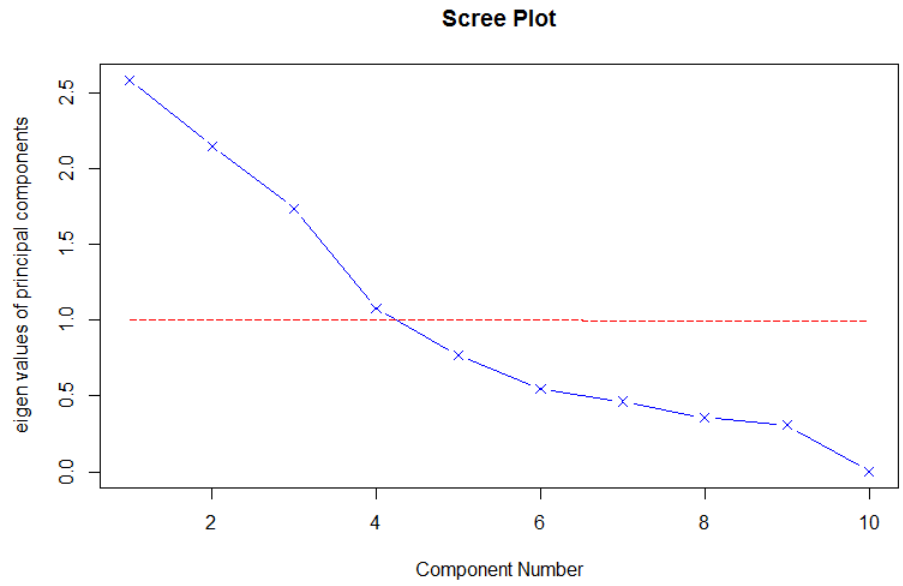
- Elevation
- Vertical Distance to Hydrology
- Horizontal Distance to Fire
- Hillside 3PM
- Aspect
- Slope
- Horizontal Distance to Road

Details about the backward selection results appear in the appendix. The variables are further evaluated using principal component analysis to reduce the number of quantitative variables and remove multicollinearity.

B. Using Principal Component Analysis:

In order to remove multicollinearity, we employ principal component analysis (PCA) to extract components that represent the largest proportion of the variance. This technique is employed to reduce the number of variables and identify PCA components for "input" explanatory variables into our machine learning models. By evaluating quantitative variables, we conclude that four principal components (see appendix) are needed to represent the greatest proportion of the variance. These components are extracted using varimax (orthogonal) rotation. We leverage the scree plot to identify the optimal number of components.

Figure 7: Scree Plot



Using VARIMAX rotation, we obtain the following components:

- PC1: High loadings on ASPECT and HILL_3PM.
- PC2: High loadings on HILL_NOON.
- PC3: High loadings on HOZ_HYDRO and VERT_HYDRO.
- PC4: High loadings on HOZ_ROAD.

Using extracted components, machine learning models are built.

V. Predictive Modeling: Methods and Results

The modeling task consists of splitting the original dataset into training and testing samples. For instance, we split 70% of the original dataset into "training" and 30% as "testing". For computational purposes, we sample 2.5% of the 70% of the training dataset for building our models. Using the *set.seed()* function, we ensure the reproducibility of results. The training dataset contains 406,710 observations and the test dataset contains 174,302 observations. The

sampled training dataset contains 10,171 observations. The following table displays the frequency of each COVERTYPE.

Table 5: Distribution of Training, Sampled Training and Testing Datasets

	TRAINING (70%)		SAMPLED TRAINING (2.5%)		TESTING (30%)	
COVERTYPE	**Frequency**	**Proportion**	**Frequency**	**Proportion**	**Frequency**	**Proportion**
1	148,288	34.46%	3,708	34.46%	63,552	36.46%
2	198,311	48.76%	4,958	48.76%	84,990	48.76%
3	25,028	6.15%	626	6.15%	10,726	6.15%
4	1,923	0.47%	49	0.47%	824	0.47%
5	6,646	1.63%	167	1.63%	2,847	1.63%
6	12,157	2.99%	304	2.99%	5,210	2.99%
7	14,357	3.53%	359	3.53%	6,153	3.53%
Total	**406,710**	**100%**	**10,171**	**100%**	**174,302**	**100%**

In the following sections, we present the following models: (1) linear discriminant analysis, (2) quadratic discriminant analysis, (3) K-nearest neighbors, (4) random forest, (5) gradient boosting, (6) support vector machine, and (7) neural network. Using various tuning parameters, kernels, shrinkage values, and learning rates, we exploit the predictive performance of these models. Table 6 lists examined models.

Table 6: List of Models

Model Letter	Model Type	Algorithmic Parameter	R Package
A	Linear Discriminant	Maximum Likelihood	MASS
B	Quadratic Discriminant	Maximum Likelihood	MASS
C	K-Nearest Neighbors	K = 1	CLASS
D	Random Forest	Default Setting	randomForest
E	Random Forest	"*improve*" parameter [improve=0.05] (relative improvement in OOB error)	randomForest tuneRF()
F	Gradient Boosting	Optimal Shrinkage Value	GBM
G	Support Vector Machine	Linear Kernel	e1071
H	Support Vector Machine	Radial Kernel	e1071
I	Neural Network	Learning Rate = 0.05, number of nodes = 2	neuralnet
J	Neural Network	Learning Rate = 0.35, number of nodes = 2	neuralnet

A. Linear Discriminant Analysis (LDA)

The goal of LDA is to find the best linear combination of the variables that will most accurately classify the forest cover type. LDA may be more generalizable and perform better when the classes are largely separated compared to Logistic Regression. This is because the confidence intervals for the coefficients can be very large and therefore create large variations in individual predictions. However, because it is linear based, if there are complex relationships between the features and the response variables, LDA may perform poorly.

We applied the algorithm with the seven variables chosen by the backward variable selection method to classify the forest cover type. Fitting the seven variables into the LDA model using the training set, we achieved an accuracy of 0.6695. This was intriguing because we achieved a better accuracy of 0.6713 with our test data. Considering we have rare case of using less training data than we have testing data, that could be a reason for the anomaly, as the training results are almost always better.

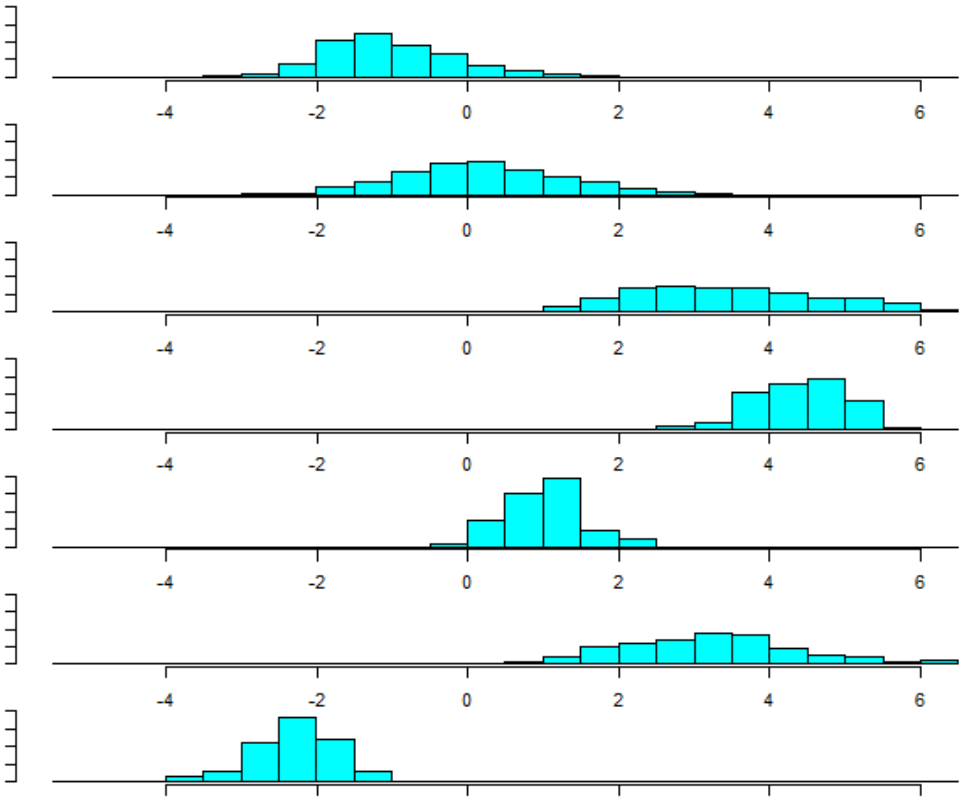
Out-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	25.53%	10.88%			0.01%		3.23%
2	10.66%	36.80%	1.44%		1.62%	0.81%	0.02%
3		1.03%	4.46%	0.45%		2.15%	
4			0.15%	0.02%		0.01%	
5						0.00%	
6		0.05%	0.11%			0.03%	
7	0.27%						0.29%

Model	**In-Sample MSE**	**Out-Sample MSE**	**Test Accuracy**
LDA	33.95%	32.87%	67.13%

As a comparison using all sixty-three variables our accuracy rate only improved marginally to 0.6822, thus indicating that our linear discriminant components did a very good job of identifying the most significant variables. This also indicates that the addition of an extra 56 variables provided only a 1% increase in accuracy. These additional variables provide very little to the model other than excessive noise and, therefore, in the spirit of parsimony we recommend the seven variable model with an accuracy of 0.6713.

Figure 8: Separation of Forest Covertype for Linear Discriminant (Overlapping Areas Show Areas of Misclassification)



B. Quadratic Discriminant Analysis (QDA)

QDA has a similar goal as LDA but it uses quadratic combinations of the variables to most accurately classify the forest cover type. As opposed to LDA, it can handle more complex relationships between the variables and the response variable. It does by assuming that each class has its own individual covariance, while LDA assumes a common covariance over all classes. As usual, though, while QDA is a more flexible model that can achieve lower bias, this often means it may have higher variance on test data. This issue can be somewhat reduced with large, robust training sets.

We applied the algorithm with the seven variables chosen by the backward variable selection method to classify the forest cover type. Fitting the seven variables into the QDA model using the training set, we achieved an accuracy of 0.6633.

Out-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	26.27%	12.69%					2.27%
2	8.89%	33.00%	0.92%		1.25%	0.60%	0.02%
3	0.05%	1.90%	4.66%	0.21%	0.03%	1.73%	
4		0.00%	0.07%	0.21%	0.00%	0.03%	
5	0.10%	0.62%	0.00%	0.00%	0.31%		
6	0.01%	0.40%	0.50%	0.05%	0.04%	0.62%	
7	1.15%	0.13%					1.23%

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
QDA	33.67%	33.69%	66.31%

With LDA we actually achieved a higher accuracy on the test set. QDA was close to achieving the same with an accuracy of 0.6631 on the test set. This shows that the test data set's

variable relationships to the output are closer to linear than non-linear relationships. Otherwise, QDA would have outperformed LDA.

Table 7: Train and test accuracy for LDA and QDA

Model	# Variables	Formula	Train Accuracy	Test Accuracy
Discriminant Analysis Models				
LDA	62	covertype~	67.99%	68.22%
LDA	7	covertype~bkwd selected variables	66.95%	67.13%
QDA	7	covertype~bkwd selected variables	66.33%	66.31%

C. K-Nearest Neighbor (KNN)

We ran K-nearest neighbor algorithm with a number of different K-values. KNN scores an example by averaging the "K" nearest outcomes based on their Euclidean distance from the example. K values represent the level of smoothness of the model and, as such, can affect the model's performance if not chosen correctly. For example, K=1 offers large flexibility to capture complex non-linear boundaries, but can underperform if not tuned properly. Hence we tested numerous K-nearest neighbor modes using various values of K.

A general rule of thumb is to use a K-value that equivalent to the square root of the number of observations. We had 174,302 observations which equates to a K-value of 417. This k-value achieved a mean accuracy rate of 0.7002272.

Another good starting point for K is to see 10 positive examples in each neighborhood so your model can express rates smaller than the baseline rate to some precision. We have 7 covertypes so each neighborhood has a prior probability rate of 1/7 = 14.3%. Therefore, 10

positive examples for each neighborhood (10/.143) is a k-value of 70. This approach achieved a mean accuracy rate of 0.770605, an improvement over k=417.

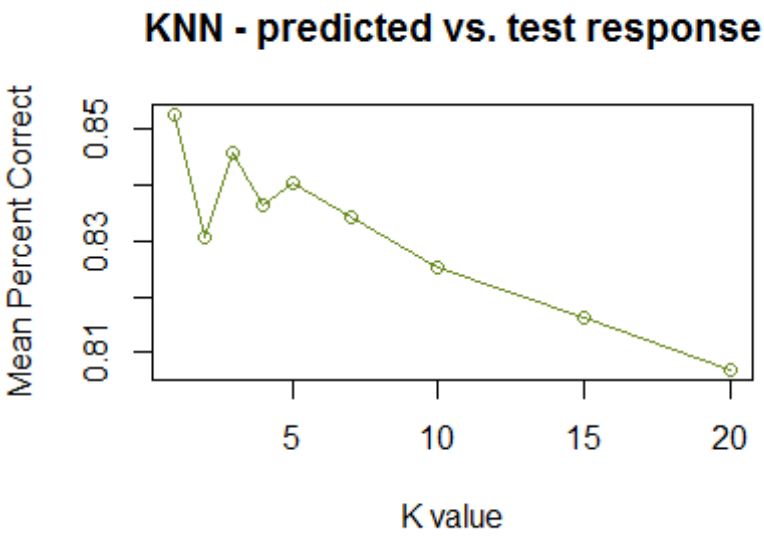
Table 8: K-NN Results

Model	# Variables	K-value	Train Accuracy	Test Accuracy
KNN	1	1	0.697411	0.852549
KNN	2	2	0.703636	0.830656
KNN	3	3	0.697411	0.845905
KNN	4	4		0.836307
KNN	5	5		0.840312
KNN	7	7		0.834437
KNN	10	10		0.825435
KNN	15	15		0.816279
KNN	20	20		0.807053

Seeing a trend of improving accuracy with lower k-values, this indicates there may be some complex non-linear boundaries between categories in our data set. We ran models with k-values of 1, 2, 3, 4, 5, 7, 10, 15, and 20.

Here we achieved the highest predictive accuracy of any k-values. K=1 had the highest accuracy rate of 0.8525 followed by K=3 at 0.8459 with the remaining trend decreasing accuracy with higher k-values. It is often noted that using K=1 values can have low bias but very high variance with small data sets. However, considering our data set had 174,302 observations we feel that using K=1 is appropriate.

Figure 9: K-Nearest Neighbors Results by Model's K-value



Out-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	31.32%	4.72%	0.01%		0.07%	0.02%	0.45%
2	4.64%	42.62%	0.48%		0.49%	0.30%	0.08%
3	0.01%	0.47%	4.95%	0.11%	0.03%	0.58%	
4			0.11%	0.31%		0.05%	
5	0.07%	0.54%	0.03%	0.00%	1.03%	0.01%	
6	0.02%	0.34%	0.57%	0.05%	0.01%	2.03%	
7	0.41%	0.07%					3.00%

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
KNN	30.25%	14.75%	85.25%

D. Random Forest – Default R Settings

Using random forest, the objective is to attain significant improvements in classification accuracy. Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

In R, the *randomForest* package is used to achieve a classification or regression based on a forest of trees using random inputs. The classification random forest model is utilized in this section using default settings from the package.

We randomly divided the observations into training and test datasets and applied random forests to the training set. Training dataset consists of 101,680 observations while test dataset has 174,302 observations. They both have 63 variables.

Using the training data, the first random forest uses all the 62 predictor. We then run a Variable Importance to observe the most significant variable:

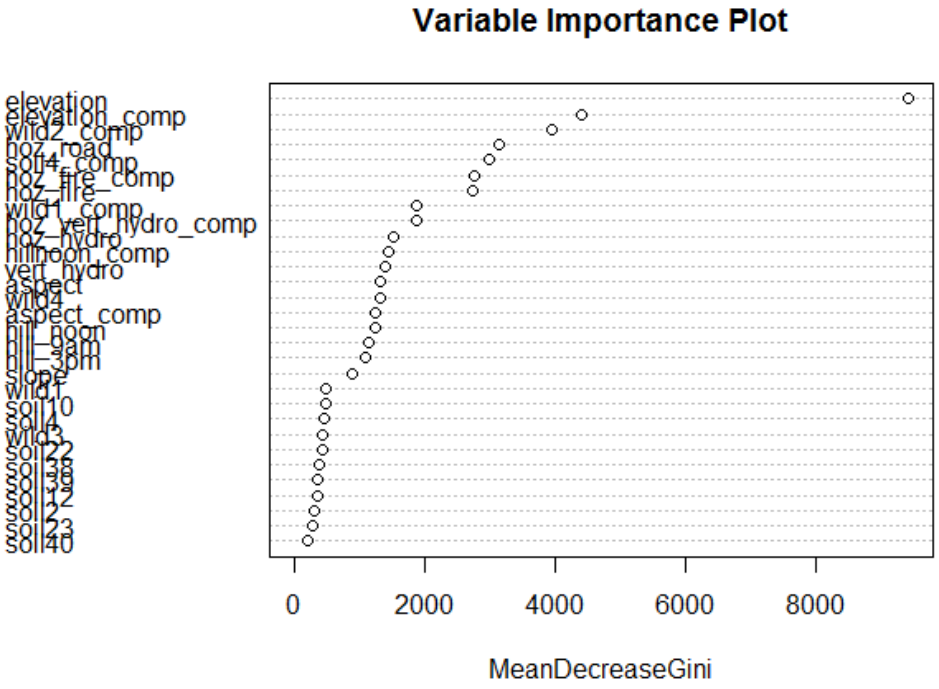
Table 9: Mean Decrease in Gini

	MeanDecreaseGini
elevation	9413.75
elevation_comp	4406.22
wild2_comp	3950.03
hoz_road	3142.58
soil4_comp	2991.67
hoz_fire_comp	2763.68
hoz_fire	2723.95
wild1_comp	1885.00
hoz_vert_hydro_comp	1882.62
hoz_hydro	1528.52
hillnoon_comp	1445.51
vert_hydro	1402.28
aspect	1323.90
wild4	1316.01
aspect_comp	1249.83

	MeanDecreaseGini
hill_noon	1241.25
hill_9am	1142.34
hill_3pm	1092.72

From above, ELEVATION proves to be the best predictor. Its mean decrease Gini is significantly higher that all the rest. Below is a plot visually representing the significance of these predictor variables.

Figure 10: Variable Importance Plot



Elevation_comp to hill_3pm are also good predictors so we run a second model with these 18 predictors. In order of importance, their mean decrease Gini is above 1,000. This second model has a better Out-of-Bag (OOB) error.

The test error is displayed as a function of the number of trees where each colored line corresponds to a different variable available for splitting at each interior node. A side by side of both models shows how well the second model performs.

Figure 11: Mean Error Trend by Number of Trees

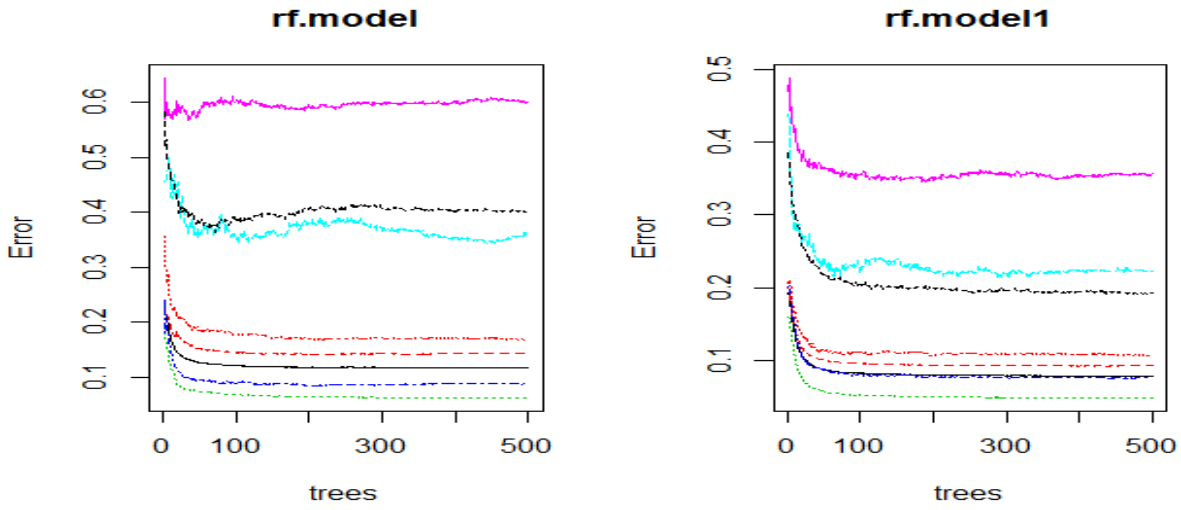


Table 10: Out-of-Bag Error Results

RFModel	OOB Error	# predictor variables	# of splits
rf.model	11.81%	62	7
rf.model1	7.85%	18	4

The RF Model, rf.model1 is an improvement of the first having the lesser OOB Error.

The following confusion matrices are presented for rf.model1.

In-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	33.08%	3.24%			0.01%	0.01%	0.12%
2	2.03%	46.39%	0.14%		0.08%	0.10%	0.02%
3		0.20%	5.69%	0.03%	0.01%	0.23%	
4		0.00%	0.09%	0.37%		0.01%	
5	0.04%	0.52%	0.02%	0.00%	1.05%		
6		0.18%	0.38%	0.01%	0.00%	2.42%	
7	0.35%	0.02%					3.16%

Out-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	33.14%	3.18%			0.02%		0.12%
2	2.00%	46.42%	0.15%		0.07%	0.11%	0.02%
3		0.19%	5.72%	0.03%	0.01%	0.20%	
4		0.00%	0.09%	0.37%	0.00%	0.01%	
5	0.03%	0.53%	0.03%		1.03%	0.01%	
6	0.01%	0.19%	0.38%	0.01%	0.00%	2.39%	
7	0.37%	0.02%	0.00%				3.14%

In-sample MSE and Out-Sample MSE are presented below:

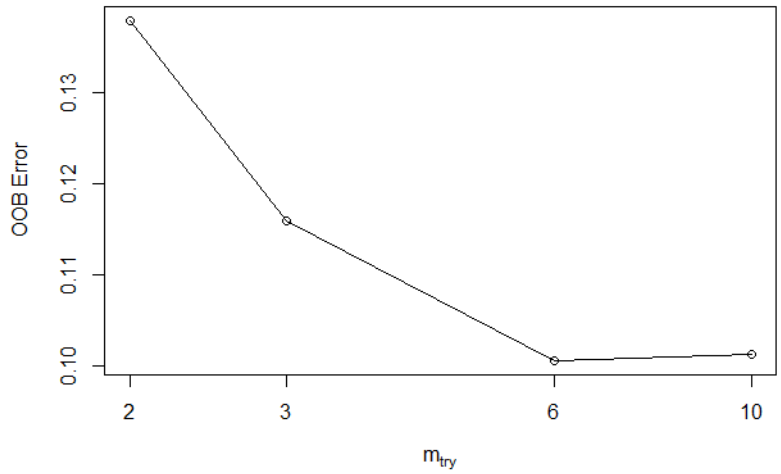
Model	In-Sample MSE	Out-Sample MSE
Random Forest – Default R Settings	7.85%	7.80%

E. Random Forest – Using tuneRF() Function

We further explore to see what happens when we try different values for *mtry* which is the number of variables selected at each node. We apply the function 'tuneRF' which searches for the optimal value (with respect to OOB error estimate) of *mtry* for randomForest. We also set the (relative) improvement in OOB error to be 0.05 for the search to continue.

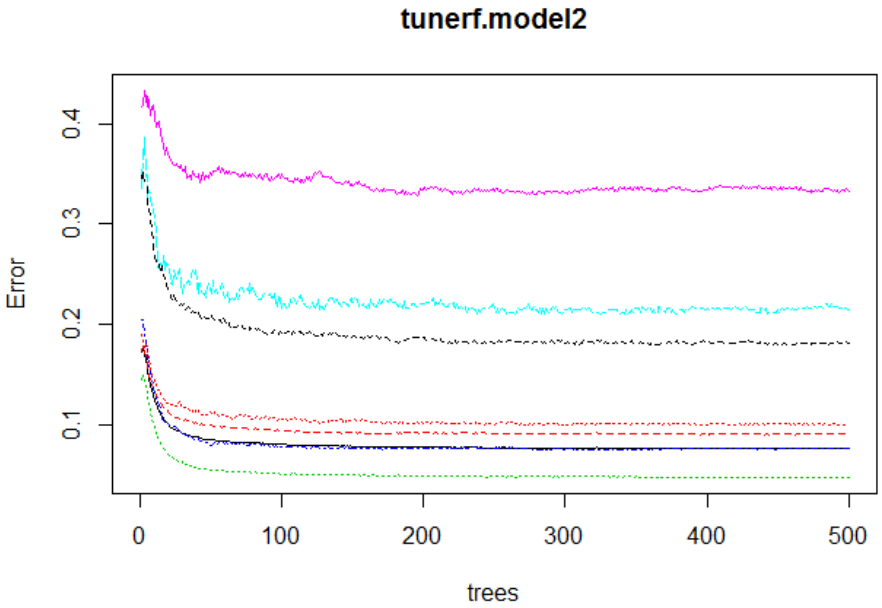
Applying this function to the training dataset after it is prepared provided the following:

Figure 11: Optimal Number of Considered Variables



It is clear that the less OOB occurs at *mtry*=6. We then run a new Random Forest with the 18 variables chosen above. The result of the model, although small, is a slight development from the last.

Figure 12: Plot of Out-of-Bag Error by Number of Trees



In-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	33.16%	3.14%			0.01%	0.01%	0.14%
2	1.98%	46.43%	0.15%		0.08%	0.10%	0.02%
3		0.20%	5.68%	0.03%	0.01%	0.23%	
4			0.09%	0.37%		0.01%	
5	0.03%	0.49%	0.02%		1.09%		
6		0.17%	0.35%	0.01%		2.45%	
7	0.33%	0.02%					3.18%

Out-Sample Confusion Matrix:

	Predicted						
Actual	1	2	3	4	5	6	7
1	33.30%	3.03%			0.02%		0.12%
2	1.95%	46.48%	0.14%		0.08%	0.10%	0.02%
3		0.19%	5.72%	0.03%	0.01%	0.20%	
4			0.08%	0.38%		0.01%	
5	0.03%	0.52%	0.03%		1.05%	0.01%	
6	0.01%	0.18%	0.35%	0.01%		2.44%	
7	0.35%	0.03%					3.15%

Final Random Forest Results:

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
Random Forest – **Default Setting**	7.85%	7.80%	92.22%
Random Forest – **Improve=0.05 Parameter for OOB**	7.64%	7.48%	92.52%

Our second model using default setting produces a good model but as we explored further by tuning the model, we resulted in a slightly better one. Based on the results from the Confusion Matrix and the list of top predictors, we can confirm that the model has a low out of sample

error. We therefore feel confident that it is a good model to use to predict cases from the "test" dataset.

F. Gradient Boosting – Using Various Shrinkage Values

Gradient boosting is used to build upon the performance of a base learner algorithm. This base learner is applied to data and the residuals are determined. How these residuals are determined is based on the loss function, a function that measures the difference between the model and the desired prediction. Next, another model is fit based on the residuals from the base model. Any remaining residuals from that model are then fit with another model and so forth reducing the amount of residuals with each iteration.

Gradient boosting has three tuning parameters when used with trees: the number of trees, interaction depth, and shrinkage. Each interacts with the other and can be a challenge to coordinate to find the most effective gradient boosting process.

In order to determine the most effective combination of number of trees, interaction depth, and shrinkage, we used the carat package to apply three different combinations of the number of trees (100, 300, 500 trees), four different interaction depths (1, 2, 3, and 4), and three different shrinkage values (0.001, 0.01, 0.1). This creates 36 different models to try to identify the best combination of tuning parameters. Furthermore, we used k-fold cross validation of 10 folds to improve the accuracy and reduce the variance of identifying the best tuning parameters.

It should be noted that due to the computational demands and local hard drive size, we used a sample size of 3000 observations from the training set in order to determine the best combination.

When running a model on all of the variables, the results identified that the optimum combination consisted of 100 trees, an interaction depth of 4, and a shrinkage value of 0.1 at an accuracy rate of 0.7454.

Table 11: Optimum Tuning Parameters using All Variables

shrinkage	interaction	n.trees	Accuracy	Kappa	Accuracy SD	Kappa SD
0.100	3	500	0.7390221	0.5720508	0.02697083	0.04468758
0.100	**4**	**100**	**0.7453711**	**0.5813908**	**0.03470773**	**0.05725804**
0.100	4	300	0.7346954	0.5655907	0.02916640	0.04847813

When running a model on the backward selection variables, the results identified that the optimum parameter combination consisted of 500 trees, an interaction depth of 4, and a shrinkage value of 0.01 at an accuracy rate of 0.6970.

Table 12: Optimum Tuning Parameters using Backward Selection Variables

shrinkage	interaction	n.trees	Accuracy	Kappa	Accuracy SD	Kappa S
0.010	4	300	0.6939918	0.4906904	0.02138787	0.03471678
0.010	**4**	**500**	**0.6969942**	**0.4971770**	**0.01862534**	**0.03031111**
0.100	1	100	0.6759703	0.4603003	0.02944727	0.04880033

The following table displays the two gradient boosting models using the optimal shrinkage parameters.

Table 13: Model Performance

Model	# Variables	# Trees	Interaction Depth	Shrinkage	In-Sample MSE	Out-Sample MSE	Train Accuracy	Test Accuracy
GBM	62	100	4	0.100	30.26%	30.26%	69.74%	69.74%
GBM	**7**	**500**	**4**	**0.010**	**29.64%**	**29.99%**	**70.36%**	**70.01%**

We applied these tuning parameters to the gbm() model fit to the training data set using all variables and achieved an accuracy rate of 0.6974106.

We applied these tuning parameters to the gbm() model fit to the test data set using all variables and achieved an accuracy rate of 0.6974086.

The difference is subtle as the GBM model on the test data almost matches the accuracy it achieved on the training data. The full model also showed these variables had the highest influence on the model and should be considered the most important.

Table 14: Variable Importance GBM on All Variables (Test Data)

Variable	Relative Influence (%)
elevation	73.60593834
hoz_vert_hydro_comp	5.14079334
elevation_comp	3.30426266
hoz_road	2.21946459
soil4_comp	2.21887789
wild2_comp	2.03673077
hoz_fire_comp	1.80487394
soil4	1.51679563
hill_noon	1.36830807
soil2	1.36321819
wild1	1.25223288

Comparing the two types of models we find that the reduced model the variable elevation is still the most influential but at a much higher level (91.5% vs. 73.6%). Furthermore, when we use the backward selected variables, with better parsimony and using 500 trees, an interaction depth of 4, and 0.01 shrinkage factor, had the best accuracy on the test set compared to the full model (0.7036 vs. 0.6974). Hence, we should use the model with the 7 variables instead of the full model.

G. Support Vector Machine – Linear Kernel

In this section, we present the support vector machine model (SVM) using the linear kernel setting from the e1071 package. SVMs are models that calculate the optimal separating hyperplane between two or more classes. With a linear setting, the model optimizes a linear separator to classify the different classes. Often, the classes overlap in which a clean separation is impossible. To accommodate this, the model softens the hyperplane by introducing margins on both sides of the hyperplane. The distance of the margins to the decision boundary is a tuning parameter, cost, that determines how many observations can be on the wrong side of the boundary. When we increase the cost value, the margin becomes wider and therefore, the separator becomes softer, allowing for more observations to be on the wrong side of the decision boundary.

For our SVM with a linear kernel, we chose seven cost values ranging from 0.001 to 100 and used cross-validation to determine the value with the lowest error, which turned out to be cost=1. Using that value, we tested the predictive accuracy of the linear SVM on both the training and test data and the resulting misclassification rate was similar for both, 27.56% and 27.7% respectively. The performance of the model on the training and test data suggests that it has a good fit to unseen data.

In-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	25.73%	10.14%	0.03%			0.01%	0.55%
	2	9.14%	38.64%	0.67%			0.28%	0.02%
	3		0.47%	5.36%	0.17%		0.16%	
	4			0.21%	0.28%			
	5		1.54%	0.09%		0.01%		
	6		0.68%	1.88%			0.43%	
	7	1.51%	0.02%					2.00%

Out-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	25.64%	10.14%	0.01%			0.01%	0.65%
	2	8.86%	38.79%	0.75%			0.31%	0.04%
	3		0.52%	5.26%	0.22%		0.15%	
	4		0.00%	0.21%	0.26%			
	5	0.01%	1.55%	0.05%		0.02%	0.01%	
	6		0.62%	1.83%	0.03%	0.00%	0.51%	
	7	1.67%	0.03%				0.00%	1.82%

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
SVM – Linear Kernel	27.56%	27.70%	72.30%

H. Support Vector Machine – Radial Kernel

In this section, we present the support vector machine model (SVM) using the radial kernel setting from the *e1071* package. Sometimes, a linear separation is not the most effective separator and non-linear decision boundaries need to be considered. To find such boundaries, we fit a hyperplane on a higher dimension by transforming the variables. A radial kernel setting is an example of this attempt and often a popular kernel for SVMs. There are two tuning parameters to consider, the cost value and a gamma value.

We use cross-validation to identify the best values which was cost=10 and gamma=2. When we use this model to fit the training data, we get a misclassification rate of 0.91%, which is a remarkable improvement over the linear kernel. However, when we fit the model to the test data, we get a rate of 23.35%. While that is still a noticeable improvement over the linear kernel's out-of-sample error, the divergence in the errors may indicate that the model is

overfitting the data. More investigation needs to be done to uncover the reason for the very large discrepancy between the training and test misclassification rates.

In-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	35.88%	0.56%			0.01%		0.01%
	2	0.27%	48.46%	0.01%			0.01%	
	3		0.01%	6.14%				
	4				0.48%	0.00%		
	5		0.01%		0.00%	1.63%		
	6		0.02%				2.97%	
	7		0.01%					3.52%

Out-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	28.10%	7.92%			0.08%	0.02%	0.34%
	2	6.92%	40.56%	0.40%		0.56%	0.28%	0.04%
	3	0.02%	1.43%	3.92%	0.07%	0.03%	0.68%	
	4		0.15%	0.12%	0.17%	0.00%	0.03%	
	5	0.14%	0.90%	0.03%		0.55%	0.02%	
	6	0.08%	0.88%	0.64%	0.02%	0.01%	1.35%	
	7	0.94%	0.59%					2.00%

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
SVM – Radial Kernel	0.91%	23.35%	76.65%

I. Neural Network – Learning Rate (0.05)

A Neural Network seeks to mimic the function of the brain by "learning" from the data in an iterative process called backpropagation. The objective of backpropagation is to assign optimal weights to each node, also known as a neuron, so the algorithm can map arbitrary inputs to outputs. A Neural Network has three types of layers: input, hidden, and output. The input layer contains a node for each predictor variable, in this case the four PCA components. The hidden layer contains nodes that augment the weights with additional functions that produce the classification in the output layer.

Neural Network algorithms are well-known for being computationally intensive. The size of the training data set, number of variables, and model parameters all effect run time. A common issue with this experimentation was running an algorithm overnight only to have the algorithm not reach convergence within the parameter limits (stepmax). Due to the restrictions of the computer hardware used to conduct this analysis, the number of model variants is limited.

In this section of the report we compute two models using the neuralnet() package in R with adjustments to the learning rate parameter. The learning rate determines how much an iteration influences the model weights. We begin with the training data set that includes four principle components based on the continuous predictor variables. These components are hill3pm_comp, hillnoon_comp, hozvert_comp, hozroad_comp.

Model 1

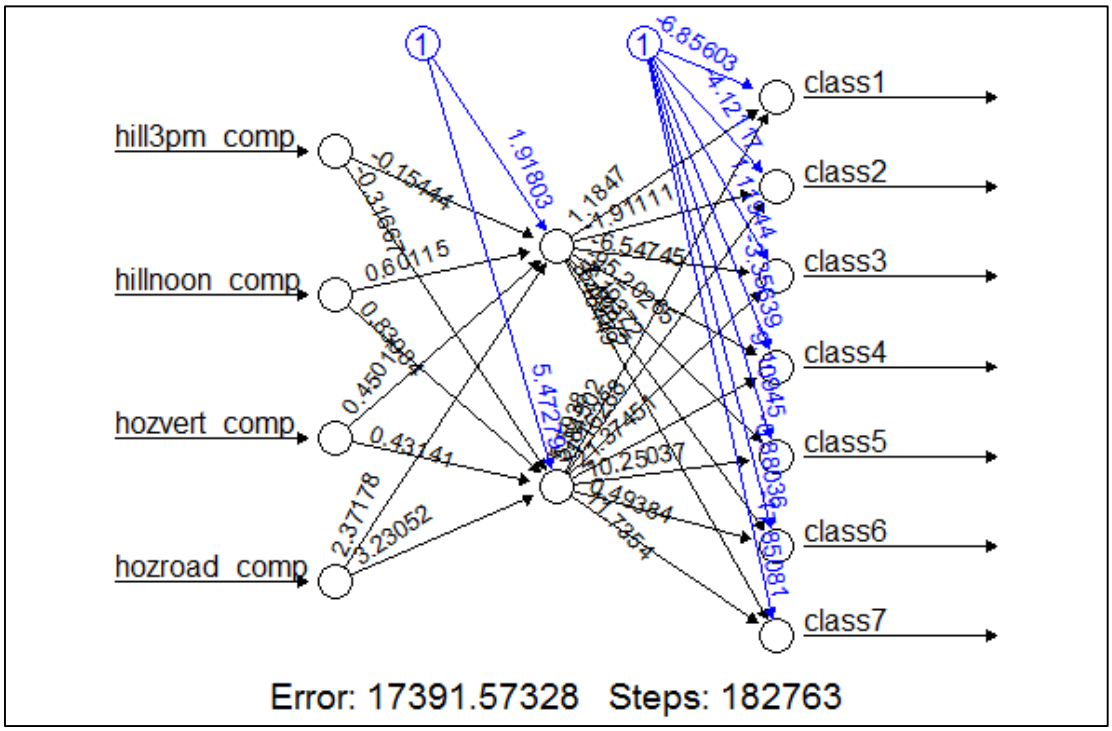
Here we build a function with minimal modification to the default parameters:

*nn1 <- neuralnet(f,data=model_cover,hidden=2,**learningrate=0.50**,*

err.fct="ce",linear.output=FALSE, stepmax=300000

The output in Figure 16 shows the four input nodes (predictors), the two hidden layer nodes, and the seven output nodes (classes). The black lines represent the weights assigned and processed by each node. The blue lines represent the bias that is factored into the weight calculations.

Figure 16: NN Model 1, 0.5 Learning Rate, 2 Neurons in Hidden Layer



In-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	13.08%	12.68%			0.07%		1.86%
	2	23.27%	35.15%	2.45%	0.01%	1.53%	1.43%	1.67%
	3	0.11%	0.91%	3.71%	0.47%	0.04%	1.56%	
	4							
	5							
	6							
	7							

The in-sample misclassification rate is 48.1%.

Out-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	12.86%	12.61%			0.09%		1.86%
	2	23.51%	35.07%	2.44%	0.05%	1.49%	1.32%	1.67%
	3	0.09%	1.07%	3.71%	0.42%	0.06%	1.67%	
	4							
	5							
	6							
	7							

The out-sample misclassification rate is 48.4 %.

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
Neural Network – Learning Rate = 0.50 Number of Neurons = 2	48.1%	48.4%	51.60%

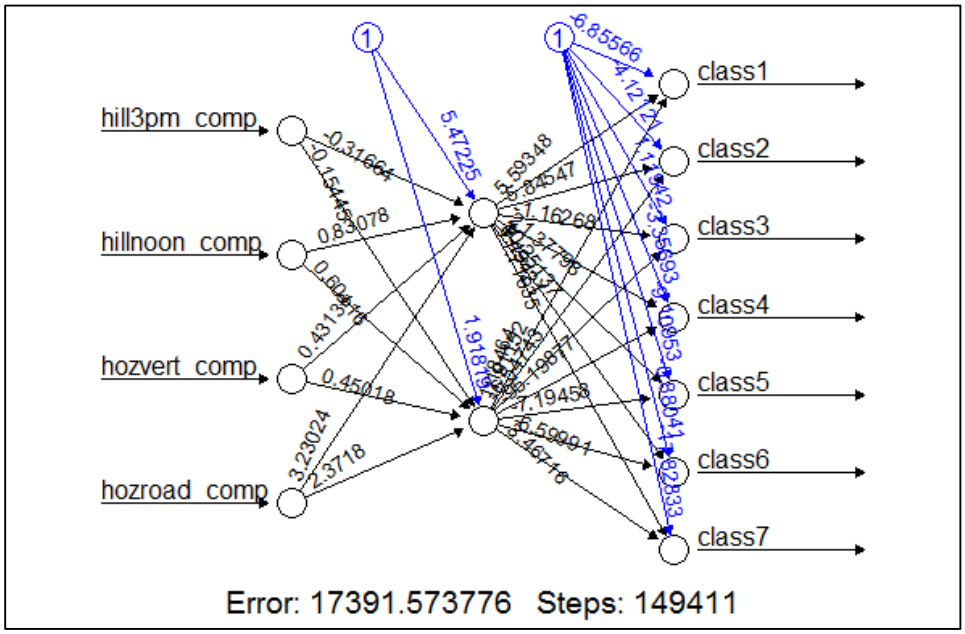
J. Neural Network – Learning Rate (0.35)

The second model has a decreased learning rate of 0.35 with the number of hidden neurons held consistent at two:

*nn2 <- neuralnet(f,data=model_cover,hidden=2,**learningrate=0.35**,*

err.fct="ce",linear.output=FALSE, stepmax=300000)

Figure 17: NN Model 1, 0.35 Learning Rate, 2 Neurons in Hidden Layer



As shown in Figures 16 and 17, the number of backpropagation steps was 182,763 in Model 1 and 149,411 in Model 2. This explains the longer run time for Model 1 and the possible effect of the learning rate parameter. The model error rates are nearly the same value at 17,391. This indicates that the learning rate had little effect on the model output.

In-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	13.08%	12.68%			0.07%		1.86%
	2	23.27%	35.15%	2.45%	0.01%	1.53%	1.43%	1.67%
	3	0.11%	0.91%	3.71%	0.47%	0.04%	1.56%	
	4							
	5							
	6							
	7							

The in-sample misclassification rate is 48.1%.

Out-Sample Confusion Matrix:

		PREDICTED						
		1	**2**	**3**	**4**	**5**	**6**	**7**
ACTUAL	**1**	12.86%	12.61%			0.09%		1.86%
	2	23.51%	35.07%	2.45%	0.05%	1.49%	1.32%	1.67%
	3	0.09%	1.07%	3.71%	0.42%	0.06%	1.67%	
	4							
	5							
	6							
	7							

The out-sample misclassification rate is 48.4 %.

Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
Neural Network – Learning Rate = 0.35 Number of Neurons = 2	48.1%	48.4%	51.60%

Comparing the results for both models we observe that the Neural Net only predicts correct results slightly better than 50% of the time. Both models performed approximately the same, showing that the learning rate did not have much of an effect on prediction. This is also consistent with the error rates shown in Figures 16 and 17. Between both models, only slightly different results were obtained on the training data. Predictions on the testing data yielded the same results.

The worst results appear to be with covertypes 4 to 7 where no predictions were made. The frequency of these classes in the data set used to train the model is less than 5% of all data. Cover type 1 was predicted the most accurately, with 75% true positives.

In an attempt to improve upon these initial modeling results, equal samples of each classification were used as the training set. The approach, based on published research, was a method to reduce the variable bias of the more frequent cover types. Unfortunately, using the

same model parameters causes inferior results, yielding misclassification rates of greater than 85%. A review of other literature suggests that many additional neurons are needed to surpass the model misclassification rates produced in this report (Blackard, 1999).

VI. Comparison of Results

In this section, we evaluate the predictive performance of each presented model. We present in-sample and out-sample predictive accuracy using misclassification rates.

Table 15: In-Sample and Out-Sample Predictive Performance

Letter	Model	In-Sample MSE	Out-Sample MSE	Test Accuracy
A	Linear Discriminant Analysis	33.95%	32.87%	67.13%
B	Quadratic Discriminant Analysis	33.67%	33.69%	66.31%
C	K-Nearest Neighbors	30.25%	14.75%	85.25%
D	Random Forest – Default Setting	7.85%	7.80%	92.22%
E	**Random Forest – *Improve=0.05* Parameter for OOB**	**7.64%**	**7.48%**	**95.52%**
F	Gradient Boosting – Optimal Shrinkages	29.64%	29.99%	70.01%
G	Support Vector Machine – Linear Kernel	27.56%	27.70%	72.30%
H	Support Vector Machine – Radial Kernel	0.91%	23.35%	76.65%
I	Neural Network – Learning Rate = 0.05 Number of Neurons = 2	48.1%	48.4%	51.60%
J	Neural Network – Learning Rate = 0.35 Number of Neurons = 2	48.1%	48.4%	51.60%

We conclude that the random forest model with the tuning parameter (number of considered variables at each split) outperforms all other models since it has the highest test accuracy of 95%. Next, K-NN algorithm exhibits a high test accuracy of 85% on unseen data. Both gradient boosting and SVM models yield similar test accuracies of 70%-72%. However, SVM with radial kernel outperforms the SVM with linear kernel, suggesting that the forest cover data exhibits non-linear patterns. Both linear and quadratic discriminant models exhibit similar test accuracies of 66%. Finally, the worst performing model is the neural network. However, we reserve some

definite conclusions on the performance of the neural network due to additional options to be considered for future modeling problems. For instance, the performance of the neural network relies heavily on the topography of the original network, such as number of hidden nodes, hidden layers, and activation functions. In order to fully evaluate the its performance, future modeling problems will require experimentation with different number of hidden neurons, activation functions, and learning rates. Overall, we conclude that pure machine learning models outperforms the traditional regression models (LDA and QDA). The test MSE of machine learning models are lower than LDA and QDA. However, there is a trade-in to consider between interpretation of individual effects of predictors and overall predictive accuracies. We lose human interpretation of the importance of variables in exchange of gaining the most accurate outcomes.

VII. Conclusions

In this project, we develop a suite of statistical learning models to predict forest cover type into seven categories. Using a set of cartographic variables, our objective is to classify forests into one of seven categories.

First, we discover that forest ELEVATION is the most important quantitative predictor to classify a covertype. In addition, we identify that quantitative variables are the most important using model-based EDA decision tree and random forest models. Gleaned from this insight, machine learning models are built using the most appropriate variables.

Our best model is the random forest with a tuning parameter (modification in the number of variables considered at each split), which exhibits a correct classification rate of 95% on unseen data. We also conclude that gradient boosting and support vector machine models yield

one of the highest accuracies (correct classifications) on unseen data compared to traditional regression models, such as LDA and QDA. Our neural network model performs the worst compared to all models due to our sample size and computing resources. We believe that by working with a much larger sample and by experimenting with a much larger number of hidden neurons, we will obtain greater prediction accuracies. For future modeling problems, we will consider using various activations functions and learning rates to improve predictive accuracy.

In terms of quality of our results, we believe that by working with a larger training dataset and utilizing available tuning parameters for the appropriate machine learning models (e.g. experimenting with different number of variables, increasing learning rates, modifying kernels, and adding the number of hidden neurons), we can improve predictive accuracies. We believe that the predictive accuracies of models will heavily depend on the topography of the data, which requires the most optimal tuning parameters. Additionally, due to computing constraints, we only utilized 2.5% of the 70% of the training data. Hence, our results exhibit poorer results.

Finally, for further modeling activities, we would approach the problem differently by examining the genetic algorithm and compare its predictive accuracy with our machine learning models. Genetic algorithm offers special features that learns from combinations of outputs and always seeks for the best outcome. It is "evolutionary" in nature, which means that the algorithm learns from historical patterns and seeks the best combinations of outputs. In the next iterations, the algorithm employs the previous outcomes to provide the next optimal outcomes. In all, this predictive modeling problem presents a large avenue of modeling research in the areas of predictive analytics using unbalanced outcomes and how they affect predictive accuracies.

VIII. Bibliography

Blackard, J.A. & Dean, D.J. (1999). *Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables*. Computers and Electronics in Agriculture, 24 (1999), 131-151.

Chiu, D. (2015). *Machine Learning with R Cookbook*. Birmingham, UK: Packt Publishing Ltd.

Cilimkovic, M. *Neural Networks and Back Propagation Algorithm*. Retrieved from http://www.dataminingmasters.com/uploads/studentProjects/NeuralNetworks.pdf

Forte, L. M. (2015). *Mastering Predictive Analytics with R*. Birmingham, UK: Packt Publishing Ltd.

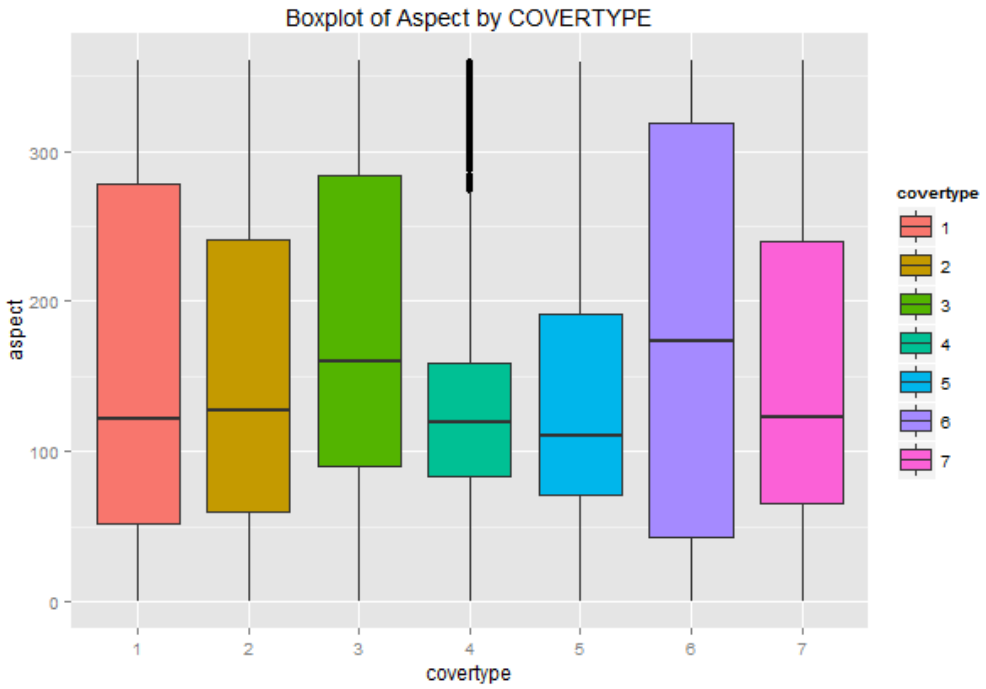
Lesmeister, C. (2015). *Mastering Predictive Machine Learning with R*. Birmingham, UK: Packt Publishing Ltd.

Pohar, M., Blas, M., & Turk, S. (2004). *Comparison of Logistic Regression and Linear Discriminant Analysis: A Simulation Study*. Metodološki zvezki, 1(1), 143-161. Retrieved from http://mrvar2.fdv.uni-lj.si/pub/mz/mz1.1/pohar.pdf

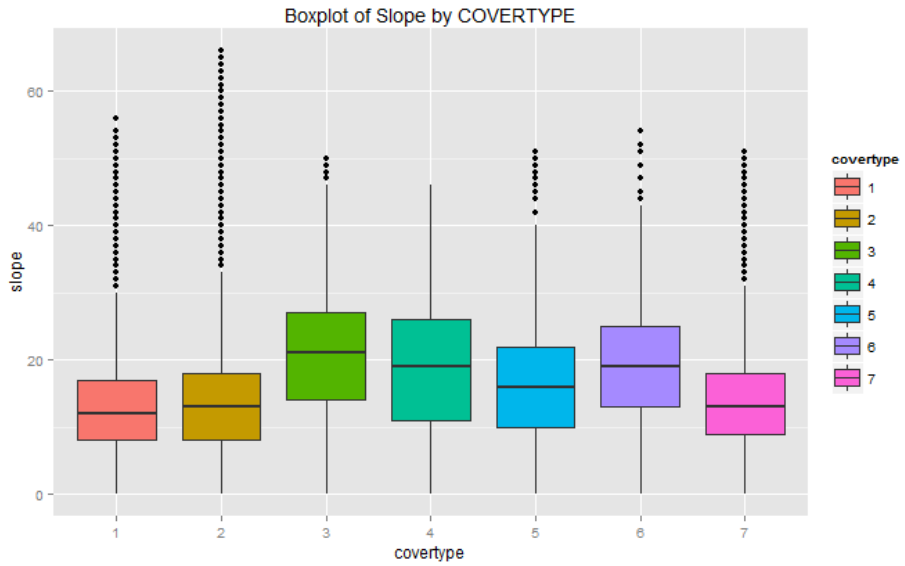
IX. Appendices

Exploratory Data Analysis:

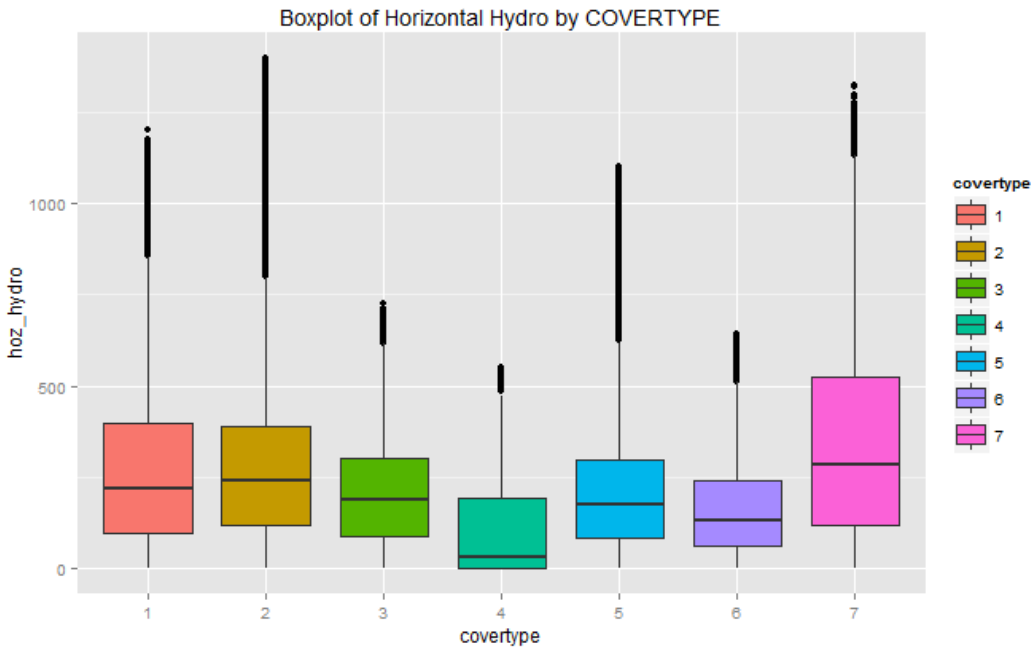
ASPECT by COVERTYPE



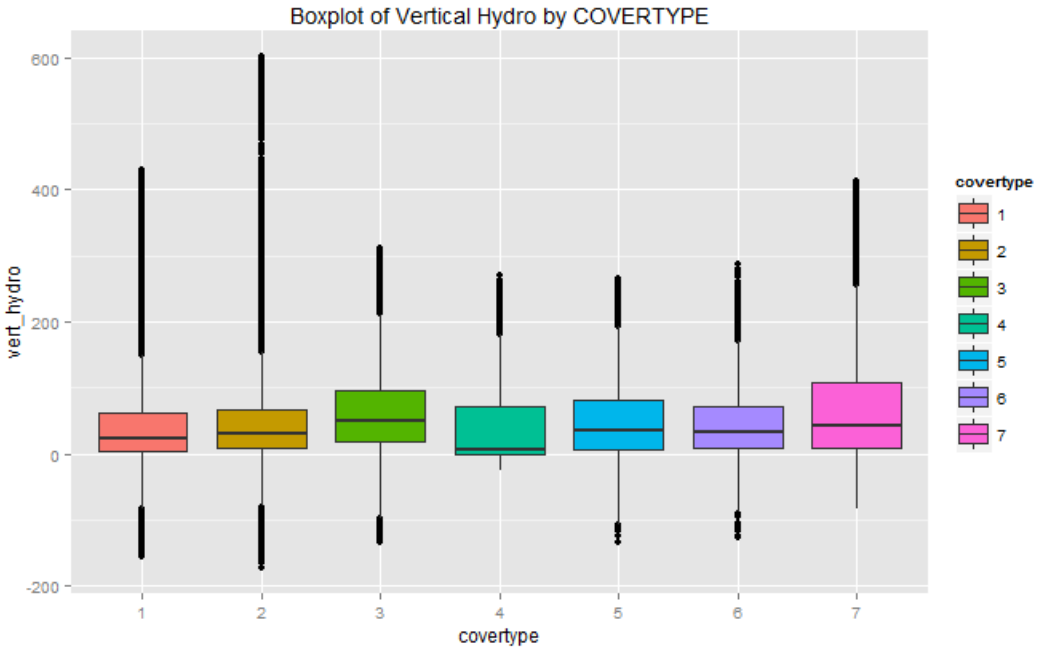
SLOPE by COVERTYPE



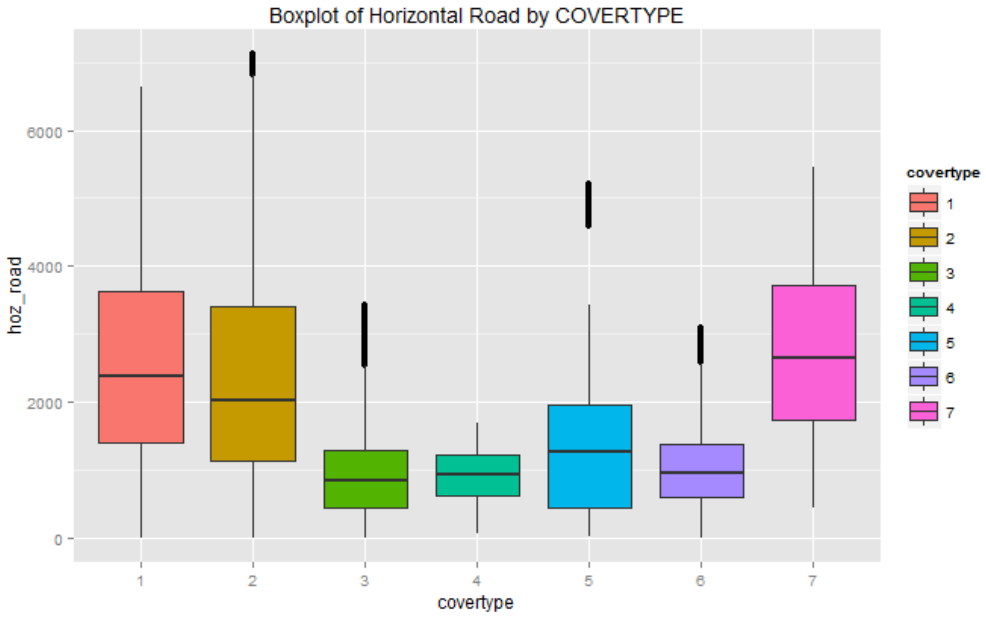
HORIZONTAL HYDRO BY COVERTYPE



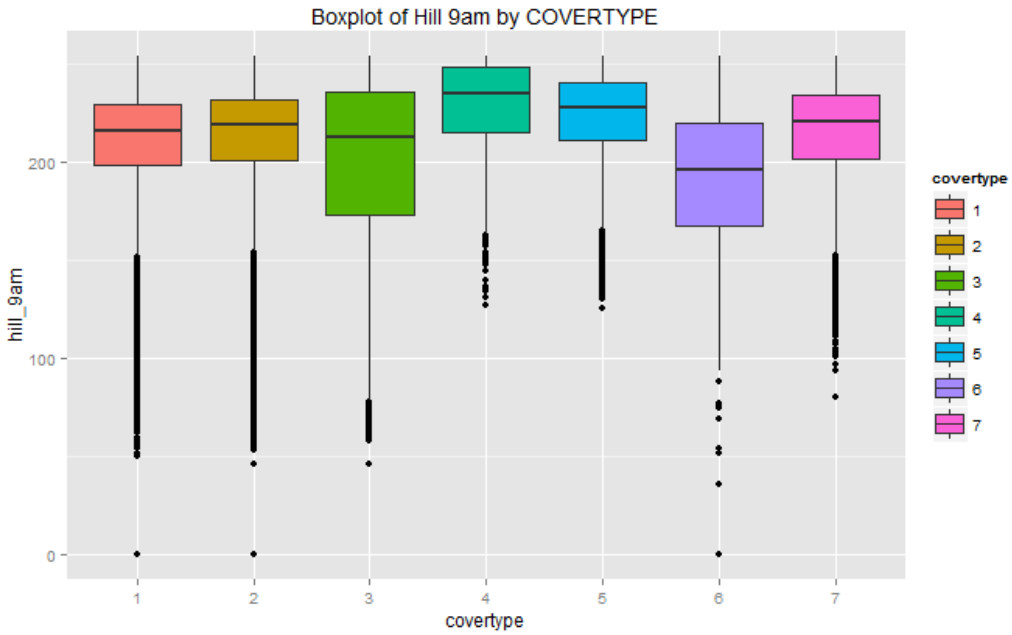
VERTICAL HYDRO BY COVERTYPE



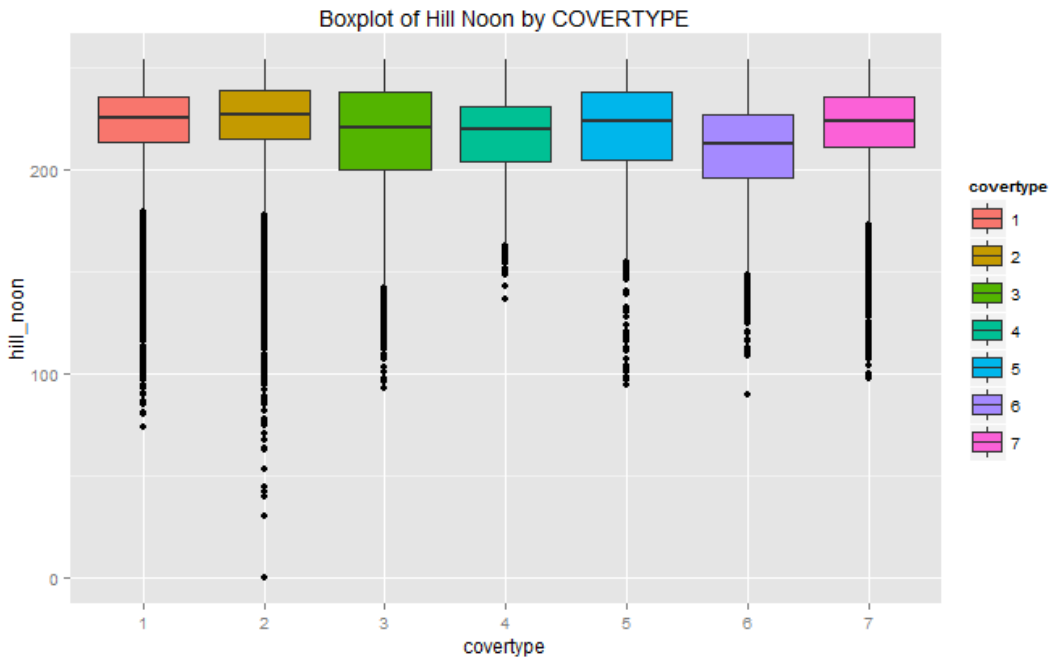
HORIZONTAL ROAD BY COVERTYPE



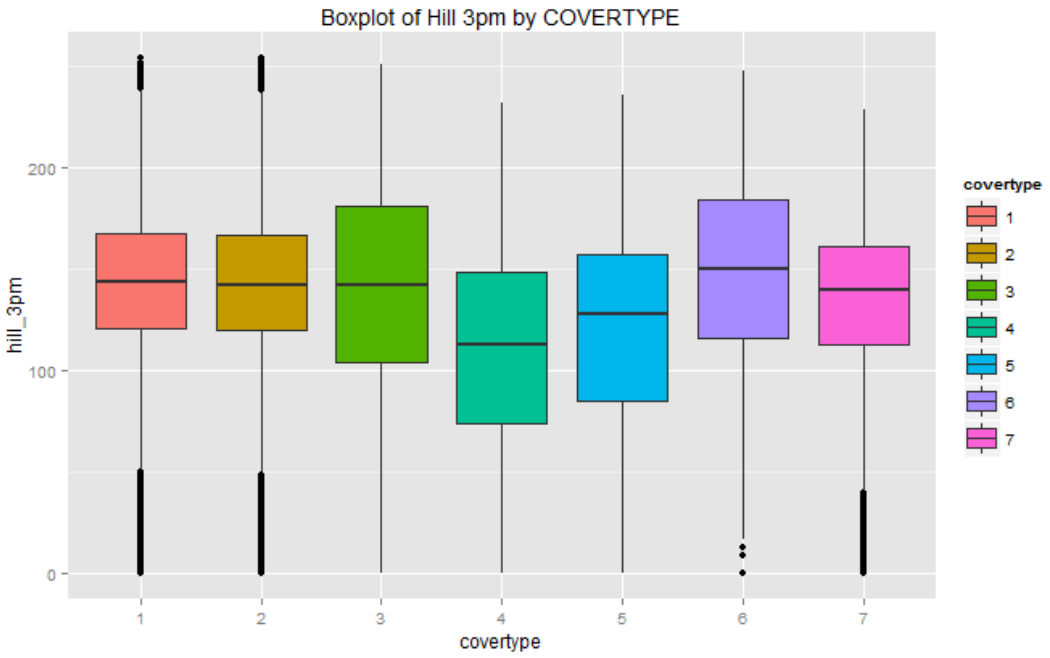
HILLSIDE 9AM BY COVERTYPE



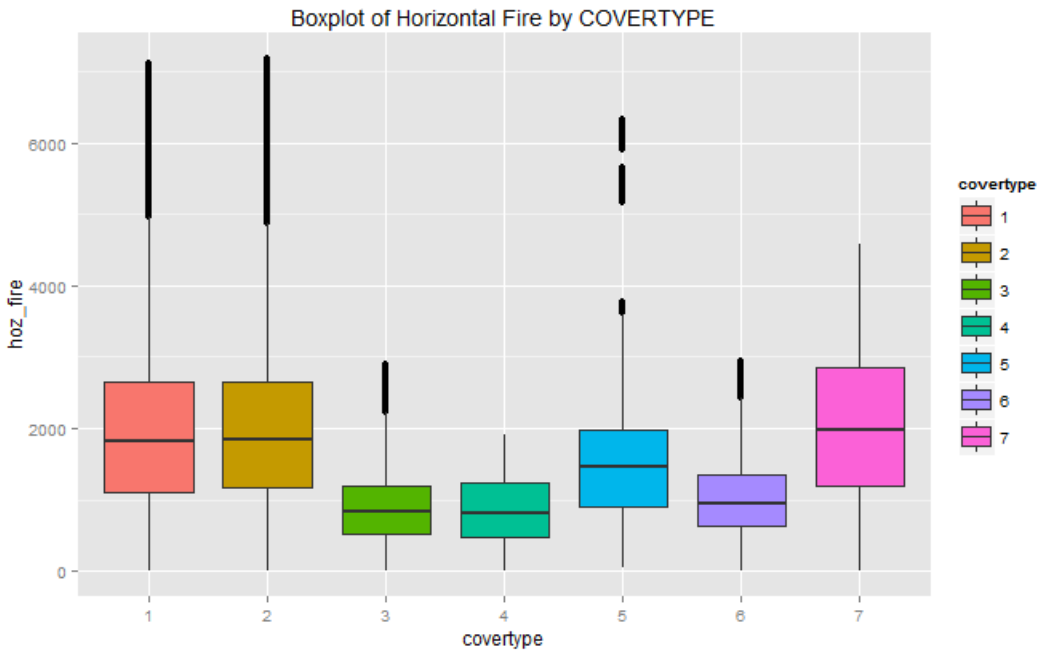
HILLSIDE NOON BY COVERTYPE



HILLSIDE 3PM BY COVERTYPE



HORIZONTAL FIRE BY COVERTYPE



Backward Selection Results:

```
> regfit.bkwd = regsubsets(covertype~elevation+aspect+slope+hoz_hydro+vert_hydro+hoz_road+
+                            hill_9am+hill_noon+hill_3pm+hoz_fire,
+                          data=newtree,nvmax=10,method="backward")
> summary(regfit.bkwd)
Subset selection object
Call: regsubsets.formula(covertype ~ elevation + aspect + slope + hoz_hydro +
    vert_hydro + hoz_road + hill_9am + hill_noon + hill_3pm +
    hoz_fire, data = newtree, nvmax = 10, method = "backward")
10 Variables  (and intercept)
           Forced in Forced out
elevation      FALSE      FALSE
aspect         FALSE      FALSE
slope          FALSE      FALSE
hoz_hydro      FALSE      FALSE
vert_hydro     FALSE      FALSE
hoz_road       FALSE      FALSE
hill_9am       FALSE      FALSE
hill_noon      FALSE      FALSE
hill_3pm       FALSE      FALSE
hoz_fire       FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: backward
          elevation aspect slope hoz_hydro vert_hydro hoz_road hill_9am hill_noon hill_3pm hoz_fire
1  ( 1 )  "*"       " "    " "   " "       " "         " "      " "      " "       " "      " "
2  ( 1 )  "*"       " "    " "   " "       "*"         " "      " "      " "       " "      " "
3  ( 1 )  "*"       " "    " "   " "       "*"         " "      " "      " "       " "      "*"
4  ( 1 )  "*"       " "    " "   " "       "*"         " "      " "      " "       "*"      "*"
5  ( 1 )  "*"       "*"    " "   " "       "*"         " "      " "      " "       "*"      "*"
6  ( 1 )  "*"       "*"    "*"   " "       "*"         " "      " "      " "       "*"      "*"
7  ( 1 )  "*"       "*"    "*"   " "       "*"         "*"      " "      " "       "*"      "*"
8  ( 1 )  "*"       "*"    "*"   " "       "*"         "*"      " "      "*"       "*"      "*"
9  ( 1 )  "*"       "*"    "*"   "*"       "*"         "*"      " "      "*"       "*"      "*"
10  ( 1 ) "*"       "*"    "*"   "*"       "*"         "*"      "*"      "*"       "*"      "*"
>
```

Table of Principal Components:

```
Principal Components Analysis
Call: principal(r = newtree[, 1:10], nfactors = 4, rotation = "varimax")
Standardized loadings (pattern matrix) based upon correlation matrix
             PC1   PC2   PC3   PC4   h2    u2 com
elevation  -0.05  0.36  0.42  0.47 0.53 0.475 2.9
aspect      0.83  0.09  0.02 -0.04 0.69 0.307 1.0
slope       0.15 -0.85  0.10 -0.16 0.79 0.210 1.2
hoz_hydro  -0.01  0.08  0.90  0.06 0.81 0.187 1.0
vert_hydro  0.10 -0.24  0.84 -0.09 0.79 0.209 1.2
hoz_road    0.08  0.15  0.04  0.81 0.68 0.318 1.1
hill_9am   -0.89  0.30 -0.04  0.04 0.89 0.107 1.2
hill_noon   0.35  0.83  0.00  0.04 0.82 0.179 1.4
hill_3pm    0.91  0.33  0.02  0.02 0.94 0.065 1.3
hoz_fire   -0.10 -0.02 -0.07  0.77 0.61 0.390 1.1

                       PC1  PC2  PC3  PC4
SS loadings           2.48 1.85 1.71 1.51
Proportion Var        0.25 0.18 0.17 0.15
Cumulative Var        0.25 0.43 0.60 0.76
Proportion Explained  0.33 0.24 0.23 0.20
Cumulative Proportion 0.33 0.57 0.80 1.00

Mean item complexity =  1.3
Test of the hypothesis that 4 components are sufficient.

The root mean square of the residuals (RMSR) is  0.08
 with the empirical chi square  339983.3  with prob <  0

Fit based upon off diagonal values = 0.91
```

Decision Tree - EDA

```
> library(rpart)
> set.seed(123)
> class.tree <- rpart(covertype~.,data=newtree,method="class")
> summary(class.tree)
Call:
rpart(formula = covertype ~ ., data = newtree, method = "class")
  n= 581012

          CP nsplit rel error    xerror        xstd
1 0.28489038      0 1.0000000 1.0000000 0.001279776
2 0.07141825      1 0.7151096 0.7151096 0.001233641
3 0.01000000      2 0.6436914 0.6436914 0.001203745

Variable importance
elevation     wild4     wild2 hoz_hydro    soil10    soil22  hoz_road    soil38     soil6     soil1     soil5
       59        17         4         4         3         3         2         2         2         2         1
```

Random Forest EDA:

```
> set.seed(568)
>
> rf.model <- randomForest(covertype~.,data=train,probability=TRUE)
> print(rf.model)

Call:
 randomForest(formula = covertype ~ ., data = train, probability = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 7

        OOB estimate of  error rate: 15.78%
Confusion matrix:
       1      2     3    4    5    6     7 class.error
1 122389  25166    51    0    2   15   665  0.17465338
2  17423 178942  1614    7   72  186    67  0.09766982
3      1   1707 22924   80    3  313     0  0.08406585
4      0      0   755 1158    0   10     0  0.39781591
5    320   4776   221    0 1322    7     0  0.80108336
6     19   2121  5060   62    0 4895     0  0.59735132
7   3374     81     0    0    0    0 10902  0.24064916
> plot(rf.model)
> plot(rf.model)
> varImpPlot(rf.model,main="Variable Importance Plot")
> names(rf.model)
 [1] "call"            "type"            "predicted"       "err.rate"        "confusion"       "votes"
 [7] "oob.times"       "classes"         "importance"      "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"          "y"               "test"            "inbag"
[19] "terms"
```

Multinomial Logistic Regression:

This output displays the p-values of each predictor, showing statistical significance.

```
>
> #2-tailed z test
> p <- (1 - pnorm(abs(z), 0, 1))*2
> p
  (Intercept) elevation       aspect     slope    hoz_hydro   vert_hydro hoz_road     hill_9am    hill_noon     hill_3pm
2           0         0 0.000000e+00 0.0000000 0.000000e+00 1.395053e-05        0 0.000000e+00 0.000000e+00 0.000000e+00
3           0         0 9.555312e-05 0.0000000 0.000000e+00 0.000000e+00        0 0.000000e+00 0.000000e+00 0.000000e+00
4           0         0 4.069652e-04 0.6750485 4.590506e-11 0.000000e+00        0 0.000000e+00 0.000000e+00 0.000000e+00
5           0         0 0.000000e+00 0.0000000 0.000000e+00 0.000000e+00        0 0.000000e+00 0.000000e+00 0.000000e+00
6           0         0 0.000000e+00 0.0000000 9.346773e-05 0.000000e+00        0 0.000000e+00 4.440892e-16 0.000000e+00
7           0         0 0.000000e+00 0.0000000 1.456797e-01 0.000000e+00        0 5.373479e-14 3.744822e-09 1.281612e-07
      hoz_fire wild1 wild2 wild3 wild4 soil1 soil2 soil3 soil4 soil5 soil6 soil7 soil8 soil9 soil10 soil11 soil12 soil13 soil14
2 1.206314e-03     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0    NaN
3 0.000000e+00     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0      0
4 0.000000e+00     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0      0
5 1.269148e-05     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0      0
6 0.000000e+00     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0      0
7 0.000000e+00     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0      0      0
  soil15 soil16 soil17 soil18 soil19 soil20 soil21 soil22 soil23 soil24 soil25 soil26 soil27 soil28 soil29 soil30 soil31 soil32
2      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
  soil33 soil34 soil35 soil36 soil37 soil38 soil39 soil40
2      0      0      0      0    NaN      0      0      0
3      0      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0      0
5      0      0      0      0      0      0      0      0
6      0      0      0      0      0      0      0      0
7      0      0      0      0      0      0      0      0
```

Exploratory Data Analysis Codes:

```
# ----------------------
# prep file for eda
# ----------------------
library(ggplot2)
library(VIM)
library(scales)
library(gplots)
library(multcomp)
library(mlogit)
library(dplyr)
load("D:/Northwestern_2014/PREDICT_454/Assignments/Project/Forest_Cover/trees.RData")
names(newtree)


# -------------
# factor
# -------------
newtree$covertype <- as.factor(newtree$covertype)

# convert numeric to factor
for (i in names(newtree)[11:55])
{
  newtree[[i]] <- as.factor(newtree[[i]])
  next
}
sapply(newtree,class)


# ---------------------
# MISSING VALUES
# ---------------------
aggr(newtree,prop=TRUE,numbers=TRUE,title="Proportion of Missing Values")


# -------------
# EDA
# -------------
#quantitative variables
summary(newtree)


# ------------------------
# CORRELATION DIAGRAM
# ------------------------
```

```
library(corrplot)
library(RColorBrewer)
M<-cor(newtree[,1:10])
head(round(M,2))
corrplot(M, type="upper", order="hclust", tl.col="black",tl.srt=45)


# ----------------------
# multinomial - EDA
# ----------------------
library(nnet)
library(DAMisc)
multi.mod1 <- multinom(covertype~.,data=newtree)
summary(multi.mod1)
exp(coef(multi.mod1))

z <- summary(multi.mod1)$coefficients/summary(multi.mod1)$standard.errors

#2-tailed z test
p <- (1 - pnorm(abs(z), 0, 1))*2
p

#store probabilities
pp <- fitted(multi.mod1)
head(pp)

cat1 <- mean(pp[,1])
cat2 <- mean(pp[,2])
cat3 <- mean(pp[,3])
cat4 <- mean(pp[,4])
cat5 <- mean(pp[,5])
cat6 <- mean(pp[,6])
cat7 <- mean(pp[,7])

print(cat1)
print(cat2)
print(cat3)
print(cat4)
print(cat5)
print(cat6)
print(cat7)

# ---------------------------------------
# plot mean probability by covertype
# ---------------------------------------
foresttype <- c("Type1","Type2","Type3","Type4","Type5","Type6","Type7")
```

```
meanprob <-
c(0.3048505,0.3955023,0.07807275,0.03914156,0.04719241,0.05938792,0.07585254)

newdf <- data.frame(foresttype,meanprob)
plot(newdf,main="Mean Probability by COVERTYPE")


# ----------------------------------------
# prepare multinomial results for graph
# ----------------------------------------
plot(NA,xlim=c(min(newtree$elevation),max(newtree$elevation)),
    ylim=c(0,1),xlab="Elevation",ylab="Predicted Probability",bty="l")
polygon(c(newtree$elevation,rev(newtree$elevation)),c(pp[,1],rep(0,nrow(pp))),col=rgb(1,0,0,0.
3))


# ------------------------
# PRINCIPAL COMPONENTS
# ------------------------
library(psych)
names(newtree)
pcaout <- fa.parallel(newtree[,1:10],fa="pc",n.iter=500,show.legend=FALSE,main="Scree Plot")
pc <- principal(newtree[,1:10],nfactors=4,rotation="varimax")
pc
names(pc)

#create components using scores
pca.scores <- pc$scores
pca.scores <- as.data.frame(pca.scores)
head(pca.scores)

# plot components
pc1.2 <- qplot(x=PC1, y=PC2, data=pca.scores) + theme(legend.position="none")
pc1.2

# new components
newtree$hill3pm_comp <- pca.scores$PC1
newtree$hillnoon_comp <- pca.scores$PC2
newtree$hozvert_comp <- pca.scores$PC3
newtree$hozroad_comp <- pca.scores$PC4

#hill3pm_comp+hillnoon_comp+hozvert_comp+hozroad_comp

# old components
newtree$wild1_comp <- pca.scores$PC1
newtree$elevation_comp <- pca.scores$PC2
```

```
newtree$aspect_comp <- pca.scores$PC3
newtree$hoz_vert_hydro_comp <- pca.scores$PC4
newtree$wild2_comp <- pca.scores$PC5
newtree$hillnoon_comp <- pca.scores$PC6
newtree$hoz_fire_comp <- pca.scores$PC7
newtree$soil4_comp <- pca.scores$PC8


# ------------------------
# boxplots
# ------------------------
ggplot(data=newtree,aes(x=covertype,y=elevation,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Elevation by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=aspect,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Aspect by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=slope,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Slope by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hoz_hydro,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Horizontal Hydro by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=vert_hydro,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Vertical Hydro by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hoz_road,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Horizontal Road by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hill_9am,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Hill 9am by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hill_noon,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Hill Noon by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hill_3pm,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Hill 3pm by COVERTYPE")

ggplot(data=newtree,aes(x=covertype,y=hoz_fire,fill=covertype)) + geom_boxplot() +
  ggtitle("Boxplot of Horizontal Fire by COVERTYPE")


# ------------------------
# DENSITY PLOTS
# ------------------------
ggplot(newtree) + geom_density(aes(x=elevation)) + ggtitle("Density of Elevation")
```

```
ggplot(newtree) + geom_density(aes(x=aspect)) + ggtitle("Density of Aspect")
ggplot(newtree) + geom_density(aes(x=slope)) + ggtitle("Density of Slope")
ggplot(newtree) + geom_density(aes(x=hoz_hydro)) + ggtitle("Density of Horizontal Hydro")


newtree$log_elevation <- log(newtree$elevation)
newtree$log_aspect <- log(newtree$aspect+1)
ggplot(newtree) + geom_density(aes(x=log_aspect)) + ggtitle("Density of LOG Aspect")


# ---------------------------
# TREE EDA
# ---------------------------
library(party)
library(partykit)
library(rpart)
set.seed(123)
class.tree <- rpart(covertype~.,data=newtree,method="class")
summary(class.tree)

plot(as.party(class.tree))


# -------------------------------------------------
# RANDOM FOREST - EDA
# -------------------------------------------------
gc()
memory.limit(28000)
library(randomForest)
set.seed(568)

rf.model <- randomForest(covertype~.,data=newtree)

print(rf.model)
plot(rf.model)
varImpPlot(rf.model,main="Variable Importance Plot")
names(rf.model)


# ----------------------------------
# backward selection
# ----------------------------------
library(leaps)

regfit.bkwd = regsubsets(covertype~elevation+aspect+slope+hoz_hydro+vert_hydro+hoz_road+
                         hill_9am+hill_noon+hill_3pm+hoz_fire,
```

```
                    data=newtree,nvmax=10,method="backward")
summary(regfit.bkwd)


# ------------------------------------
# naive multinomial regression - mlogit
# ------------------------------------
library(mlogit)
multi.mod1 <- mlogit(covertype~elevation+aspect+slope+hoz_hydro+vert_hydro+hoz_road+
                     hill_9am+hill_noon+hill_3pm+hoz_fire
                   ,data=newtree)
summary(multi.mod1)
exp(coef(multi.mod1))

z <- summary(multi.mod1)$coefficients/summary(multi.mod1)$standard.errors

#2-tailed z test
p <- (1 - pnorm(abs(z), 0, 1))*2
p
```

Modeling Codes:

```
##############################################################################
######### LDA ############
library(MASS)

#LDA - model 1
lda.fit=lda(covertype~elevation+aspect+slope+vert_hydro+hoz_road+hill_3pm+hoz_fire,data=tr
ain_smpl)
lda.fit


#LDA - model 2
lda.fit2=lda(covertype~.,data=train_smpl)
lda.fit2
#plot(lda.fit2)


#LDA model stats and graph
names(lda.fit)  #available information from LDA model
lda.fit$prior   #prior probabilities based on distribution of covertype level/total db observations
lda.fit$counts  #count of each covertype level
plot(lda.fit, type="both")

#prediction code on training set data
lda.pred.train= predict(lda.fit,train_smpl) #predict covertype for test data
lda.class.train=lda.pred.train$class # contains predictions about cover type from lda.fit

#accuracy/misclassification rate on training set data
lda.accuracy = mean(lda.class.train==train_smpl$covertype)
# 0.6695319


#prediction code on test data
lda.pred.test= predict(lda.fit,test) #predict covertype for test data
lda.class.test=lda.pred.test$class # contains predictions about cover type from lda.fit

#confusion matrix
table(lda.class.test,test$covertype) #shows table matrix over all covertype levels

#accuracy/misclassification rate
lda.accuracy = mean(lda.class.test==test$covertype) #numb of correctly predicted values/ total
predictions
lda.accuracy

par("mar") # check margin size
```

```
par(mar=c(1,1,1,1)) # set margins so not too large
#Covertype Separation by Linear Discriminant 1
# x = linear discriminant number (max we can have is # of groups -1, so 6 in this case)
# overlapping areas show areas of misclassification
ldahist(data = lda.pred.test$x[,1], g=test$covertype, xlab="Covertype", ylab="% of class",
main="Covertype Separation by Linear Discriminant 1")

# LDA1 = 0.671266 - 7 backward selected variables
# LDA2 = 0.682201 - all 63 variables
```
######### END LDA RESULTS ###########

```
##############################################################################
```
######### QDA ###########
```
#QDA - model 1
qda.fit1=qda(covertype~elevation+aspect+slope+vert_hydro+hoz_road+hill_3pm+hoz_fire,data
=train_smpl)
qda.fit1

#prediction code for training set
names(qda.fit1)     #available information from QDA model
qda.pred.train= predict(qda.fit1,test) #predict covertype for test data
qda.class.train=qda.pred.train$class # contains predictions about cover type from lda.fit

#confusion matrix
table(qda.class.train,test$covertype) #shows table matrix over all covertype levels

#accuracy/misclassification rate
mean(qda.class.train==test$covertype) #numb of correctly predicted values/ total predictions
#QDA1 = 0.6631421 - 7 backward selection variables
#QDA1 train = 0.663336 - 7 backward selection variables
```
######### END QDA RESULTS ###########

```
##############################################################################
```
######### KNN ###########
```
library(class)
train_smpl.X=cbind(train_smpl$elevation,train_smpl$aspect,train_smpl$slope,train_smpl$vert_
hydro,train_smpl$hoz_road,train_smpl$hill_3pm,train_smpl$hoz_fire)
test.X=cbind(test$elevation,test$aspect,test$slope,test$vert_hydro,test$hoz_road,test$hill_3pm,t
est$hoz_fire)
train_smpl.response=train_smpl$covertype
test.response=test$covertype
```

```
# standardize independent training variables to compensate for different scales used for each
variable
standardized.X.train_smpl=scale(train_smpl.X)
standardized.X.test=scale(test.X)


# determine best k value to use for test set
rm(mean.error)
set.seed(71)
mean.error=rep(0,20)
for(i in 1:20){
  knn.pred=knn(standardized.X.train_smpl,standardized.X.test,train_smpl.response,k=i)
  mean.error[]=mean(knn.pred==test.response)
 }
mean.error
#0.8306560<-k=2,  0.8459054<-k=3,  0.8363071<-k=4,  0.8403116 <-k=5,
#0.8344368 <-k=7,  0.8254352 <-k=10,  0.8162786 <-k=15,  0.8070533 <-k=20
plot(mean.error, type="o", col="blue", main="KNN - predicted vs. test response",
     xlab="K value", ylab="Mean percent correct")


#knn graph of results
knn.results.kval=c(1,2,3,4,5,7,10,15,20) #k-values used in model
knn.results.err <- c("0.852549","0.8306560","0.8459054","0.8363071","0.8403116",
                  "0.8344368","0.8254352","0.8162786","0.8070533") #predictive error on test data
knn.results <- cbind(knn.results.kval,knn.results.err)
plot(knn.results, type="o", col="blue", main="KNN - predicted vs. test response",
    xlab="K value", ylab="Mean Percent Correct")

# Best KNN used K=1 at .8525% accuracy
# 2nd best KNN  K=3 at .8459% accuracy

######### END KNN RESULTS ###########



#############################################################################
#-------------------------------------------------
  1. RANDOM FOREST - TRADITIONAL
 # -------------------------------------------------
rf.model <- randomForest(covertype~.,data=train_smpl,probability=TRUE)

print(rf.model)
plot(rf.model)
importance(rf.model)
```

```
varImpPlot(rf.model,main="Variable Importance Plot")

rf.model1 <-
randomForest(covertype~elevation+elevation_comp+wild2_comp+hoz_road+soil4_comp+
hoz_fire_comp  +hoz_fire+wild1_comp+hoz_vert_hydro_comp+hoz_hydro+
hillnoon_comp+vert_hydro+aspect+wild4+aspect_comp+hill_noon+
hill_9am+hill_3pm,data=train_smpl, probability=TRUE)
par(mfrow=c(1,2))
plot(rf.model)
plot(rf.model1)
print(rf.model1)
importance(rf.model1)
varImpPlot(rf.model1,main="Variable Importance Plot")

# ---------------------
 in-sample MSE
# ---------------------
rf.predict <- predict(rf.model1,type="response")
mean(rf.predict != train_smpl$covertype)
table(train_smpl$covertype,rf.predict,dnn=c("Actual","Predicted"))


# -------------------------
 out-sample MSE
# -------------------------
rf.predict2 <- predict(rf.model1,newdata=test,type="response")
mean(rf.predict2 != test$covertype)
table(test$covertype,rf.predict2,dnn=c("Actual","Predicted"))

postResample(rf.predict2,test$covertype)


# ------------------------------------------------
 2. RANDOM FOREST - tune function
# ------------------------------------------------

#---------------
prep file
#---------------
names(train_smpl)
x <- train_smpl[,1:10]
y <- train_smpl[,55]

tunerf.model <- tuneRF(x,y,improve=0.05,trace=FALSE)
```

```
tunerf.model2 <-
randomForest(covertype~elevation+elevation_comp+wild2_comp+hoz_road+soil4_comp+hoz_f
ire_comp

+hoz_fire+wild1_comp+hoz_vert_hydro_comp+hoz_hydro+hillnoon_comp+vert_hydro
                    +aspect+wild4+aspect_comp+hill_noon+hill_9am+hill_3pm,mtry=6,
                    data=train_smpl,probability=TRUE)

print(tunerf.model2)
plot(tunerf.model2)
importance(rf.model)
varImpPlot(tunerf.model2,main="Variable Importance Plot")

# ---------------------
# in-sample MSE
# ---------------------
tunerf.predict <- predict(tunerf.model2,type="response")
mean(tunerf.predict != train_smpl$covertype)
table(train_smpl$covertype,tunerf.predict,dnn=c("Actual","Predicted"))

# --------------------------
# out-sample MSE
# -------------------------
tunerf.predict2 <- predict(tunerf.model2,newdata=test,type="response")
mean(tunerf.predict2 != test$covertype)
table(test$covertype,tunerf.predict2,dnn=c("Actual","Predicted"))

postResample(tunerf.predict2,test$covertype)



#############################################################################
######### GRADIENT BOOSTING ###########
library(gbm) #gradient boosting
library(caret) #tunehyper-parameters


#list various parameters to test to optimize gbm model
grid = expand.grid(.n.trees=seq(100,500, by=200),
            .interaction.depth=seq(1,4, by=1), .shrinkage=c(.001,.01,.1),
            .n.minobsinnode=10)

#train the tuning parameters to reduce bias and variance
control = trainControl(method="CV", number=10)

#gbm model 1 - uses all predictors and response covertype
```

```
set.seed(71)
gbm.train.full = train(covertype~., data=small.train_smpl, method="gbm",
                       trControl=control, tuneGrid=grid)
#view optimum combination of parameters to use
gbm.train1  #500trees, 4depth, .01shrinkage - backward selected variables
gbm.train.full  #100, 4depth, .1shrikage - all variables

#must change distribution type to "multinomial" for >2 classes in classification
set.seed(71)
gbm.fit.bkwd =
gbm(covertype~elevation+aspect+slope+vert_hydro+hoz_road+hill_3pm+hoz_fire,
            data=train_smpl, distribution="multinomial", n.trees=500,
            interaction.depth=4, shrinkage=0.01)

#shows variable importance values and bar graph
summary(gbm.fit.full)
summary(gbm.fit1)
summary(gbm.fit.bkwd)

#prediction code on test data set
gbm.predict1 = predict(gbm.fit.full, type="response", n.trees=500)

#class designation for prediction code
gbm.class <- ifelse(gbm.predict1<.14285714, 1,
           ifelse(gbm.predict1 >= .14285714 & gbm.predict1 < .28571428, 2,
             ifelse(gbm.predict1 >= .28571428 & gbm.predict1 < .42857142, 3,
               ifelse(gbm.predict1 >= .42857142 & gbm.predict1 < .57142856, 4,
                 ifelse(gbm.predict1 >= .57142856 & gbm.predict1 < .7142857, 5,
                   ifelse(gbm.predict1 >= .7142857 & gbm.predict1 < .85714284, 6, 7))))))


#accuracy rate - all variables
table(test$covertype,gbm.class,dnn=c("Actual","Predicted"))
accuracy.fit.full = mean(gbm.class!=test$covertype)
accuracy.fit.full #0.6974086 - test

table(train_smpl$covertype,gbm.class,dnn=c("Actual","Predicted"))
accuracy.fit.train = mean(gbm.class!=train_smpl$covertype)
accuracy.fit.train #0.6974106 - train


#accuracy rate - backward selected variables
table(test$covertype,gbm.class,dnn=c("Actual","Predicted"))
accuracy.fit1 = mean(gbm.class!=test$covertype)
accuracy.fit1 #0.7001239 - test
```

```
table(train_smpl$covertype,gbm.class,dnn=c("Actual","Predicted"))
accuracy.fit1.train = mean(gbm.class!=train_smpl$covertype)
accuracy.fit1.train #0.7036361 – train
```
######### END GRADIENT BOOSTING RESULTS ###########

```
#############################################################################
# ----------------------
# SVM
# ----------------------
library(e1071)
library(ROCR)
set.seed(453)

load("/Users/sungpark/Desktop/PREDICT 454/Project/Data/modeltrees_v2.RData")
sapply(train_smpl,class)   #look at data types
sapply(test,class)


# ------------------
# 1 - linear kernel
# ------------------
#linear.svm <- svm(covertype~.-soil7-soil15-soil21-soil25-soil36, data=train, kernel="linear",
scale=TRUE)
#summary(linear.svm)


# cross-validation linear kernel
set.seed(1)
linear.svm.cv = tune(svm,covertype~.-soil7-soil15-soil21-soil25-soil36, data=train_smpl,
kernel="linear",
                range=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))

summary(linear.svm.cv) # best model is cost = 1
bestmod = linear.svm.cv$best.model
summary(bestmod)

# ---------------------
# in-sample MSE
# ---------------------
linear.trainMSE <- predict(bestmod,type="response")
mean(linear.trainMSE != train_smpl$covertype)
table(train_smpl$covertype,linear.trainMSE,dnn=c("Actual","Predicted"))

# ---------------------
# out-sample MSE
```

```
# ---------------------
linear.testMSE <- predict(bestmod,newdata=test,type="response")
mean(linear.testMSE != test$covertype)
table(test$covertype,linear.testMSE,dnn=c("Actual","Predicted"))
# ------------------------------------------------
# ------------------
# 2 - radial kernel
# ------------------
#radial.svm <- svm(covertype~.,data=svm_train,kernel="radial")

# cross-validation radial kernel
set.seed(1)
radial.svm.cv = tune(svm,covertype~.-soil7-soil15-soil21-soil25-soil36, data=train_smpl,
kernel="radial",
                range=list(cost=c(0.1, 1, 10, 100, 1000)),
                gamma=c(0.5,1,2,3,4))
summary(radial.svm.cv) # best model is cost = 10
bestmod.radial = radial.svm.cv$best.model
summary(bestmod.radial)

# ---------------------
# in-sample MSE
# ---------------------
radial.trainMSE <- predict(bestmod.radial,type="response")
mean(radial.trainMSE != train_smpl$covertype)
table(train_smpl$covertype,radial.trainMSE,dnn=c("Actual","Predicted"))

# ---------------------
# out-sample MSE
# ---------------------
radial.testMSE <- predict(bestmod.radial,newdata=test,type="response")
mean(radial.testMSE != test$covertype)
table(test$covertype,radial.testMSE,dnn=c("Actual","Predicted"))
```

```
################################################
##### NNET - Forest Cover#####################
################################################

## Paul DellaGrotte
## Modified: 2/7/2016


library(neuralnet)
#library(carat)
set.seed(123)

names(train_smpl)   #56-63, 55 covertype

nnet_covertype <- train_smpl

#Binarize the categorical output
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '1')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '2')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '3')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '4')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '5')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '6')
nnet_covertype <- cbind(nnet_covertype,train_smpl$covertype == '7')
names(nnet_covertype)

# create binary variables for class covertype
names(nnet_covertype)[60] <- 'class1'
names(nnet_covertype)[61] <- 'class2'
names(nnet_covertype)[62] <- 'class3'
names(nnet_covertype)[63] <- 'class4'
names(nnet_covertype)[64] <- 'class5'
names(nnet_covertype)[65] <- 'class6'
names(nnet_covertype)[66] <- 'class7'
names(nnet_covertype)

#only select appropriate variables
train_smpl2 <- nnet_covertype[,56:66]
names(train_smpl2)
################################################

################################################
##### Model 1 - learning rate=0.05 ############
################################################

f <- as.formula(paste("class1+class2+class3+class4+class5+class6+
```

```
                class7~hill3pm_comp+hillnoon_comp+hozvert_comp+hozroad_comp"))

ptm <- proc.time()  # start timer

nn1 <- neuralnet(f,data=train_smpl2,hidden=2,learningrate=0.05,
            err.fct="ce",linear.output=FALSE, stepmax=300000)

proc.time() - ptm   # Output time elapsed

################################################
##### Model 2 - learning rate=0.35 ############
################################################


ptm <- proc.time() # start timer


nn2 <- neuralnet(f,data=train_smpl2,hidden=2,learningrate=0.35,
             err.fct="ce",linear.output=FALSE, stepmax=300000)

proc.time() - ptm  # Output time elapsed

################################################
### Compute Model 1 Results on Testing Set ####
################################################

nn1_predict_test <- data.frame(compute(nn1, test[ , 56:59]))
nn1_predict_test <- nn1_predict_test[ ,9:15]


varNames <- c("1", "2", "3", "4", "5", "6", "7")

names(nn1_predict_test) <- varNames

nn1_predict_test$covertype <- colnames(nn1_predict_test)[max.col(nn1_predict_test,
                                        ties.method="first")]

# confusion Matrix of test
table(nn1_predict_test$covertype, test$covertype)

################################################
### Compute Model 1 Results on Training Set ###
################################################

nn1_predict_train <- data.frame(compute(nn1, train_smpl2[ , 1:4]))
```

```
nn1_predict_train <- nn1_predict_train[ ,9:15]

varNames <- c("1", "2", "3", "4", "5", "6", "7")
names(nn1_predict_train) <- varNames # Change column names for class

# Take highest weight and compute variable column (class)
nn1_predict_train$covertype <- colnames(nn1_predict_train)[max.col(nn1_predict_train,
                                        ties.method="first")]

# confusion matrix of train
table(train_smpl$covertype, nn1_predict_train$covertype)


################################################
### Compute Model 2 Results on Testing Set ####
################################################

nn2_predict_test <- data.frame(compute(nn2, test[ , 56:59]))
nn2_predict_test <- nn2_predict_test[ ,9:15]

varNames <- c("1", "2", "3", "4", "5", "6", "7")

names(nn2_predict_test) <- varNames

nn2_predict_test$covertype <- colnames(nn2_predict_test)[max.col(nn2_predict_test,
                                        ties.method="first")]

# confusion Matrix of test
table(nn2_predict_test$covertype, test$covertype)

################################################
### Compute Model 2 Results on Training Set ###
################################################

nn2_predict_train <- data.frame(compute(nn2, train_smpl2[ , 1:4]))
nn2_predict_train <- nn2_predict_train[ ,9:15]

varNames <- c("1", "2", "3", "4", "5", "6", "7")
names(nn2_predict_train) <- varNames # Change column names for class

# Take highest weight and compute variable column (class)
nn2_predict_train$covertype <- colnames(nn2_predict_train)[max.col(nn2_predict_train,
                                        ties.method="first")]

# confusion matrix of train
table(nn2_predict_train$covertype, train_smpl$covertype)
```