# Forest Cover Prediction Using Logistic Regression

*Tom Thorpe*

*August 29, 2018*

## Introduction

The Forest Cover Capstone Project applies techniques learned in the Springboard Data Science Foundation Course to predict the type of tree coverage in four different wilderness areas in Colorado.

The data used comes from the Ph.D. dissertation by Jock Blackard, 1998, "Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types.", Department of Forest Sciences, Colorado State University, Fort Collins, Colorado.

The data can be found at: https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/.

The Forest Coverage predictor is used by the US Forest Service to "support the decision-making processes for developing ecosystem management strategies." Improving the accuracy of this predictor could help the Forest Service improve their planning.

Dr. Blackard found that neural networks were able to predict forest coverage with 70% accuracy. This was a 12% improvement over the Discriminant Analysis methods currently in use which had an accuracy of 58%. This project will see how logistic regression prediction compares to the neural network.

The files comprising the project are described in the README.md file including the full source code used for data exploration, data cleaning and predicting forest coverage types.

## Data

The forest cover data is described next including data validation and conversion to the alternate coding scheme.

### Data Description

The forest coverage data consist of 581,012 rows and 54 columns including elevation, aspect (the compass direction the land is facing), slope, amount of shade at 9am, noon and 3pm, distance in meters to water and roads, wilderness area, soil types and the type of tree found in the 30 meter by 30 meter sample. Only one of seven possible tree types is possible per row.

The possible tree types are:

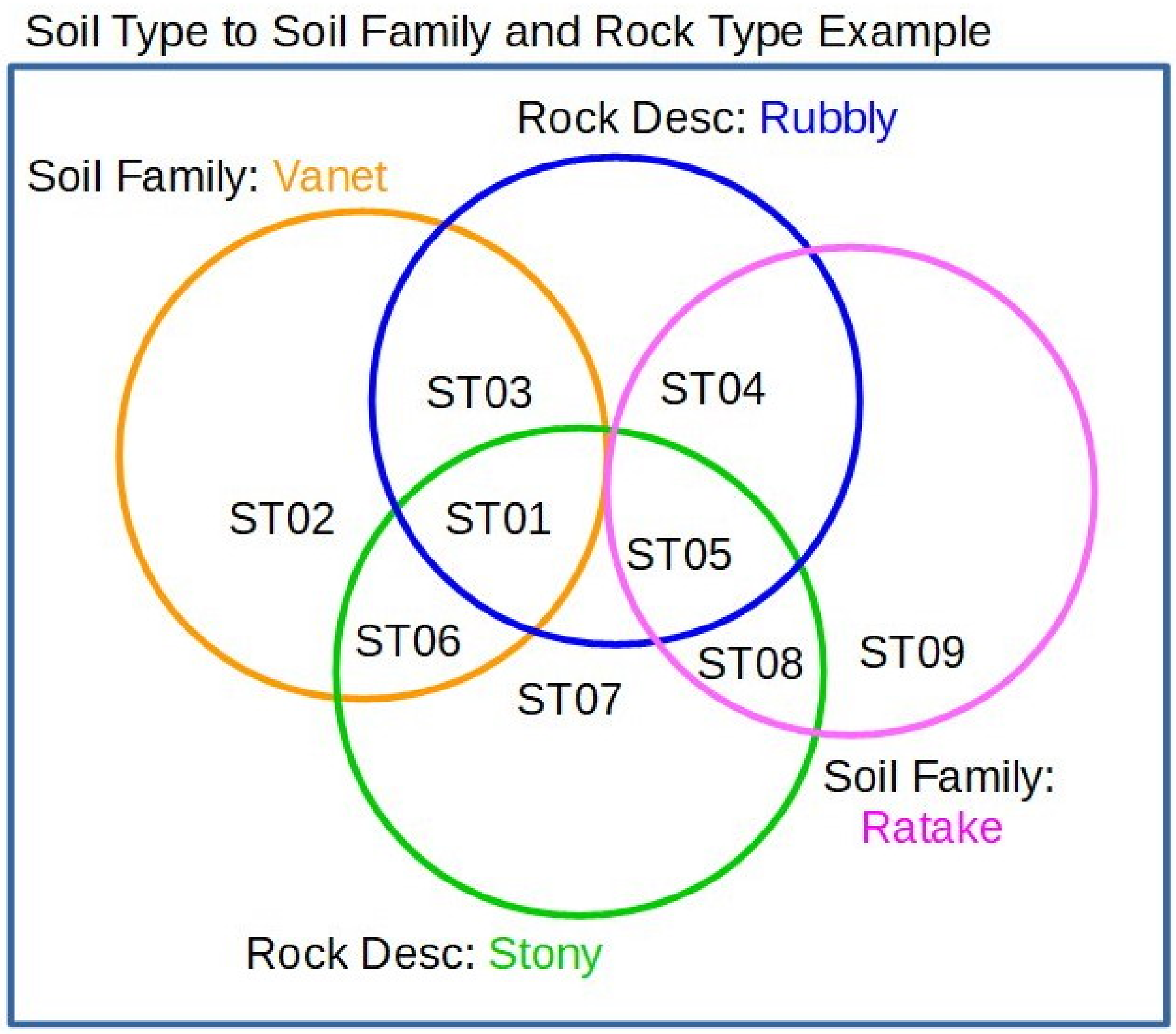
1. Aspen
2. Cottonwood / Willow
3. Douglas Fir
4. Krummholz
5. Lodgepole Pine
6. Ponderosa Pine
7. Spruce / Fir

The soil types are split into 40 columns with only one of the 40 columns having a value of '1' and all the other soil type columns having a value of '0'. Each column represents the unique intersection of soil features that could be mapped onto a shape within a multidimensional Venn Diagram.

The four dimensions of the Venn Diagram for soil type include:

- Climate Zone - 8 types, for example: montane, montane dry, sub alpine, alpine, etc.
- Geologic zone - 8 types, for example: alluvium, glacial, shale, sandstone, etc.
- Soil Family - 49 types, for example: Vanet, Ratake, Bullwark, Gateview, Rogert, Leighcan, etc.
- Rock Type - 10 types, for example: rubbly, stony, very stony, rocky, very rocky, etc.

The "Soil Family and Rock Type to Soil Type" figure shows how two Soil Families (Vanet and Ratake) and two Rock Types (rubbly and stony) could combine to create different Soil Types based on their intersections. Note: these are made up for illustration purposes only and only two the four dimensions are illustrated.



Soil Type ST01 is comprised of Soil Family Vanet and a combination of Rock Types Stony and Rubbly. Soil Type ST04 is comprised of Soil Family Ratake and Soil Type Rubbly.

## Alternate Investigation

In addition to exploring the data encoding used by Dr Blackard, the soil types are broken out into columns for the dimensions of the Venn Diagram. This results in eight climate zone columns with one climate zone selected per row and eight geologic zone columns with one geologic zone selected per row.

In the current encoding a soil type represents one or more soil families. The new encoding allows multiple soil families to be marked per row. Similarly, there can be multiple rock types in a soil type and multiple rock types could be selected per row.

For example, The description of soil type 33 is "Leighcan - Catamount families - Rock outcrop complex, extremely stony." The Leighcan soil family column and Catamount soil family column would be selected. The "Rock outcrop complex" column and the "extremely stony" column would be selected for the Rock Type dimension.

The alternate data representation will be compared with the original encoding to determine if breaking the soil types into it's discrete parts helps or hinders the ability to accurately predict the type of forest coverage.

A glimpse of the data is shown below.

```
glimpse(forestcover)
## Observations: 581,012
## Variables: 55
## $ Elev <int> 2596, 2590, 2804, 2785, 2595, 2579, 2606, 2605, 2617...
## $ Aspect <int> 51, 56, 139, 155, 45, 132, 45, 49, 45, 59, 201, 151,...
## $ Slope <int> 3, 2, 9, 18, 2, 6, 7, 4, 9, 10, 4, 11, 22, 7, 4, 7, ...
## $ H2OHD <int> 258, 212, 268, 242, 153, 300, 270, 234, 240, 247, 18...
## $ H2OVD <int> 0, -6, 65, 118, -1, -15, 5, 7, 56, 11, 51, 26, 69, 4...
## $ RoadHD <int> 510, 390, 3180, 3090, 391, 67, 633, 573, 666, 636, 7...
## $ Shade9AM <int> 221, 220, 234, 238, 220, 230, 222, 222, 223, 228, 21...
## $ Shade12PM <int> 232, 235, 238, 238, 234, 237, 225, 230, 221, 219, 24...
## $ Shade3PM <int> 148, 151, 135, 122, 150, 140, 138, 144, 133, 124, 16...
## $ FirePtHD <int> 6279, 6225, 6121, 6211, 6172, 6031, 6256, 6228, 6244...
## $ RWwild <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ NEwild <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CMwild <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CPwild <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ST01 <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ST02 <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
...
## $ ST39 <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ST40 <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CovType <int> 5, 5, 2, 2, 5, 2, 5, 5, 5, 5, 5, 2, 2, 5, 5, 5, 5, 5...
```

## Data Cleaning

The data is validated for valid ranges and what data is missing and how to handle it.

First, the non-binary data is checked for valid ranges. The following function is applied to the binary data to determine it's ranges.

```
myranges <- function(name,x) { c(name, min = min(x), mean = mean(x), median=median(x), max = max(x)) }

forestDataRanges <- data.frame("Data"=character(), "min"=double(), "mean"=double(),
"median"=double(), "max"=double(), stringsAsFactors=FALSE)
```

```
forestDataRanges[nrow(forestDataRanges)+1,] <- myranges("Elev",forestcover$Elev)
```

When applied to each of the integer data types, this gives:

```
##         Data  min               mean median  max
## 1       Elev 1859   2959.36530054457   2996 3858
## 2     Aspect    0   155.656807432549    127  360
## 3      Slope    0   14.1037035379648     13   66
## 4      H2OHD    0   269.428216628916    218 1397
## 5      H2OVD -173    46.418855376481     30  601
## 6     RoadHD    0   2350.14661142971   1997 7117
## 7   FirePtHD    0     1980.291226343   1710 7173
## 8   Shade9AM    0   212.146048618617    218  254
## 9   Shade12P    0   223.318716308785    226  254
## 10 Shade3PM    0    142.52826275533    143  254
## 11   RWwild    0 0.448865083681576      0    1
## 12   NEwild    0 0.051434393781884      0    1
## 13   CMwild    0 0.436073609495157      0    1
## 14   CPwild    0 0.063626913041383      0    1
```

The results show all the data values have reasonable values and there is no missing data. The elevation ranges from 1859 meters (6099 feet) to 3858 meters (12657 feet). These are valid ranges for elevation in the Colorado wilderness areas being sampled, but the rule of thumb for timberline (the maximum elevation for where trees are found) is 11500 feet. It might be interesting to see how accurate predictions are if samples above 11800 feet are removed.

The Aspect which is the compass heading that the terrain faces, ranges from 0 to 360 degrees and is a valid data range. The Slope is the steepness of the terrain with 0 degrees being flat and 90 degrees being vertical. The maximum Slope was found to be 66 degrees which seems logical since trees are not usually seen on near-vertical cliffs. (It's a different story in New Zealand!)

The horizontal distance to the nearest water features, range from 0 to 1397 meters which seems reasonable. The vertical distance to nearest water features, range from -173 to 601 meters which also seems reasonable and can be negative since the nearest water may be below the forest cover data sample.

The horizontal distance to the nearest road ranges from 0 to 7117 meters which is reasonable. The horizontal distance to the nearest fire features range from 0 to 7173 meters which is reasonable. The amount of shade present in a cell sample at 9AM, 12PM and 3PM ranges from 0 (full sun) to 254 (fully shaded).

## Check Soil Type encoding

Check the binary data to ensure multiple columns have not been selected. Starting with Soil Type, check that there is no more than one ST___ column set to 1 in each row.

```
  STcols=c("ST01","ST02","ST03","ST04","ST05","ST06","ST07","ST08","ST09","ST10",
           "ST11","ST12","ST13","ST14","ST15","ST16","ST17","ST18","ST19","ST20",
           "ST21","ST22","ST23","ST24","ST25","ST26","ST27","ST28","ST29","ST30",
           "ST31","ST32","ST33","ST34","ST35","ST36","ST37","ST38","ST39","ST40"
           )
   forestcover <- mutate(forestcover,STsum=rowSums(forestcover[,STcols]))
   myranges("STsum",forestcover$STsum)
```

```
##             min    mean  median     max
## "STsum"     "1"     "1"     "1"     "1"
```

```r
  # print the row numbers where the STsum is not 1
  which(forestcover$STsum!=1)
```

```
## integer(0)
```

The soil type data is clean. There are no rows where the Soil Type columns do not have exactly one column set to 1.

Using the same method, the Wilderness indicators were checked and each row had exactly one column set to 1.

The data has been verified as clean and no missing values.

## Expand Soil type

Next, the soil type needs to be expanded into columns that comprise the different components of each soil type. A particular soil type represents the climate zone, geologic zone, one or more soil families and one or more rock types for a given sample/row.

A transform data set was manually created to translate the Soil Type to soil type components. Each row in the transform data set corresponds to one of the 40 possible soil types in the forest coverage data frame. The first 3 columns identify the soil type number that corresponds to the soil type column in the forest coverage data frame, a US Forest Service soil code and a description of the soil families and rock types for the soil type. The remaining columns are the columns that correspond to the soil type components that will be added to the forest coverage data frame and the values for each column. The columns added to the forest column data frame correspond to the individual components in the description of each soil type.

A data set was used to drive the soil type translation so that changes could be made easily without having to change code. Column names added to the forest cover data set are specified by the transform data set.

The transform data set, labeled xform in the code, is sorted by ST, the soil type column, so that it can be indexed by the soil type index that will be created from the forest coverage soil type columns data.

We want to add the same soil data column names in the xform data frame to the forest coverage data frame. Unfortunately I was not able to find a way to use a variable name to assign the name of the new column when using the *mutate()* function. We can use the *colnames* function to change the column name after each mutate operation adds a column to the forest coverage data frame.

Start by getting the current forest coverage column names and creating an empty vector to collect the xform data frame column names.

Next iterate through the column names in the xform data frame. The first three columns in the xform data frame are not to be added to the coverage data frame. These column names are skipped by checking for the column names as shown in the first *if* statement below. This is the only hard coding required and the only requirement of column layout in the xform data frame.

For every other xform data frame column, a column is added to the forest coverage data frame. The xform column name is added to the xform data frame column names and the forest service column names vectors. After the column is added to the forest coverage data frame, the forest service column names vector will be used to reset the new column name. The xform column name vector will be used later to index both the xform and forest coverage data frames when setting values in the forest coverage data frame.

```r
for(colname in colnames(xform)){
  if (colname!="ST" & colname !="USFS_Code" & colname != "Description")
  {
    #print(colname)
    forestcover <- mutate(forestcover,colname = 0) # add column named "colname" to forestcover
    forestcnames<-c(forestcnames,colname)          # add the actual column name to forest colnames vect
    colnames(forestcover) <- forestcnames           # set the forest cover column names
    xformcnames=c(xformcnames,colname)              # add the column name to xform column names vector
  }
}
```

The new columns have been added to the forest coverage data frame and the column values need to be populated based on the values in the xform data frame.

Each row in the transform data frame is processed. Each row represents a soil type. If a data column in the transform soil type is set to ‘1’, the same data column should be set to ‘1’ in the forest cover data frame, for the same soil type.

## Expand Soil Type in to Components

Go through every row/soil type in the transform data frame to determine which columns in the forest coverage data frame must be set. The code to do this is shown below.

```
  for(ndx in 1:nrow(xform)) {
    # if a property is set for the current soiltype (the ndx variable),
    # set the same property in forest cover for that soil type
    forestcover$ClimateZone[forestcover$SoilType == ndx] <- xform$ClimateZone[ndx]
    forestcover$GeoZone[forestcover$SoilType == ndx] <- xform$GeoZone[ndx]

    # Set Climate Zone
    if (xform[ndx,"Montane_low"] == 1) { forestcover$Montane_low[forestcover$SoilType == ndx] <- 1 }
    ...
    if (xform[ndx,"Alpine"] == 1) { forestcover$Alpine[forestcover$SoilType == ndx] <- 1 }

    # Set Geologic Zone
    if (xform[ndx,"Alluvium"] == 1) { forestcover$Alluvium[forestcover$SoilType == ndx] <- 1 }
    ...
    if (xform[ndx,"Glacial"] == 1) { forestcover$Glacial[forestcover$SoilType == ndx] <- 1 }

    # Set Soil Family
    if (xform[ndx,"Aquolis_cmplx"] == 1) { forestcover$Aquolis_cmplx[forestcover$SoilType == ndx] <- 1
    if (xform[ndx,"Argiborolis_Pachic"] == 1) { forestcover$Argiborolis_Pachic[forestcover$SoilType ==
    ...
    if (xform[ndx,"Vanet"] == 1) { forestcover$Vanet[forestcover$SoilType == ndx] <- 1 }
    if (xform[ndx,"Wetmore"] == 1) { forestcover$Wetmore[forestcover$SoilType == ndx] <- 1 }

    # Set Rock Type
    if (xform[ndx,"Rock_Land_cmplx"] == 1) { forestcover$Rock_Land_cmplx[forestcover$SoilType == ndx] <
    ...
    if (xform[ndx,"Till_Substratum"] == 1) { forestcover$Till_Substratum[forestcover$SoilType == ndx] <
  }
```

## Data Exploration

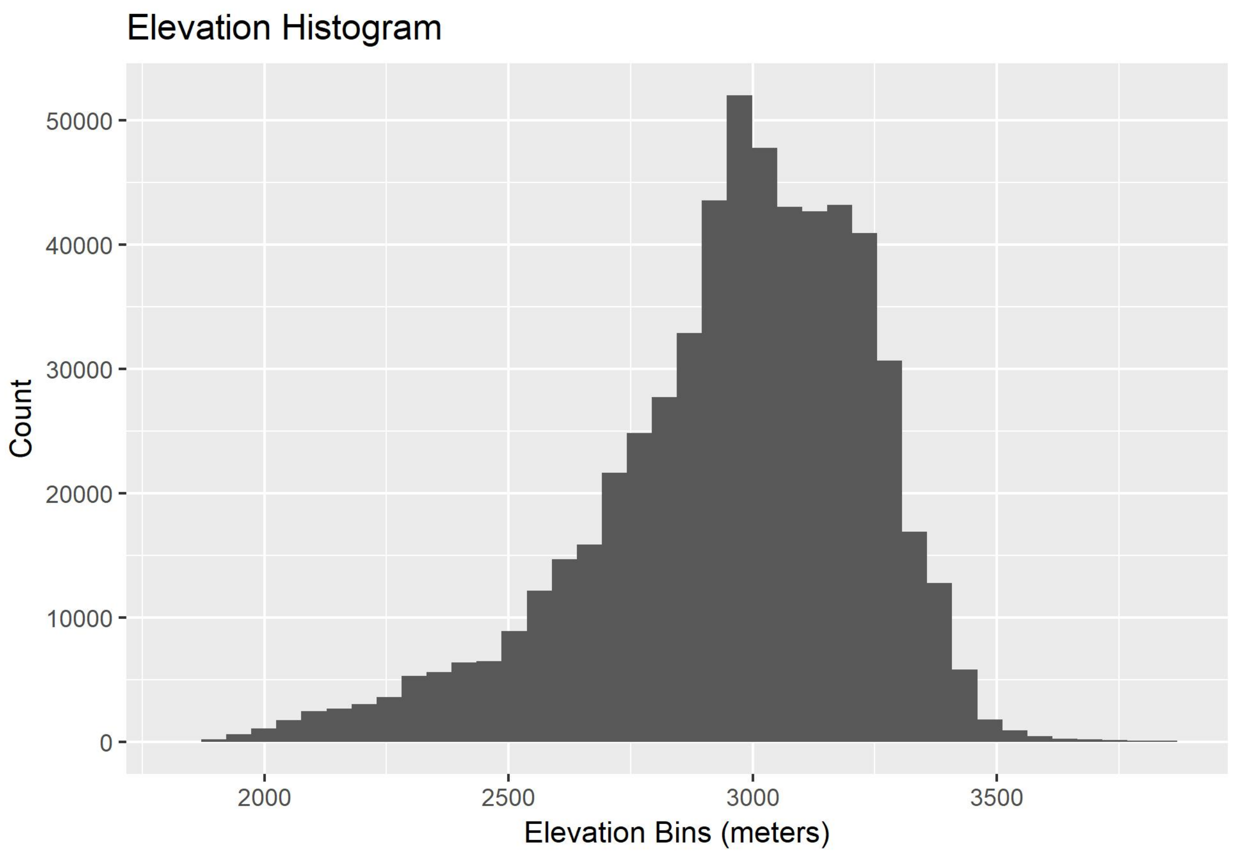
The forest cover data has been cleaned and transformed into an alternate coding. Before attempting logistic regression, lets explore the data.

A table showing the number of occurrences for each tree type is shown below.

```
##             Var1   Freq Percent
## 1         Aspen   9493     1.6
## 2 Cotton&Willow   2747     0.4
## 3    DouglasFir  17367     2.9
## 4     Krummholz  20510     3.5
## 5     Lodgepole 283301    48.7
## 6     Ponderosa  35754     6.1
## 7    Spruce&Fir 211840    36.4
```

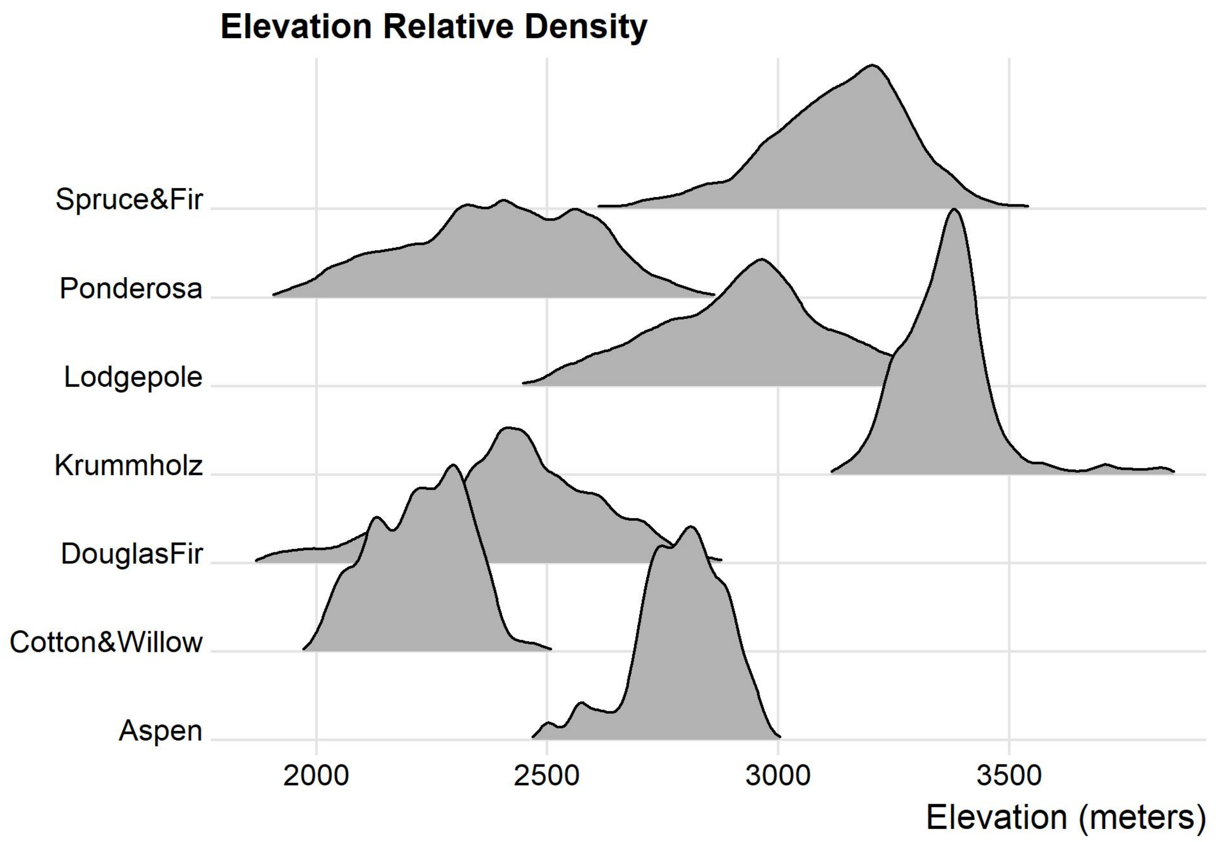
Lodgepole Pine represents 48.7 percent of the sample. So always guessing "Lodgepole" would provide success rate of 48.7 percent and can be used as a baseline for comparing our predictions. Spruce and Fir represent the next largest number of trees. The two together represent 85.1 percent.

**Elevation Histogram**



A good histogram for elevation is generated Using 40 bins. There are two humps in the histogram and there may be a more complicated distribution. The elevation may be related to other variables. Next the elevation is grouped by coverage type and wilderness to see how elevation relates to coverage type.

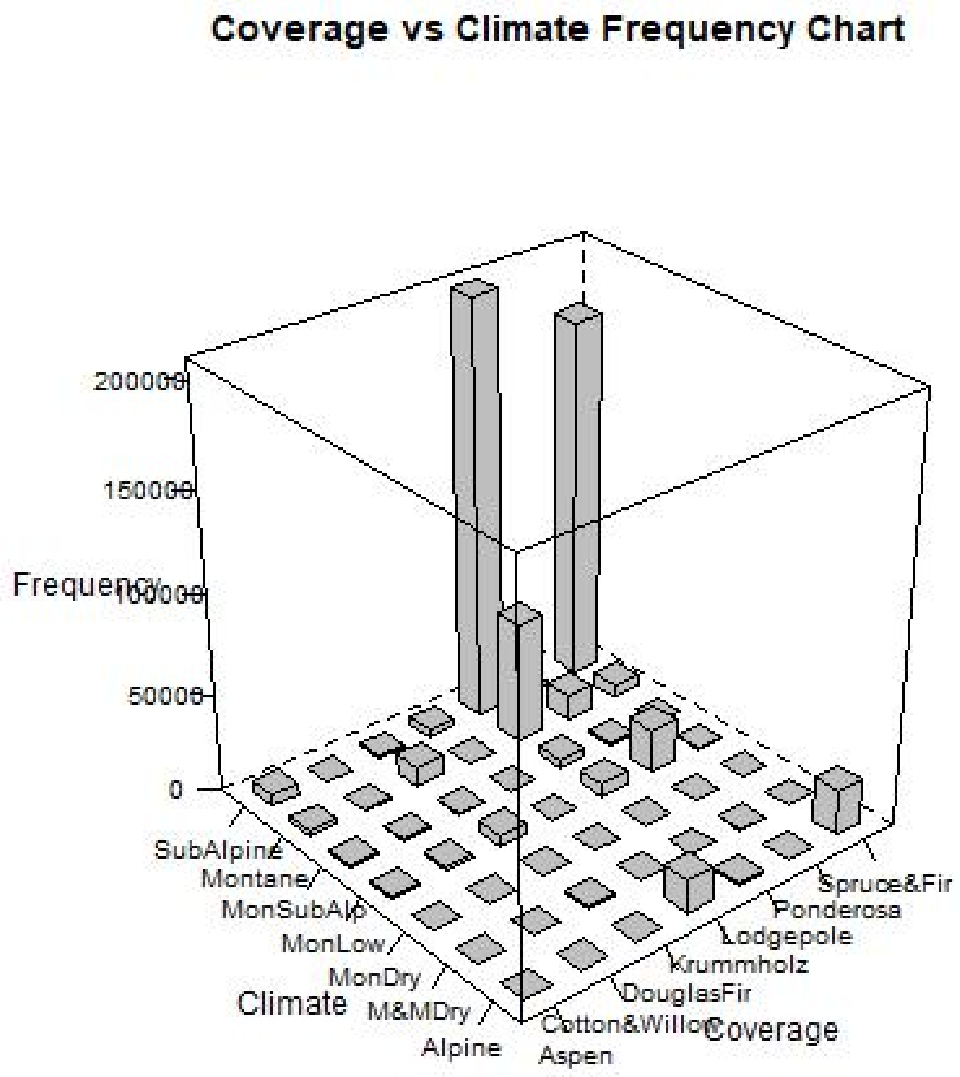
### Elevation Density



The density ridges geom gives a good feel for the ranges of elevation for each coverage type. It looks like the elevation is a significant factor in helping determine coverage type.

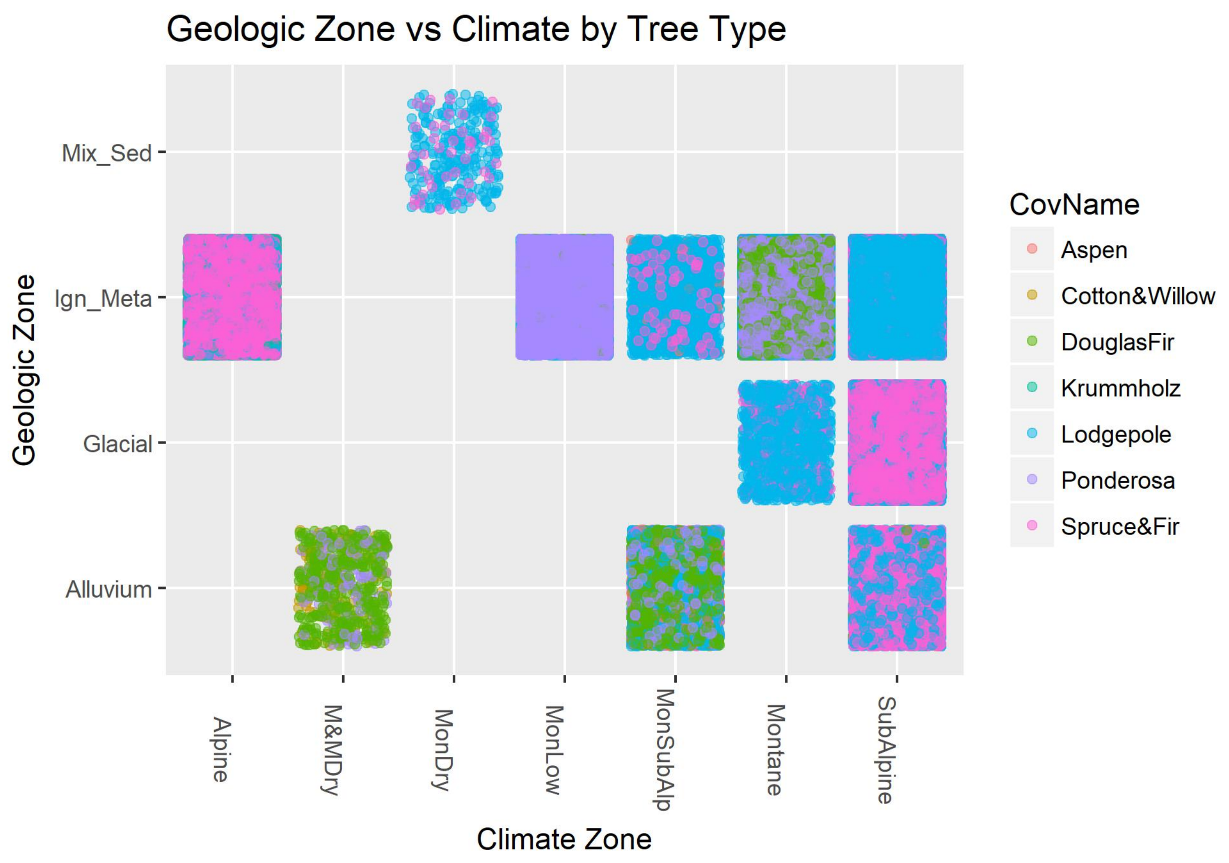
### Coverage vs Climate Frequency

```
## pdf
##   2
```



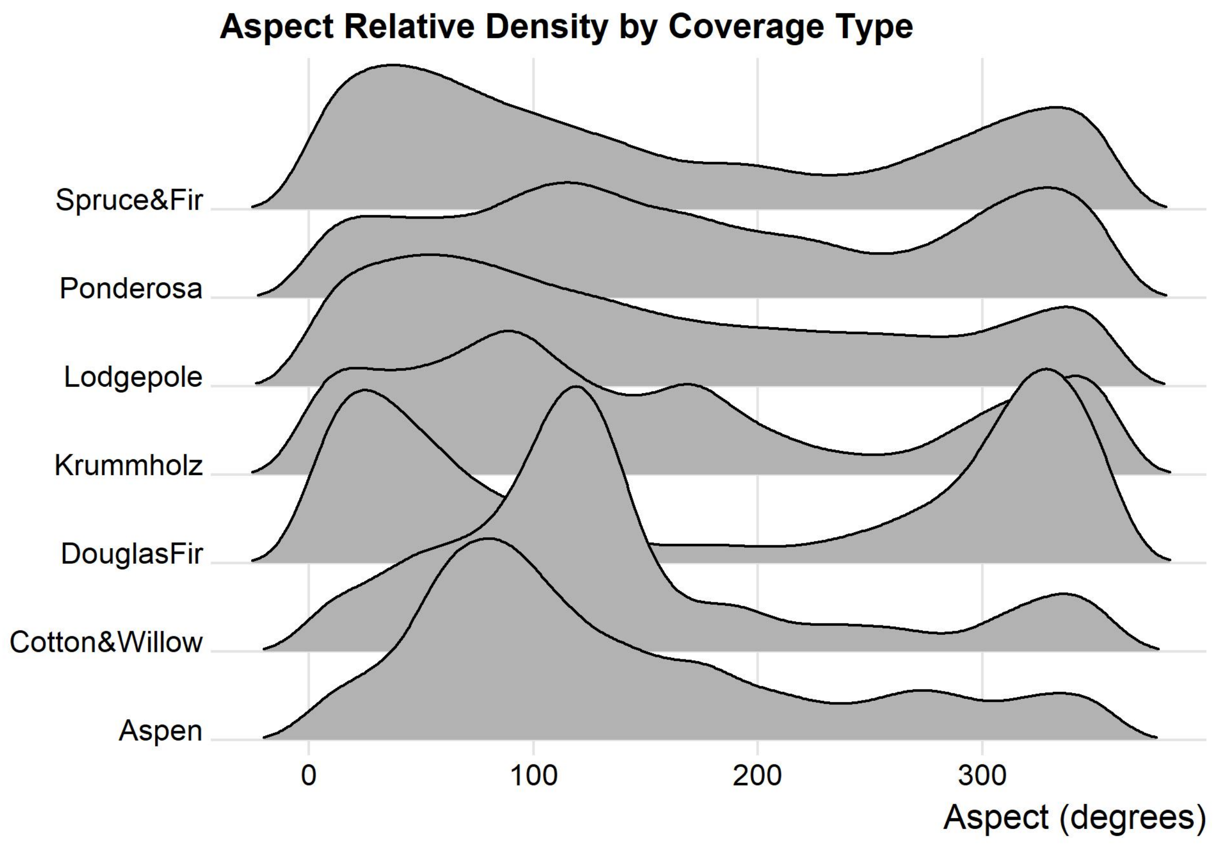
This gives a good view of the potential challenge to determine the various coverage types. The Lodgepole and Spruce & Fir trees make up the largest portion of the tree types. Determining the other tree types looks like they are in the “noise” of the data and might be more difficult to determine.

**Geologic Zone vs Climate with Tree Type**



Looking at the coverage type vs Climate and Geologic zones shows the two combinations may be helpful in determine coverage type but it is difficult to determine from this graph. The jitter geom was used to try to show the density, but the color coding is not distinct enough to get a feeling of the relative density of the tree coverage.

#### Aspect Relative Density vs Tree Type



Many other data were examined but did not suggest as clear a relationship to coverage type as the previous graphs. For example, the aspect of the slope (direction the slope of the cell faces) looks similar for each tree type. There are concentrations of tree types near aspects of 100 and 360 degrees. This occurs for all tree types and shows that the aspect will probably not be a significant factor in determining coverage type.

## Statistics Analysis

After looking at some of the data relationships graphically, some statistical tests are applied to the data to test if variables follow a normal distribution or are independent.

#### Shaprio Test - Elevation

The Elevation histogram looks like it possibly has a normal distribution. It is not perfect but might be close enough statistically.

The Shapiro test is used to determine if data is normally distributed. The maximum number of data points for this Shaprio test is 5000. A sample of the forest cover data set was extracted for the Shapiro test.

The Shapiro test result is shown below.

```
shapiro.test(altforestcover$Elev)
```

```
##
##  Shapiro-Wilk normality test
```

```
##
## data:  altforestcover$Elev
## W = 0.95909, p-value < 2.2e-16
```

The null hypothesis for the Shapiro test is that the data follows a normal distribution. If the P-value is less than the 0.05 significance level, the null hypothesis is rejected and the data is not considered to be normally distributed otherwise the data is normally distributed.

The P-value for elevation data is 2e-16 which is nearly zero and much less than 0.05, therefore the null hypothesis is rejected and the data is not normally distributed. The previous histogram shows this visually: The graph has a long left tail and a short right tail.

### Chi Square Test - Elevation & Coverage Name

It looks like elevation can be used to help identify coverage type. A chi-square test will be used to see if the coverage type and elevation variables are independent.

ElevSlot, "Elevation Slot" is used with Chi-square testing. It is calculated by diving the elevation by 100 and truncating the value by saving as an integer. This results in 21 elevation bins.

```
## Warning in chisq.test(table(forestcover$CovName, forestcover$ElevSlot), :
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(forestcover$CovName, forestcover$ElevSlot)
## X-squared = 763760, df = 120, p-value < 2.2e-16
```

If the P-value is less than significance factor of 0.05, the null hypothesis is rejected and the variables are not independent. The P-value is 2e-16 which is nearly zero. This shows that the coverage type and elevation are dependent, if the chi-square test is valid.

### Coverage Type vs Soil Type Independence check

The original paper used the Soil Type categories to predict coverage type. Let's try a chi-square test on them.

```
## Warning in chisq.test(table(forestcover$CovName, forestcover$SoilType), :
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(forestcover$CovName, forestcover$SoilType)
## X-squared = 762250, df = 234, p-value < 2.2e-16
```

If the P-value is less than significance factor of 0.05, the null hypothesis is rejected and the variables are not independent. The P-value is 2e-16 which is nearly zero. This shows that the coverage type and soil type are dependent, if the chi-square test is valid.

### Chi Square Test - Climate & Coverage Name Independence Test

The climate vs coverage type frequency graphs looked like there was a relationship between the two. A chi-square test on climate and coverage name is shown below.

```
## Warning in chisq.test(table(forestcover$CovName, forestcover
## $ClimateName), : Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(forestcover$CovName, forestcover$ClimateName)
## X-squared = 551740, df = 36, p-value < 2.2e-16
```

If the P-value is less than significance factor of 0.05, the null hypothesis is rejected and the variables are not independent. The P-value is 2e-16 which is nearly zero. This shows that the climate zone and coverage type are dependent, if the chi-square test is valid.

### Data Observations

There are many interesting data distributions in the continuous data. The elevation data seems to be the most easy to intuitively see a relationship predicting the outcome of the coverage type.

The other categorical data seems difficult to relate to outcome intuitively.

Statistically, chi-square testing indicates that both elevation and soil type are related to the coverage type.

It will be interesting to see how the machine learning algorithms find relationships with the different data and if splitting out the soil type into individual components improves the accuracy of the predicted tree type.

## Predicting Coverage Types

Logistic regression was chosen to test how well it predicted the different tree coverage types. It was chosen mainly to as a learning experience to become more familiar with the technique.

Logistic regression can only predict a true or false result, for example if the tree coverage was Aspen or Not Aspen. To use logistic regression to predict each of the seven tree coverages, an individual logistic regression needed to be run for each tree coverage type. Since we are trying to compare the aggregated Soil Types with the individualized components of the soil types, logistic regression was run for each of the tree types using the aggregated soil type and run again using the individualized data.

Each logistic regression was run initially with all data and then with insignificant features removed. The code for the initial logistic run for aggregated features (soil types 01 through 40) for Aspen tree coverage is shown below.

```
Aspen_Agg_LogMod =
    glm(Aspen ~
        Elev +      # Elevation in meters of data cell
        Aspect +    # Direction in degrees slope faces
        Slope +     # Slope / steepness of hill in degrees (0 to 90)
        H2OHD +     # Horizontal distance in meters to nearest water
        H2OVD +     # Vertical distance in meters to nearest water
        RoadHD +    # Horizontal distance in meters to nearest road
        FirePtHD + # Horizontal distance in meters to nearest fire point
        Shade9AM + Shade12PM + Shade3PM + # Amount of shade at 9am, 12pm and 3pm
        # Wilderness areas:
          RWwild + NEwild + CMwild + CPwild +
        # Aggregated Soil type:
          ST01 + ST02 + ST03 + ST04 + ST05 + ST06 + ST07 + ST08 + ST09 + ST10 +
          ST11 + ST12 + ST13 + ST14 + ST15 + ST16 + ST17 + ST18 + ST19 + ST20 +
          ST21 + ST22 + ST23 + ST24 + ST25 + ST26 + ST27 + ST28 + ST29 + ST30 +
          ST31 + ST32 + ST33 + ST34 + ST35 + ST36 + ST37 + ST38 + ST39 + ST40 ,
        data=forestTrain, family=binomial)
```

The code for the initial logistic run for individualized data features (soil types split into climate, geology, soil family and rock types) for Krummholz tree coverage is shown below.

```r
Krumm_Ind_LogMod =
  glm(Krummholz ~
    Elev + # Elevation in meters of cell
    Aspect + # Direction in degrees slope faces
    Slope + # Slope / steepness of hill in degrees (0 to 90)
    H2OHD + # Horizontal distance in meters to nearest water
    H2OVD + # Vertical distance in meters to nearest water
    RoadHD + # Horizontal distance in meters to nearest road
    FirePtHD + # Horizontal distance in meters to nearest fire point
    Shade9AM + Shade12PM + Shade3PM + # Amount of shade at 9am, 12pm and 3pm
    # Wilderness areas:
    RWwild + NEwild + CMwild + CPwild +
    # ClimateName +
    Montane_low + Montane + SubAlpine + Alpine + Dry + Non_Dry +
    # GeoName +
    Alluvium + Glacial + Sed_mix + Ign_Meta +
    # Soil Family:
    Aquolis_cmplx + Argiborolis_Pachic + Borohemists_cmplx + Bross +
    Bullwark + Bullwark_Cmplx + Catamount + Catamount_cmplx +
    Cathedral + Como + Cryaquepts_cmplx + Cryaquepts_Typic + Cryaquolls +
    Cryaquolls_cmplx + Cryaquolls_Typic + Cryaquolls_Typic_cmplx +
    Cryoborolis_cmplx + Cryorthents + Cryorthents_cmplx + Cryumbrepts +
    Cryumbrepts_cmplx + Gateview + Gothic + Granile + Haploborolis +
    Legault + Legault_cmplx + Leighcan + Leighcan_cmplx + Leighcan_warm +
    Moran + Ratake + Ratake_cmplx + Rogert + Supervisor_Limber_cmplx +
    Troutville + Unspecified + Vanet + Wetmore +
    # Rock Type:
    Bouldery_ext + Rock_Land + Rock_Land_cmplx + Rock_Outcrop +
    Rock_Outcrop_cmplx + Rubbly + Stony + Stony_extreme + Stony_very +
    Till_Substratum ,
    data=forestTrain, family=binomial)
```

In the initial models when all variables are used as shown above, the coefficients for many variables are extremely large in the millions and even trillions including the intercept. This is an indication that there are non-significant variables that need to be removed.

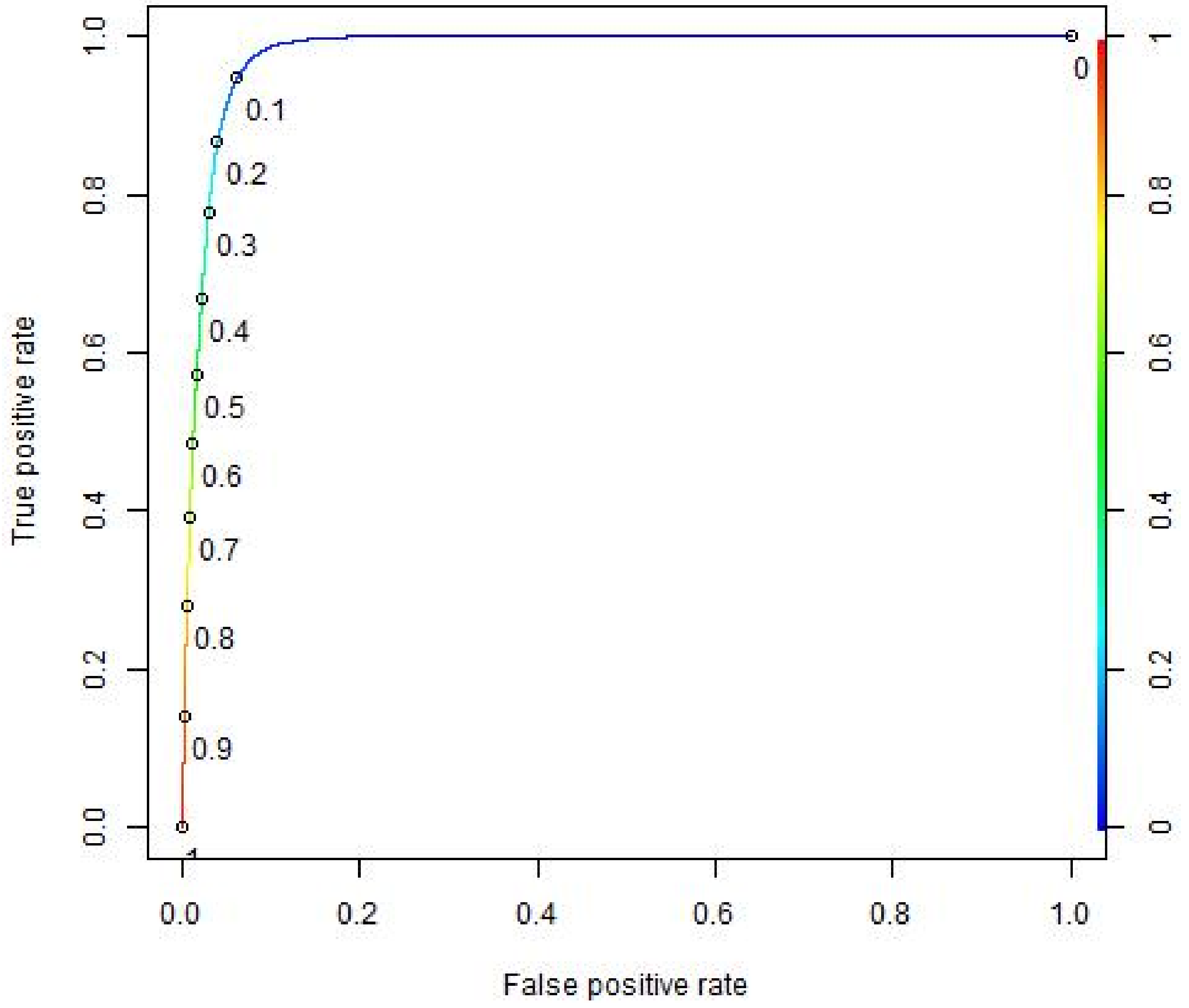
With the non-significant variables removed, the logistic model for all the tree coverage model look much better. The coefficients are more reasonable. Some of the variables that were significant in the original model were no longer significant.

The logistic models for the seven tree types with non-significant features removed are listed in Appendix A.

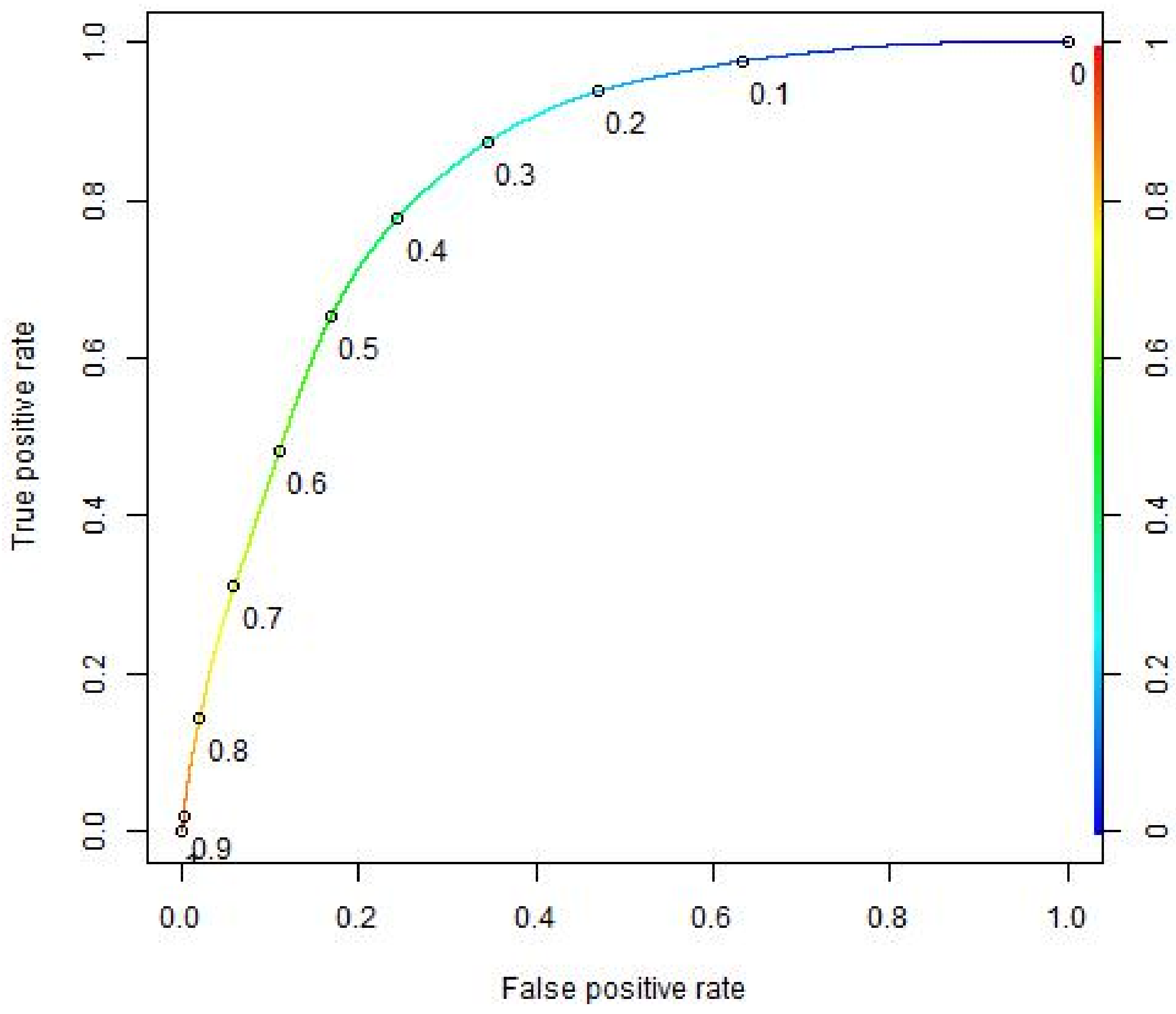
## Response Operating Characteristics Graphs

The Response Operating Characteristic was plotted for each selected model to aid in determining the threshold value that gave good sensitivity (percentage of correct True predictions) and specificity (percentage of correct False predictions). The ROC curve for two models is shown below.

**ROC Curve for Ponderosa Individualized Model**



**ROC Curve for Spruce / Fir Aggregated Model**



Some logistic models show a better ROC than others. The Ponderosa ROC had an AUC of 98% and is reflected in a highly desirable curve approaching the (1,1) point on the graph. The Spruce Fir ROC had an AUC of 84% and the curve is closer to the diagonal indicating the model is not as accurate as Ponderosa.

The Response Operating Characteristic Curves for all models are listed in Appendix B.

## Calc Probabilities using preferred Models

Once the best model for each tree type was determined, probabilities were added to the forest cover data frame as shown below.

```
# Calculate probabilities for each tree type based on preferred logistic model
load("Aspen_Ind_Sig_LogMod.Rdata")
forestcover$AspenProb=predict(Aspen_Ind_Sig_LogMod, type="response",newdata=forestcover)
...
load("SprFir_Agg_Sig_LogMod.Rdata")
forestcover$SprFirProb=predict(SprFir_Agg_Sig_LogMod, type="response",newdata=forestcover)
```

A function was created to find the optimum threshold for best sensitivity and specificity. It optionally printed a confusion matrix illustrating the sensitivity and specificity calculations. A sample output is shown below.

```
Model Performance for threshold= 0.099
Predicted          | Actual FALSE=Predict:Other | TRUE=Predict:Ponderosa
0=Actual:Other     |       357111 (TN)          |     24570 (FP)
1=Actual:Ponderosa |          434 (FN)          |     24594 (TP)
Sensitivity= 0.982 (True positive rate of Ponderosa = TP/(TP+FN) = 24594/(24594+434)
Specificity= 0.935 (True negative rate of Other = TN/(TN+FP) = 357111 /( 357111 + 24570
Sens^2+Spec^2=1.841
Baseline (Other) Accuracy=0.938462
Logistic Accuracy=0.938521
```

This function is listed in Appendix C. This function was used for each model with only significant features remaining. The results are shown below with the preferred model indicated with an 'X'.

| Model Description | Base | Acc | Sens | Spec | AUC | Count | Thresh | Preferred |
|---|---|---|---|---|---|---|---|---|
| Ponderosa Agg | 93% | 92% | 97% | 91% | 97% | 10726 | 0.082 | |
| Ponderosa Ind | 93% | 92% | 97% | 92% | 98% | 10726 | 0.068 | X |
| Douglas Fir Agg | 97% | 87% | 97% | 86% | 95% | 5210 | 0.033 | X |
| Douglas Fir Ind | 97% | 87% | 94% | 87% | 95% | 5210 | 0.032 | |
| Krummholz Agg | 96% | 90% | 95% | 89% | 97% | 6153 | 0.029 | X |
| Krummholz Ind | 96% | 86% | 96% | 86% | 96% | 6153 | 0.030 | |
| Cotton/Willow Agg | 99% | 95% | 94% | 95% | 98% | 824 | 0.008 | X |
| Cotton/Willow Ind | 99% | 95% | 93% | 95% | 98% | 824 | 0.008 | |
| Aspen Agg | 98% | 57% | 85% | 56% | 79% | 2848 | 0.012 | |
| Aspen Ind | 98% | 68% | 93% | 68% | 87% | 2848 | 0.011 | X |
| Lodgepole Agg | 51% | 75% | 79% | 72% | 82% | 84990 | 0.482 | X |
| Lodgepole Ind | 51% | 69% | 91% | 49% | 80% | 84990 | 0.345 | |
| Spruce/Fir Agg | 63% | 73% | 87% | 66% | 83% | 63552 | 0.307 | X |
| Spruce/Fir Ind | 63% | 73% | 87% | 65% | 84% | 63552 | 0.298 | |
| Weighted Average | | 76% | 84% | 72% | | 174303 | | |

The preferred model was chosen based on the best combination of sensitivity and specificity. We see that the differences between using the Soil Type aggregated data and splitting the soil type in to it's constituent parts often differs by only a percent. The Soil Type data models were chosen for 5 out of the 7 models. So the individuated data is not a significant improvement.

An estimated overall performance was calculated using a weighted average each of the individual models. If the accuracy estimate of 76% is correct, it would represent a 6% improvement over the neural network.

## Combined Logistic Models to Predict Tree Coverage

Once the preferred logistic models were chosen, the next step was to combine them together to predict each of the tree types.

To determine the tree type with the combined models, the models were applied to the training set in a particular order. Once the order to apply the models was chosen, two methods were explored to assign tree coverages. The first method allowed the tree to be assigned to a row if the probability was greater than the threshold for the model and the tree coverage had not yet been assigned by a previous model that had already been run. The second method assigned the tree coverage overriding any previous tree assignments as long as the probability exceeded the threshold for the model.

Two ideas were tried to determine the best order to apply the models. The first idea was to apply the models in the order of best to worst sensitivity if the tree coverage had not been assigned by a previous model. The second idea was to apply the models in the order of worst to best specificity overriding any previous tree assignments.

Choosing the worst specificity models first may seem counter-intuitive but it turned out to be better. Applying the models with the best sensitivity first tended to have too many false positives so that some tree types had zero correct identifications.

Applying the models with the worst specificity first and allowing the subsequent models with increasingly better specificity to override previous models that were applied generated fewer false positives. While this method was not quite as accurate as the best sensitivity first method, all tree types were represented.

A function to assign tree types is shown in Appendix D showing four different methods of applying the models.

A function to calculate a 7x7 confusion matrix and calculate statistics after the tree types were determined is shown in Appendix E.

The thresholds that were ideal when the models were run individually

A function to calculate thresholds for the combined models optimizing sensitivity and specificity is shown in Appendix F.

### Stats for applying highest Sensitivity first method to training set

The optimized thresholds using the highest sensitivity first method and confusion matrix with statistics is shown below.

Confusion Matrix (rows are actual, columns are predicted)

| Tree | Aspen_Pre | Cot&Wil | DougFir | Krumm | Lodge | Ponder | Spr&Fir |
|---|---|---|---|---|---|---|---|
| Aspen_Act | 0 | 6 | 154 | 0 | 5870 | 471 | 144 |
| Cot&Wil | 0 | 1 | 25 | 0 | 9 | 1873 | 0 |
| DougFir | 0 | 78 | 2230 | 0 | 1554 | 8283 | 0 |
| Krumm | 0 | 0 | 0 | 8748 | 974 | 56 | 4554 |
| Lodge | 0 | 29 | 3583 | 326 | 142076 | 4946 | 47351 |
| Ponder | 0 | 23 | 682 | 0 | 2312 | 21888 | 0 |
| SprFir | 0 | 51 | 415 | 7146 | 33741 | 103 | 106816 |

| Tree | TP | FP | FN | TN | Accuracy | Sensitivity | Specificity |
|---|---|---|---|---|---|---|---|
| Aspen | 0 | 0 | 6645 | 399873 | 0.9836539 | 0.000000000 | 1.0000000 |
| CotWill | 1 | 187 | 1907 | 404423 | 0.9948489 | 0.000524109 | 0.9995378 |
| DougFir | 2230 | 4859 | 9915 | 389514 | 0.9636572 | 0.183614656 | 0.9876792 |
| Krumm | 8748 | 7472 | 5584 | 384714 | 0.9678833 | 0.610382361 | 0.9809478 |
| Lodge | 142076 | 44460 | 56235 | 163747 | 0.7522988 | 0.716430253 | 0.7864625 |
| Ponder | 21888 | 15732 | 3017 | 365881 | 0.9538790 | 0.878859667 | 0.9587750 |
| SpruceFir | 106816 | 52049 | 41456 | 206197 | 0.7699856 | 0.720405741 | 0.7984519 |

Weighted Avg Sens= 0.692777882958086

Weighted Avg Spec= 0.818366298037202

Accuracy = 0.692777882958086

**Stats for applying lowest Specificity first method to training set**

The optimized thresholds using the lowest specificity first method and confusion matrix with statistics is shown below.

Confusion Matrix (rows are actual, columns are predicted)

| Tree | Aspen_Pre | Cot&Wil | DougFir | Krumm | Lodge | Ponder | Spr&Fir |
|---|---|---|---|---|---|---|---|
| Aspen_Act | 260 | 0 | 119 | 0 | 5126 | 551 | 585 |
| Cot&Wil | 0 | 119 | 8 | 0 | 0 | 1772 | 3 |
| DougFir | 105 | 0 | 1778 | 0 | 1095 | 8877 | 292 |
| Krumm | 57 | 0 | 0 | 8272 | 80 | 56 | 5892 |
| Lodge | 2187 | 0 | 3231 | 270 | 140331 | 5959 | 46250 |
| Ponder | 581 | 60 | 503 | 0 | 705 | 22392 | 624 |
| SprFir | 1553 | 0 | 406 | 6292 | 34866 | 118 | 105053 |

| Tree | TP | FP | FN | TN | Accuracy | Sensitivity | Specificity |
|---|---|---|---|---|---|---|---|
| Aspen | 260 | 4483 | 6381 | 395304 | 0.9732696 | 0.03915073 | 0.9887865 |
| CotWill | 119 | 60 | 1783 | 404466 | 0.9954654 | 0.06256572 | 0.9998517 |
| DougFir | 1778 | 4267 | 10369 | 390014 | 0.9639887 | 0.14637359 | 0.9891778 |
| Krumm | 8272 | 6562 | 6085 | 385509 | 0.9688826 | 0.57616494 | 0.9832632 |
| Lodge | 140331 | 41872 | 57897 | 166328 | 0.7545223 | 0.70792724 | 0.7988857 |
| Ponder | 22392 | 17333 | 2473 | 364230 | 0.9512681 | 0.90054293 | 0.9545737 |
| SpruceFir | 105053 | 53646 | 43235 | 204494 | 0.7616281 | 0.70843898 | 0.7921825 |

Weighted Avg Sens= 0.684039448352508

Weighted Avg Spec= 0.82164066329442

Accuracy = 0.684039448352508

As discussed previously, even though the accuracy of the highest sensitivity first method with an accuracy of 69.3% is better than the lowest specificity first method accuracy of 68.4%, the highest sensitivity first method does not predict any Aspen trees. The lowest specificity first method has predictions for all tree types and will be used as the model to apply to the testing set.

**Stats for applying lowest Specificity first method to testing set**

The final model chosen from the training set was then applied to the testing set. Applying the lowest specificity first method to the testing set produced the confusion matrix and statistics shown below.

Confusion Matrix (rows are actual, columns are predicted)

| Tree | Aspen_Pre | Cot&Wil | DougFir | Krumm | Lodge | Ponder | Spr&Fir |
|---|---|---|---|---|---|---|---|
| Aspen_Act | 121 | 0 | 50 | 0 | 2199 | 237 | 240 |
| Cot&Wil | 0 | 38 | 9 | 0 | 0 | 769 | 0 |

| Tree | Aspen_Pre | Cot&Wil | DougFir | Krumm | Lodge | Ponder | Spr&Fir |
|---|---|---|---|---|---|---|---|
| DougFir | 44 | 0 | 781 | 0 | 491 | 3772 | 116 |
| Krumm | 24 | 0 | 0 | 3495 | 38 | 22 | 2574 |
| Lodge | 954 | 0 | 1485 | 99 | 60175 | 2533 | 19706 |
| Ponder | 236 | 18 | 224 | 0 | 331 | 9587 | 257 |
| SprFir | 633 | 0 | 162 | 2805 | 14617 | 52 | 45283 |

| Tree | TP | FP | FN | TN | Accuracy | Sensitivity | Specificity |
|---|---|---|---|---|---|---|---|
| Aspen | 121 | 1891 | 2726 | 169439 | 0.9734925 | 0.04250088 | 0.9889628 |
| CotWill | 38 | 18 | 778 | 173343 | 0.9954299 | 0.04656863 | 0.9998962 |
| DougFir | 781 | 1930 | 4423 | 167043 | 0.9635256 | 0.15007686 | 0.9885781 |
| Krumm | 3495 | 2904 | 2658 | 165120 | 0.9680670 | 0.56801560 | 0.9827168 |
| Lodge | 60175 | 17676 | 24777 | 71549 | 0.7562652 | 0.70834118 | 0.8018941 |
| Ponder | 9587 | 7385 | 1066 | 156139 | 0.9514804 | 0.89993429 | 0.9548384 |
| SpruceFir | 45283 | 22893 | 18269 | 87732 | 0.7636772 | 0.71253462 | 0.7930576 |

Weighted Avg Sens= 0.685472998169854

Weighted Avg Spec= 0.823379445775709

Accuracy = 0.685472998169854

The accuracy of 68.5% on the testing set is not as accurate as the 70% of the neural network but is 10% better than the 58% accuracy of the Discriminant Analysis methods that were in use prior to the implementation of the neural network.

## Conclusion

The accuracy of the model with the best sensitivity and specificity is 68.5% which is about 1.5% less than the 70% accuracy of the neural network that this project is based. It does not improve on the accuracy of the neural network but comes very close. Looking at the model performance during the individual model build phase it looked like the overall performance could perform better than the neural network. But the performance of the combined models could not be predicted. They had to be combined to determine the overall performance. While the neural network gives the better result, building and comparing the logistic models helps show which features are important to predict the model coverage and would be recommended even if the logistic regression models are not to be used.

The project helped deepen my understanding of logistic regression in all the different aspects of data exploration, data cleaning, data visualization to predicting an outcome. Although the data was already clean, splitting the Soil Type data into it's constituent parts gave great experience in data manipulation.

One of the main goals was to see if breaking the Soil Type into it's constituent parts would result in better prediction. But the difference between the two types of data is often less than 1%. So it seems that they are essentially equivalent. There may be a mathematical proof of this but I have not figured that out yet.

As I was completing the project I did some research on the original paper but could not find a copy online. I did find a kaggle competition in 2015 using the same data (see: https://www.slideshare.net/danielgribel/forest-cover-type-prediction-56288946) which resulted in one team achieving 100% accuracy with the data. I was not able to find information on the methodology that achieved 100% accuracy.

## Appendicies

### Appendix A - Logistic Regression Model Coefficients

The output of the optimized logistic regression models is shown for each tree coverage.

#### Appendix A1 - Aspen Model using Significant Individualized Features

```
load("../ForestCoverage/Aspen_Ind_Sig_LogMod.Rdata")
summary(Aspen_Ind_Sig_LogMod)
```

```
##
## Call:
## glm(formula = Aspen ~ Elev + Aspect + H2OHD + H2OVD + RoadHD +
##     FirePtHD + Shade9AM + Shade12PM + Shade3PM + RWwild + NEwild +
##     CMwild + CPwild + Montane_low + Montane + Non_Dry + Sed_mix +
##     Catamount_cmplx + Legault + Rock_Land + Rock_Land_cmplx +
##     Rock_Outcrop_cmplx, family = binomial, data = forestTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1363  -0.1733  -0.0948  -0.0441   3.7975
##
## Coefficients: (2 not defined because of singularities)
##                      Estimate Std. Error    z value Pr(>|z|)
## (Intercept)        -1.036e+01  8.037e+00 -1.289e+00 0.197297
## Elev               -3.499e-03  9.012e-05 -3.883e+01  < 2e-16 ***
## Aspect              2.338e-03  1.905e-04  1.227e+01  < 2e-16 ***
## H2OHD              -1.100e-03  1.104e-04 -9.970e+00  < 2e-16 ***
## H2OVD               5.548e-03  3.172e-04  1.749e+01  < 2e-16 ***
## RoadHD             -4.794e-04  1.522e-05 -3.150e+01  < 2e-16 ***
## FirePtHD           -4.613e-05  1.369e-05 -3.369e+00 0.000753 ***
## Shade9AM            1.449e-02  2.598e-03  5.580e+00 2.41e-08 ***
## Shade12PM           4.966e-03  2.588e-03  1.919e+00 0.054979 .
## Shade3PM           -4.990e-03  2.078e-03 -2.402e+00 0.016312 *
## RWwild              1.328e+01  8.026e+00  1.654e+00 0.098040 .
## NEwild             -4.504e+15  4.640e+05 -9.706e+09  < 2e-16 ***
## CMwild              1.354e+01  8.026e+00  1.687e+00 0.091556 .
## CPwild                     NA         NA         NA       NA
## Montane_low        -1.655e+00  1.685e+00 -9.820e-01 0.326211
## Montane            -1.266e+00  1.684e+00 -7.520e-01 0.452223
## Non_Dry             1.324e+00  1.685e+00  7.860e-01 0.431902
## Sed_mix                    NA         NA         NA       NA
## Catamount_cmplx     2.480e-01  6.793e-02  3.651e+00 0.000261 ***
## Legault            -2.320e+01  2.969e+03 -8.000e-03 0.993765
## Rock_Land           1.229e+00  3.782e-02  3.249e+01  < 2e-16 ***
## Rock_Land_cmplx     2.196e-01  5.993e-02  3.665e+00 0.000247 ***
## Rock_Outcrop_cmplx -5.508e-01  4.660e-02 -1.182e+01  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 67859  on 406708  degrees of freedom
## Residual deviance: 53044  on 406688  degrees of freedom
## AIC: 53086
##
## Number of Fisher Scoring iterations: 25
```

### Appendix A2 - Cottonwood/Willow Model using Significant Aggregated Features

```
load("../ForestCoverage/CotWil_Agg_Sig_LogMod.Rdata")
summary(CotWil_Agg_Sig_LogMod)
```

```
##
## Call:
## glm(formula = Cottonwood_Willow ~ Elev + Aspect + Slope + H2OHD +
##     H2OVD + RoadHD + FirePtHD + Shade9AM + ST04 + ST10 + ST11 +
##     ST16 + ST17, family = binomial, data = forestTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3396  -0.0294  -0.0104  -0.0033   4.3176
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  9.970e+00  5.131e-01  19.430  < 2e-16 ***
## Elev        -1.008e-02  1.986e-04 -50.784  < 2e-16 ***
## Aspect       3.589e-03  3.655e-04   9.818  < 2e-16 ***
## Slope       -6.116e-02  3.787e-03 -16.149  < 2e-16 ***
## H2OHD       -8.036e-03  5.077e-04 -15.828  < 2e-16 ***
## H2OVD        1.861e-02  1.097e-03  16.968  < 2e-16 ***
## RoadHD       7.496e-04  4.438e-05  16.890  < 2e-16 ***
## FirePtHD    -7.445e-05  3.875e-05  -1.921 0.054685 .
## Shade9AM     4.688e-02  1.532e-03  30.591  < 2e-16 ***
## ST04         2.376e-01  1.044e-01   2.277 0.022815 *
## ST10        -3.648e-01  9.916e-02  -3.679 0.000234 ***
## ST11        -4.568e-01  2.504e-01  -1.824 0.068130 .
## ST16         9.395e-01  1.769e-01   5.311 1.09e-07 ***
## ST17         1.744e+00  1.027e-01  16.982  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 24429  on 406708  degrees of freedom
## Residual deviance: 10848  on 406695  degrees of freedom
## AIC: 10876
##
## Number of Fisher Scoring iterations: 11
```

### Appendix A3 - Douglas Fir Model using Significant Aggregated Features

```r
load("../ForestCoverage/DougFir_Agg_Sig_LogMod.Rdata")
summary(DougFir_Agg_Sig_LogMod)
```

```
##
## Call:
## glm(formula = DouglasFir ~ Elev + Slope + H2OHD + FirePtHD +
##     Shade9AM + Shade12PM + Shade3PM + CMwild, family = binomial,
##     data = forestTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -6.2060  -0.1411  -0.0630  -0.0307   4.8780
##
## Coefficients:
##               Estimate Std. Error  z value Pr(>|z|)
## (Intercept) -8.111e+00  9.180e-01   -8.836  < 2e-16 ***
## Elev        -7.340e-03  6.314e-05 -116.258  < 2e-16 ***
## Slope        8.150e-02  5.253e-03   15.516  < 2e-16 ***
## H2OHD       -9.713e-04  7.896e-05  -12.300  < 2e-16 ***
## FirePtHD    -5.030e-05  1.417e-05   -3.549 0.000387 ***
## Shade9AM     1.917e-01  5.636e-03   34.013  < 2e-16 ***
## Shade12PM   -1.926e-01  4.663e-03  -41.311  < 2e-16 ***
## Shade3PM     1.742e-01  4.673e-03   37.278  < 2e-16 ***
## CMwild       1.712e+00  2.908e-02   58.880  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 109294  on 406708  degrees of freedom
## Residual deviance:  64196  on 406700  degrees of freedom
## AIC: 64214
##
## Number of Fisher Scoring iterations: 8
```

### Appendix A4 - Krummholz Model using Significant Aggregated Features

```r
load("../ForestCoverage/Krumm_Agg_Sig_LogMod.Rdata")
summary(Krumm_Agg_Sig_LogMod)
```

```
##
## Call:
## glm(formula = Krummholz ~ Elev + Aspect + Slope + H2OHD + H2OVD +
##     FirePtHD + Shade9AM + Shade12PM + Shade3PM + ST04 + ST13 +
##     ST19 + ST22 + ST23 + ST24 + ST29 + ST31 + ST32 + ST33 + ST35 +
##     ST36 + ST38 + ST39, family = binomial, data = forestTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3633  -0.1065  -0.0380  -0.0104   4.7337
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -4.764e+01  7.067e-01 -67.414  < 2e-16 ***
## Elev         1.438e-02  1.419e-04 101.303  < 2e-16 ***
## Aspect       1.586e-03  1.440e-04  11.018  < 2e-16 ***
## Slope       -2.996e-02  3.331e-03  -8.997  < 2e-16 ***
## H2OHD       -1.375e-03  6.503e-05 -21.149  < 2e-16 ***
## H2OVD       -1.232e-03  2.475e-04  -4.977 6.44e-07 ***
## FirePtHD     1.379e-04  1.045e-05  13.191  < 2e-16 ***
## Shade9AM    -5.414e-03  3.125e-03  -1.733   0.0832 .  
## Shade12PM    9.646e-03  2.594e-03   3.719   0.0002 ***
## Shade3PM    -1.757e-02  2.515e-03  -6.985 2.85e-12 ***
## ST04         4.996e+00  1.606e-01  31.106  < 2e-16 ***
## ST13        -2.084e+00  4.502e-01  -4.630 3.65e-06 ***
## ST19        -3.954e+00  1.001e+00  -3.950 7.81e-05 ***
## ST22        -2.140e+00  1.081e-01 -19.795  < 2e-16 ***
## ST23        -7.108e-01  5.968e-02 -11.911  < 2e-16 ***
## ST24        -5.079e-01  9.476e-02  -5.360 8.33e-08 ***
## ST29        -5.591e-01  5.698e-02  -9.814  < 2e-16 ***
## ST31        -8.842e-01  9.045e-02  -9.776  < 2e-16 ***
## ST32        -1.034e+00  5.424e-02 -19.054  < 2e-16 ***
## ST33        -6.648e-01  6.253e-02 -10.631  < 2e-16 ***
## ST35         8.164e-01  6.899e-02  11.834  < 2e-16 ***
## ST36         1.850e+00  2.314e-01   7.995 1.29e-15 ***
## ST38         1.178e+00  3.807e-02  30.948  < 2e-16 ***
## ST39         1.543e+00  3.818e-02  40.396  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 124217  on 406708  degrees of freedom
## Residual deviance:  55739  on 406685  degrees of freedom
## AIC: 55787
## 
## Number of Fisher Scoring iterations: 10
```

### Appendix A5 - Lodgepole Model using Significant Aggregated Features

```r
load("../ForestCoverage/Lodge_Agg_Sig_LogMod.Rdata")
summary(Lodge_Agg_Sig_LogMod)
```

```
## 
## Call:
## glm(formula = LodgepolePine ~ Elev + Aspect + Slope + H2OHD + 
##     H2OVD + RoadHD + FirePtHD + Shade12PM + Shade3PM + RWwild + 
##     NEwild + CMwild + ST02 + ST03 + ST04 + ST06 + ST08 + ST09 + 
##     ST10 + ST11 + ST12 + ST13 + ST16 + ST18 + ST19 + ST20 + ST21 + 
##     ST22 + ST23 + ST24 + ST25 + ST26 + ST27 + ST28 + ST29 + ST30 + 
##     ST31 + ST32 + ST33 + ST34 + ST35 + ST36, family = binomial, 
##     data = forestTrain)
## 
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.9057  -0.8850  -0.1397   0.8644   3.6729
##
## Coefficients:
##               Estimate Std. Error  z value Pr(>|z|)
## (Intercept)  3.097e+00  9.637e-02   32.136  < 2e-16 ***
## Elev        -5.309e-03  2.877e-05 -184.562  < 2e-16 ***
## Aspect      -2.867e-04  4.624e-05   -6.201 5.61e-10 ***
## Slope        1.410e-02  7.705e-04   18.294  < 2e-16 ***
## H2OHD        1.518e-03  2.455e-05   61.826  < 2e-16 ***
## H2OVD        4.717e-04  8.983e-05    5.252 1.51e-07 ***
## RoadHD       7.861e-05  3.098e-06   25.370  < 2e-16 ***
## FirePtHD     1.652e-05  3.418e-06    4.833 1.34e-06 ***
## Shade12PM    2.461e-02  3.249e-04   75.743  < 2e-16 ***
## Shade3PM    -2.869e-03  1.636e-04  -17.540  < 2e-16 ***
## RWwild       4.924e+00  3.940e-02  124.987  < 2e-16 ***
## NEwild       5.376e+00  4.192e-02  128.255  < 2e-16 ***
## CMwild       4.723e+00  3.528e-02  133.879  < 2e-16 ***
## ST02        -1.950e+00  5.470e-02  -35.656  < 2e-16 ***
## ST03        -4.212e-01  5.708e-02   -7.378 1.60e-13 ***
## ST04        -8.647e-01  4.002e-02  -21.604  < 2e-16 ***
## ST06         1.942e+00  5.880e-02   33.036  < 2e-16 ***
## ST08         2.181e+00  1.977e-01   11.030  < 2e-16 ***
## ST09         1.873e+00  1.058e-01   17.708  < 2e-16 ***
## ST10         1.441e+00  3.377e-02   42.669  < 2e-16 ***
## ST11         2.092e+00  3.862e-02   54.174  < 2e-16 ***
## ST12         3.302e+00  3.897e-02   84.745  < 2e-16 ***
## ST13         2.704e+00  3.626e-02   74.569  < 2e-16 ***
## ST16         1.779e+00  6.004e-02   29.630  < 2e-16 ***
## ST18         1.820e+00  9.554e-02   19.052  < 2e-16 ***
## ST19         1.810e+00  4.931e-02   36.712  < 2e-16 ***
## ST20         1.766e+00  3.931e-02   44.936  < 2e-16 ***
## ST21        -1.005e+00  2.732e-01   -3.680 0.000233 ***
## ST22         1.439e+00  3.264e-02   44.077  < 2e-16 ***
## ST23         1.782e+00  3.011e-02   59.176  < 2e-16 ***
## ST24         2.410e+00  3.313e-02   72.741  < 2e-16 ***
## ST25         3.633e+00  1.346e-01   26.982  < 2e-16 ***
## ST26         3.123e+00  7.219e-02   43.266  < 2e-16 ***
## ST27         2.069e+00  8.084e-02   25.596  < 2e-16 ***
## ST28         3.965e+00  1.814e-01   21.856  < 2e-16 ***
## ST29         2.522e+00  3.084e-02   81.779  < 2e-16 ***
## ST30         2.468e+00  3.401e-02   72.555  < 2e-16 ***
## ST31         2.505e+00  3.230e-02   77.549  < 2e-16 ***
## ST32         2.822e+00  3.022e-02   93.365  < 2e-16 ***
## ST33         2.573e+00  3.070e-02   83.820  < 2e-16 ***
## ST34         3.931e+00  1.030e-01   38.159  < 2e-16 ***
## ST35        -1.389e+00  3.800e-01   -3.655 0.000257 ***
## ST36         2.270e+00  2.249e-01   10.095  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 563568  on 406708  degrees of freedom
## Residual deviance: 419386  on 406666  degrees of freedom
## AIC: 419472
##
## Number of Fisher Scoring iterations: 7
```

### Appendix A6 - Ponderosa Model using Significant Individualized Features

```r
load("../ForestCoverage/Ponder_Ind_Sig_LogMod.Rdata")
summary(Ponder_Ind_Sig_LogMod)
```

```
##
## Call:
## glm(formula = PonderosaPine ~ Elev + Aspect + Slope + H2OHD +
##     H2OVD + RoadHD + FirePtHD + Shade9AM + Shade12PM + Shade3PM +
##     CMwild + Cathedral + Haploborolis + Ratake + Wetmore, family = binomial,
##     data = forestTrain)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.13999  -0.10743  -0.03966  -0.01621   2.96120
##
## Coefficients:
##                Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   2.335e+01  5.328e-01   43.821  < 2e-16 ***
## Elev         -1.010e-02  7.687e-05 -131.382  < 2e-16 ***
## Aspect        1.538e-03  1.246e-04   12.343  < 2e-16 ***
## Slope        -2.642e-03  3.242e-03   -0.815   0.4152
## H2OHD         3.058e-03  7.934e-05   38.542  < 2e-16 ***
## H2OVD         2.392e-03  2.157e-04   11.087  < 2e-16 ***
## RoadHD       -8.696e-05  1.601e-05   -5.432 5.57e-08 ***
## FirePtHD     -4.025e-04  1.697e-05  -23.714  < 2e-16 ***
## Shade9AM     -2.699e-02  2.977e-03   -9.065  < 2e-16 ***
## Shade12PM     4.257e-02  2.485e-03   17.133  < 2e-16 ***
## Shade3PM     -3.009e-02  2.459e-03  -12.236  < 2e-16 ***
## CMwild        1.474e+00  2.886e-02   51.058  < 2e-16 ***
## Cathedral    -5.267e-01  6.383e-02   -8.252  < 2e-16 ***
## Haploborolis -1.252e-01  4.882e-02   -2.564   0.0103 *
## Ratake        2.066e+00  3.509e-02   58.883  < 2e-16 ***
## Wetmore       1.386e+00  4.007e-02   34.594  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 188044  on 406708  degrees of freedom
## Residual deviance:  71679  on 406693  degrees of freedom
## AIC: 71711
##
## Number of Fisher Scoring iterations: 9
```

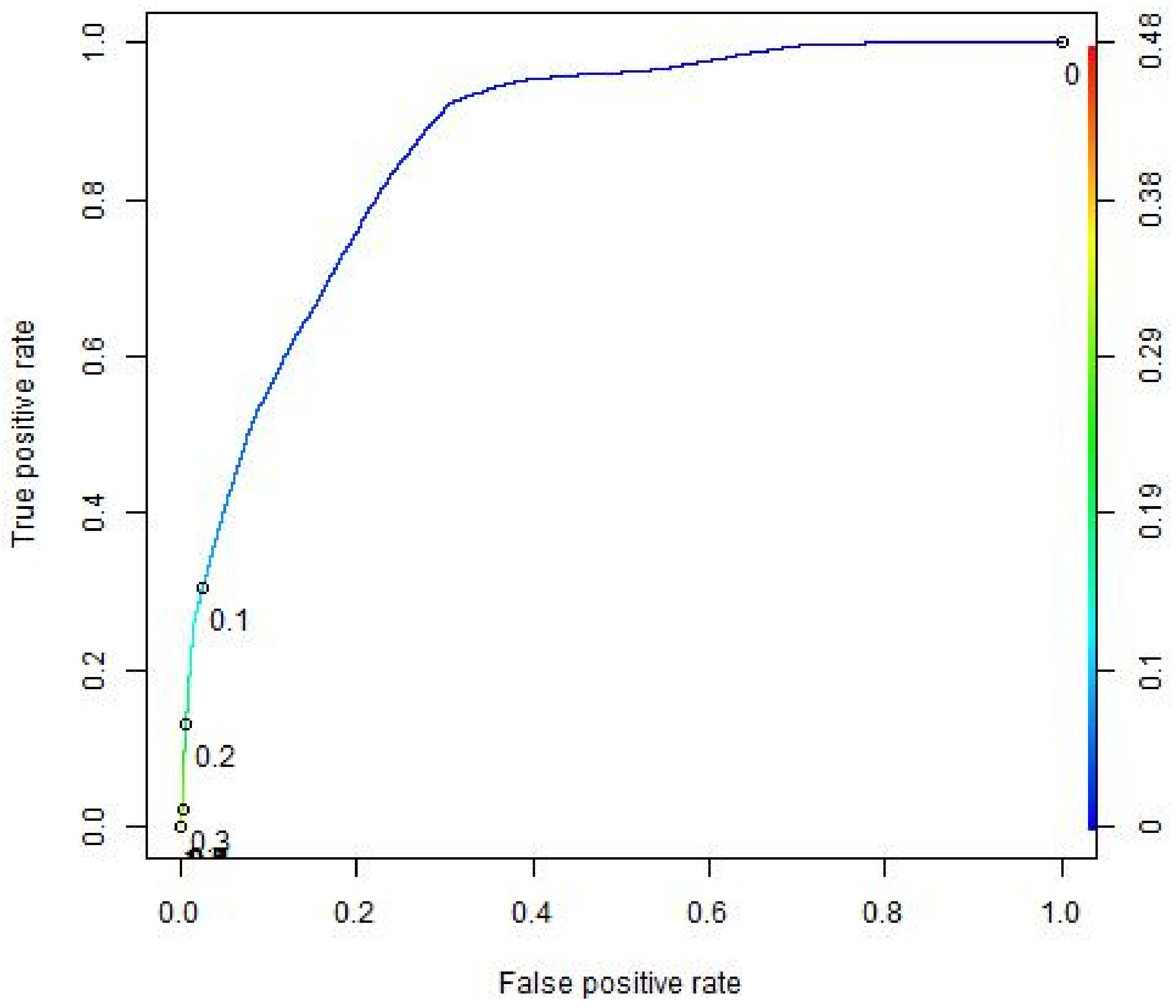
**Appendix A7 - Spruce/Fir Model using Significant Aggregated Features**

```r
load("../ForestCoverage/SprFir_Agg_Sig_LogMod.Rdata")
summary(SprFir_Agg_Sig_LogMod)
```

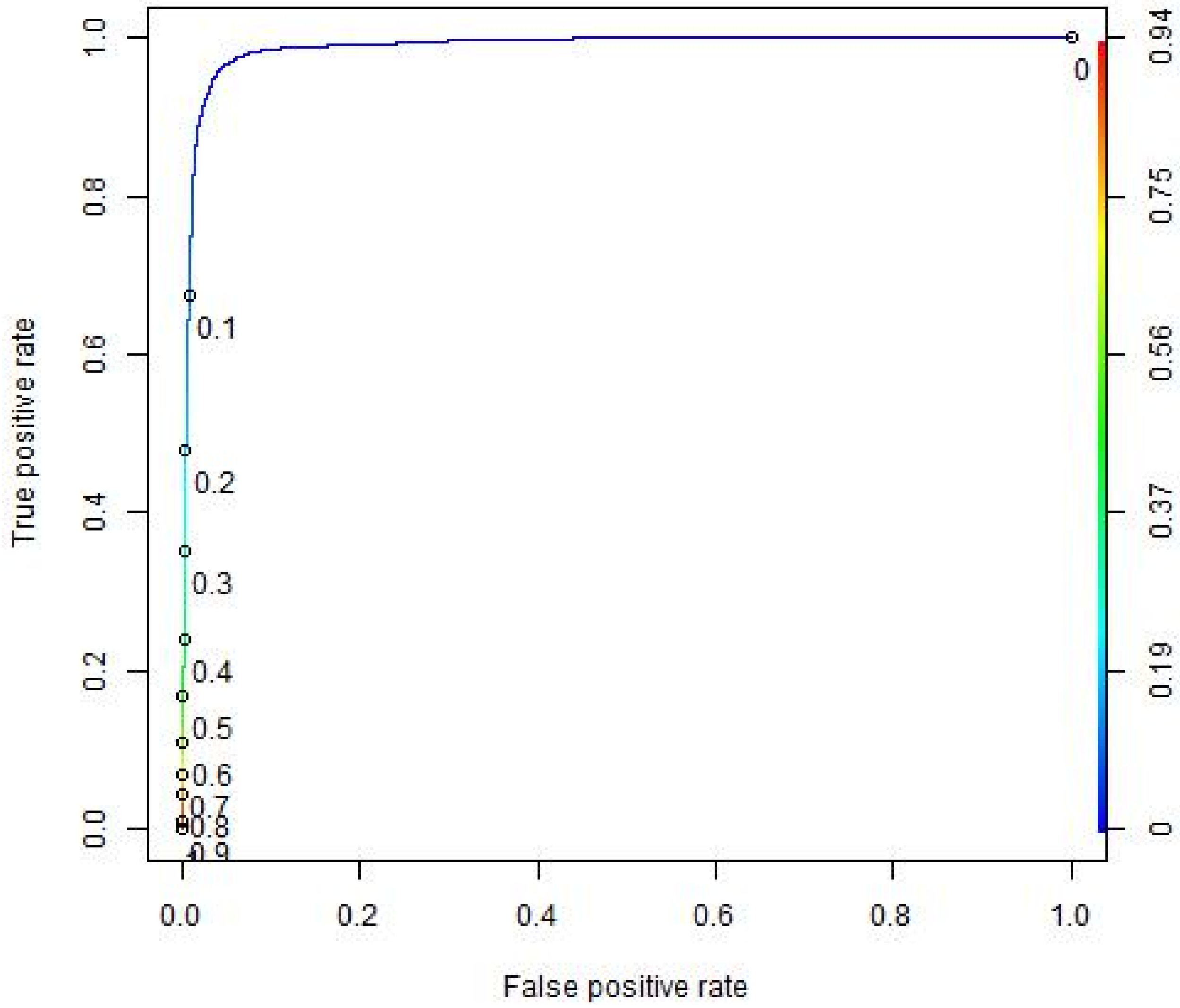
```
##
## Call:
## glm(formula = Spruce_Fir ~ Elev + Aspect + Slope + H2OHD + H2OVD +
##     RoadHD + FirePtHD + Shade9AM + Shade12PM + Shade3PM + ST04 +
##     ST08 + ST09 + ST10 + ST11 + ST12 + ST16 + ST17 + ST18 + ST19 +
##     ST20 + ST21 + ST22 + ST23 + ST24 + ST25 + ST26 + ST27 + ST28 +
##     ST29 + ST30 + ST31 + ST32 + ST33 + ST36 + ST38 + ST39, family = binomial,
##     data = forestTrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3322  -0.7740  -0.2732   0.8484   3.0533
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.714e+01  2.905e-01 -59.017  < 2e-16 ***
## Elev         6.465e-03  3.030e-05 213.401  < 2e-16 ***
## Aspect      -5.464e-04  4.825e-05 -11.324  < 2e-16 ***
## Slope       -8.028e-03  1.488e-03  -5.396 6.81e-08 ***
## H2OHD       -1.419e-03  2.468e-05 -57.489  < 2e-16 ***
## H2OVD       -1.417e-03  9.354e-05 -15.150  < 2e-16 ***
## RoadHD      -3.439e-05  2.907e-06 -11.831  < 2e-16 ***
## FirePtHD    -9.349e-06  3.435e-06  -2.722 0.006494 **
## Shade9AM     6.014e-03  1.675e-03   3.591 0.000329 ***
## Shade12PM   -2.901e-02  1.378e-03 -21.055  < 2e-16 ***
## Shade3PM     1.196e-02  1.369e-03   8.734  < 2e-16 ***
## ST04         1.120e-01  9.529e-02   1.176 0.239768
## ST08         1.747e+00  1.970e-01   8.867  < 2e-16 ***
## ST09         2.680e+00  1.046e-01  25.614  < 2e-16 ***
## ST10         7.718e-01  4.726e-02  16.333  < 2e-16 ***
## ST11         8.323e-01  5.205e-02  15.991  < 2e-16 ***
## ST12         8.084e-01  3.473e-02  23.277  < 2e-16 ***
## ST16         1.828e+00  6.470e-02  28.254  < 2e-16 ***
## ST17         1.216e+00  8.861e-02  13.723  < 2e-16 ***
## ST18         8.469e-01  1.795e-01   4.717 2.39e-06 ***
## ST19         1.932e+00  4.648e-02  41.561  < 2e-16 ***
## ST20         2.166e+00  3.658e-02  59.219  < 2e-16 ***
## ST21         4.485e+00  2.038e-01  22.009  < 2e-16 ***
## ST22         2.353e+00  2.655e-02  88.629  < 2e-16 ***
## ST23         2.019e+00  2.444e-02  82.645  < 2e-16 ***
## ST24         1.587e+00  2.787e-02  56.956  < 2e-16 ***
## ST25        -1.922e-01  1.329e-01  -1.446 0.148311
## ST26         6.971e-01  8.128e-02   8.577  < 2e-16 ***
## ST27         1.865e+00  7.793e-02  23.936  < 2e-16 ***
## ST28         1.324e-01  2.116e-01   0.626 0.531570
## ST29         1.319e+00  2.313e-02  57.041  < 2e-16 ***
## ST30         1.153e+00  2.794e-02  41.270  < 2e-16 ***
## ST31         1.504e+00  2.665e-02  56.418  < 2e-16 ***
## ST32         1.050e+00  2.382e-02  44.085  < 2e-16 ***
## ST33         1.423e+00  2.436e-02  58.421  < 2e-16 ***
## ST36        -1.018e+00  3.239e-01  -3.144 0.001669 **
## ST38         2.847e-01  2.878e-02   9.892  < 2e-16 ***
## ST39         2.910e-01  2.972e-02   9.791  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 533620  on 406708  degrees of freedom
## Residual deviance: 384051  on 406671  degrees of freedom
## AIC: 384127
##
## Number of Fisher Scoring iterations: 6
```

## Appendix B - Response Operating Characteristics for Optimized Logistic Models

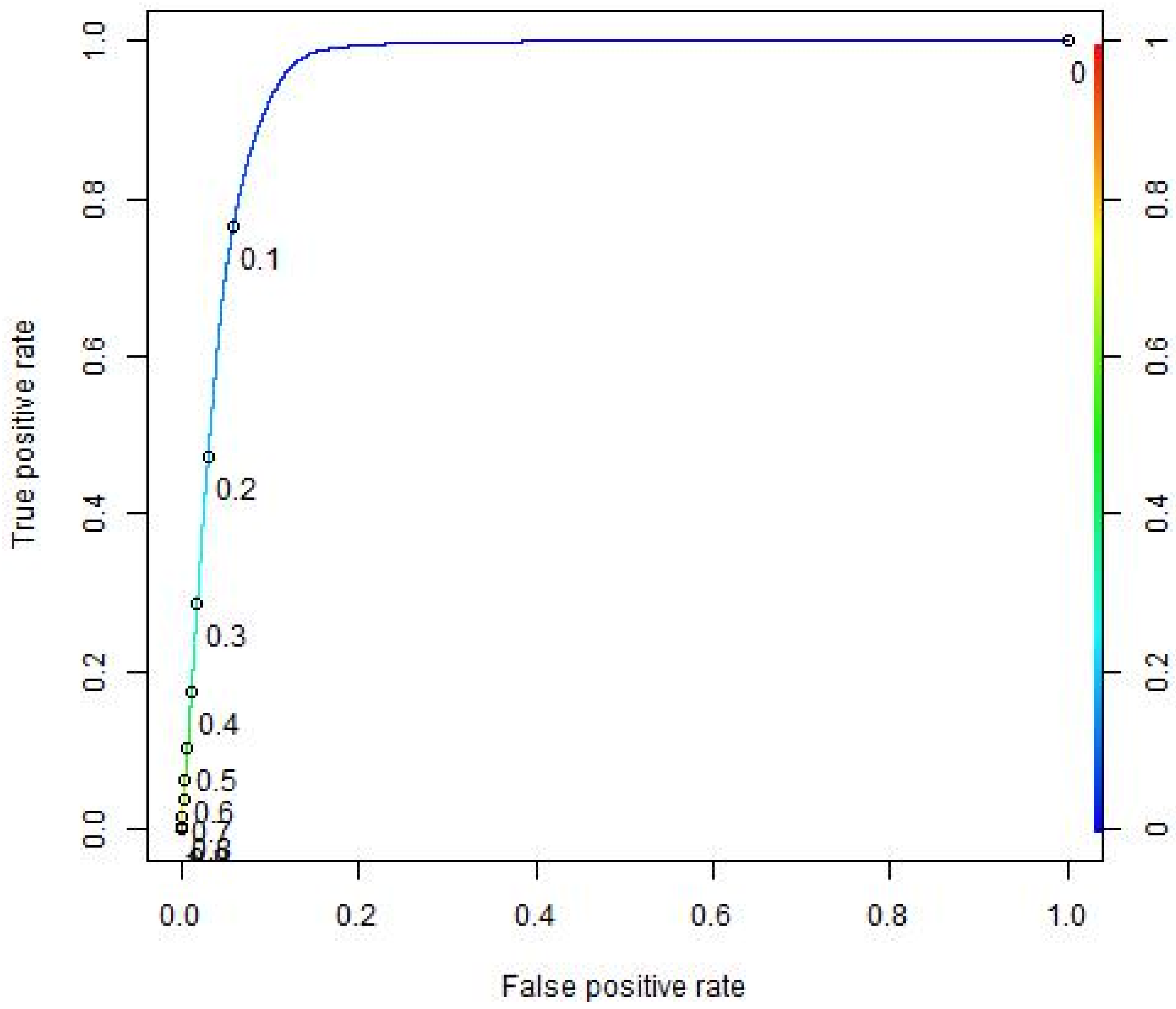
### Appendix B1 - Response Operating Charactistics for Aspen Model



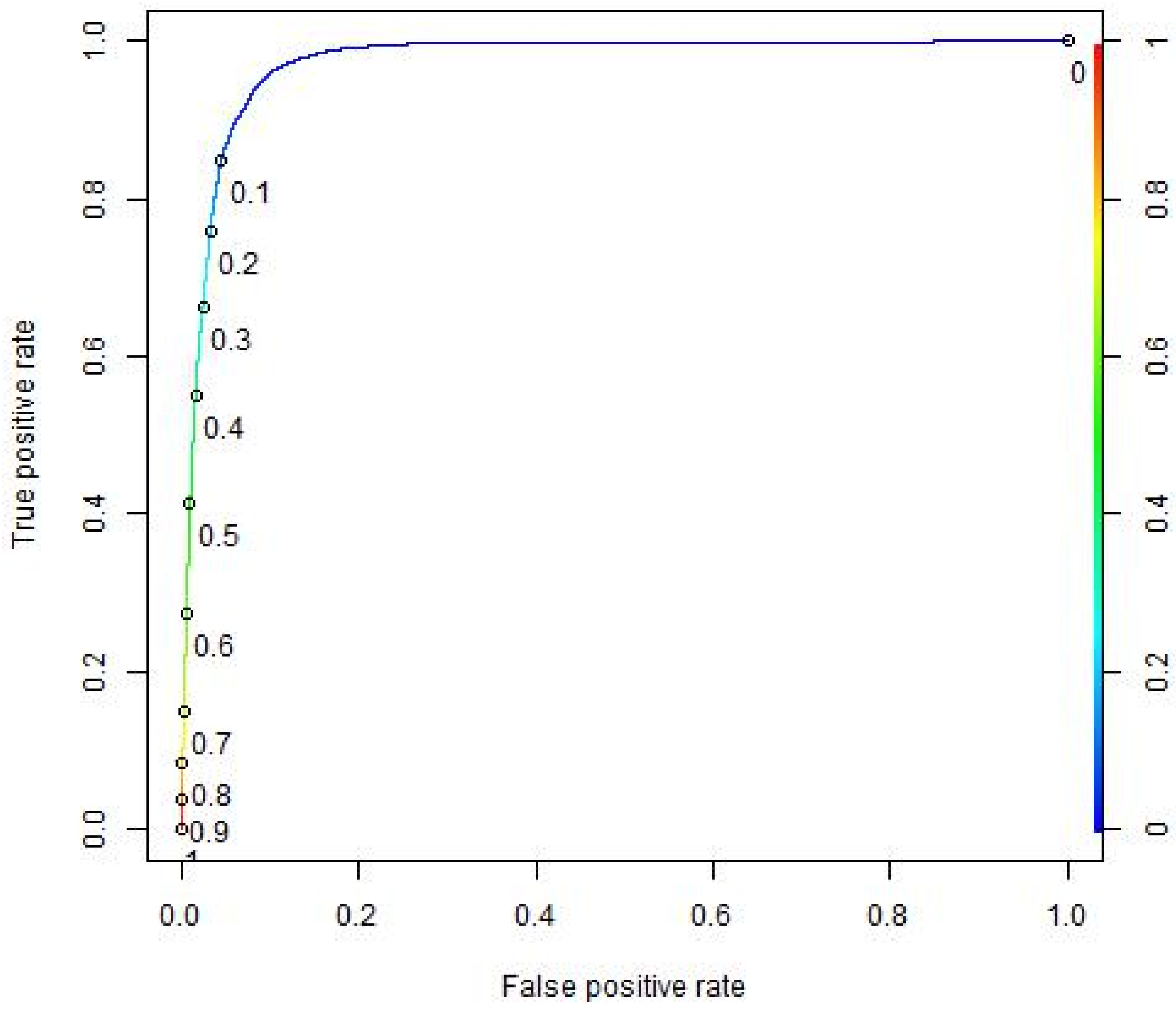
**Appendix B2 - Response Operating Charactistics for Cottonwood / Willow Model**



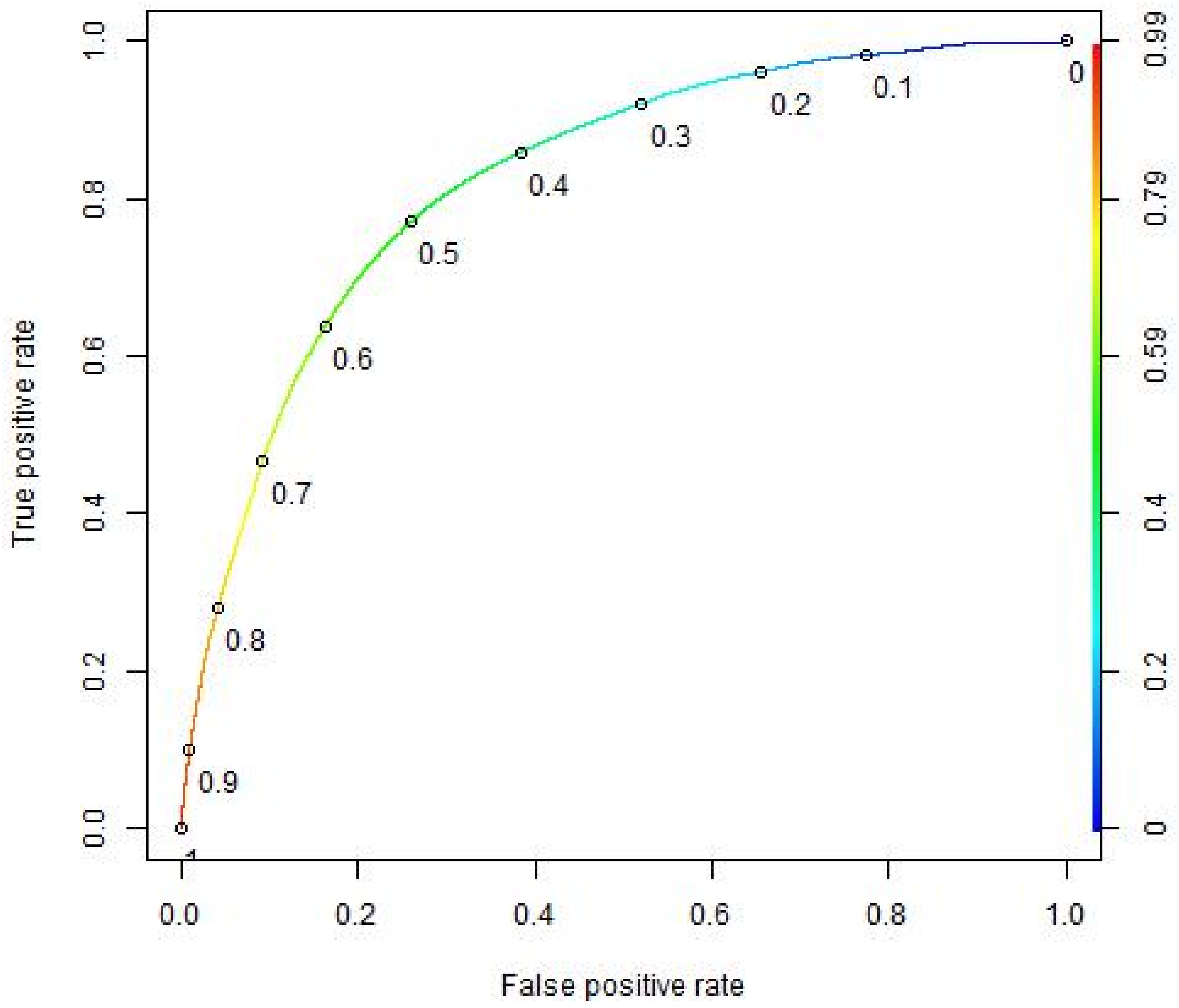
**Appendix B3 - Response Operating Charactistics for Douglas Fir Model**



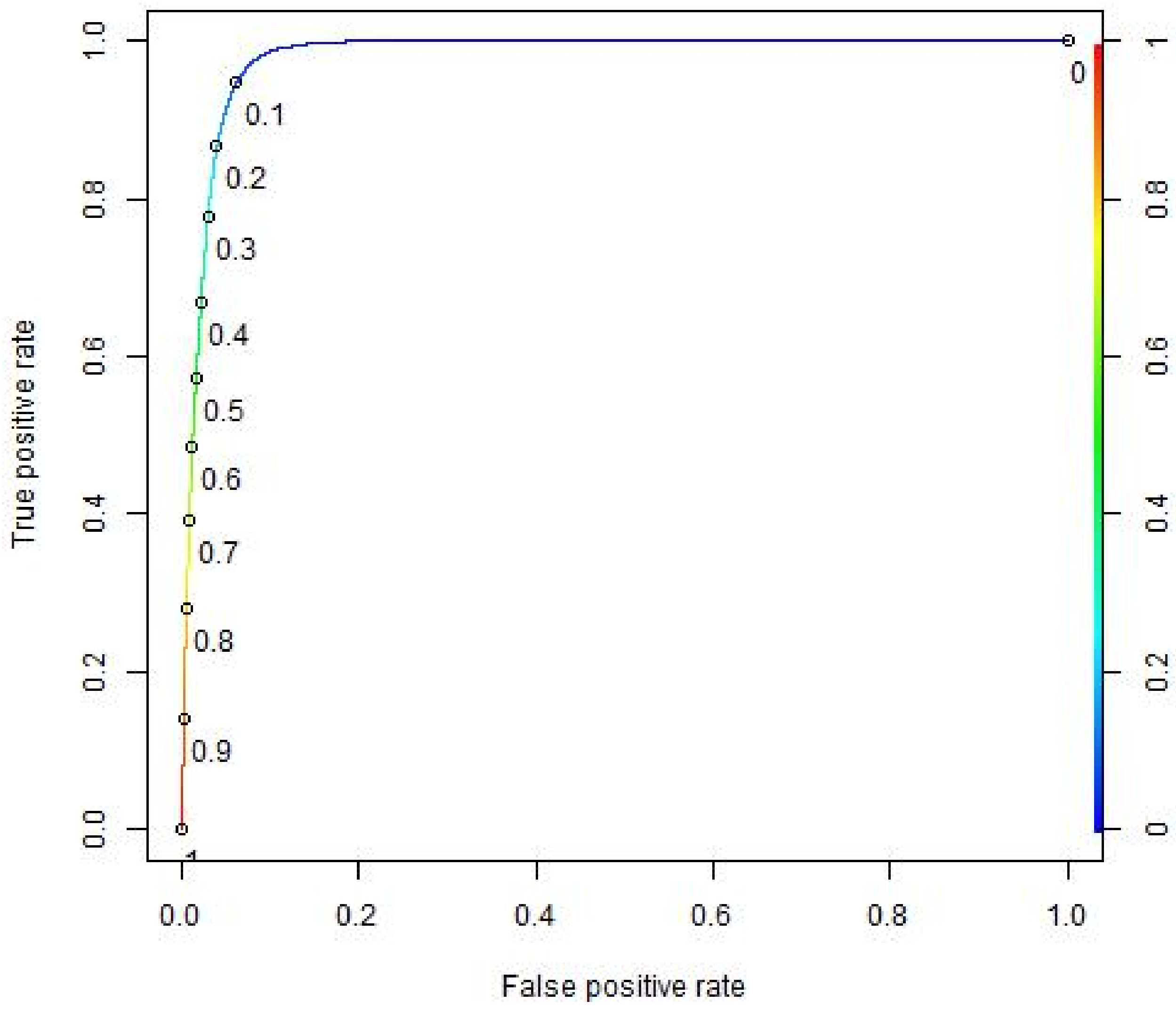
**Appendix B4 - Response Operating Charactistics for Krummholz Model**



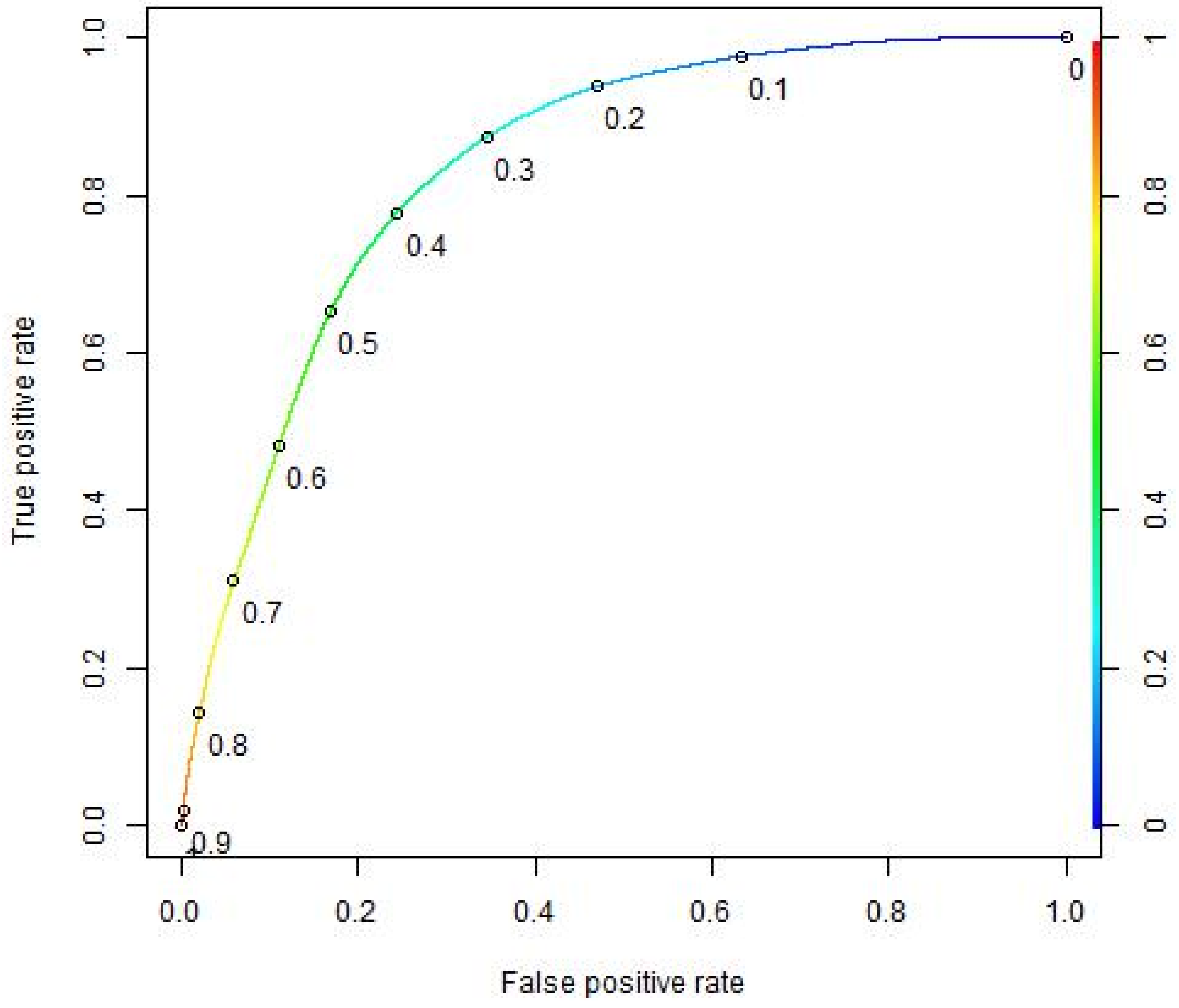
**Appendix B5 - Response Operating Charactistics for Lodgepole Model**



**Appendix B6 - Response Operating Charactistics for Ponderosa Model**



**Appendix B7 - Response Operating Charactistics for Spruce / Fir Model**



## Appendix C - Logistic Model Accuracy Function

A function to help determine threshold for best accuracy and testing is shown below.

```
#load("logisticAccuracy.Rdata")

calcLogisticModelAccuracy <- function(actualValues, predictedValues,
                         thresholdStart, thresholdEnd, thresholdParts,
                         positiveLabel, negativeLabel, printLevel,
                         findThreshold=0,
                         saveFile="", desc="LogisticStats", AIC=NA, AUC=NA, Append=TRUE) {
```

```r
# Description
#   -Calculate accuracy of logistic regression model
#   -depending on print level option:
#      print accuracy of logistic model and baseline model
#      print confusion matrix
#      print sensitivity and specificty
#
# Input Values
#   -actualValues = actual values of outcome variables, a vector of 0's and 1's
#   -predictedValues = logistic model predicted probalilities between 0 and 1
#   -thresholdStart = threshold initial value for applying to predicted values
#      to determine predicted outcome
#   -thresholdEnd = end value for incrementing the threshold
#   -thresholdParts = number of partitions to apply threshold values between
#      thresholdStart and thresholdEnd
#   -positiveLabel = text to label true outcomes. This will be displayed
#      on the confusion matrix when the print level is greater than 1.
#   -negativeLable = text to label false outcomes. This will be displayed
#      on the confusion matrix when the print level is greater than 1.
#   -printLevel = level of detail printed by calcLogisticModelAccuracy
#      0 - no printed output unless and error is encountered
#      1 - print threshold, logistic model accuracy and baseline accuracy
#      2 - Print level 1 and confusion matrix and sensitivity and specificity values
#      3 - Print level 2 and details of sensitivity and specificity calculations
#      4 - Print level 3 and debug information
# -findThreshold
#      1 - search for threshold producing best sensitivity and specificity combination
#      2 - search for threshold producing best accuracy
#
# Return Values
#   -function status:
#      - "OK":function completed without errors
#      - "ERROR": function did not complete, and error information
#         See other variables for possible additional error information
#   -logistic model accuracy based on last threshold value tested
#   -baseline model accuracy based on last threshold value tested
#   -confusion matrix values in following order: TN, FN, FP, TP
#   -sensitivity
#   -specificity

# x <- data.frame('1', 'ab', 'username', '<some.sentence>', '2017-05-04T00:51:35Z', '24')
# write.table(x, file = "Tweets.csv", sep = ",", append = TRUE, quote = FALSE,
#  col.names = FALSE, row.names = FALSE)
if (saveFile != "") {
  if (!file.exists(saveFile) | Append == FALSE){
     if (file.exists(saveFile)) { file.remove(saveFile) }
     x <- data.frame('Description', 'TrueLabel', 'FalseLabel', 'BaselineLabel',
      'BaselineAcc', 'Accuracy', 'Sensitivity', 'Specificity',
      'AIC', 'AUC%', 'TP', 'FN', 'FP','TN', 'Count', 'Threshold' )
     write.table(x, file=saveFile, sep=",", quote=FALSE,
       col.names = FALSE, row.names = FALSE)
  }
}
```

```r
# set default values in case of errors
accuracy=baseline=retVal="ERROR"
more=1
bestLabel="Sensitivity_Specificity"
SensSpec = -1

bestAccuracy=bestThreshold=NA # set default values for bestThreshold calcs

if (findThreshold) {
  thresholdStart=0.0
  thresholdEnd=1.0
  thresholdParts=10
  more=3 # allows calculation to find threshold to nearest 0.001
  bestAccuracy=bestThreshold=-1.0
  if (findThreshold == 2) { bestLabel = "Accuracy" }
  print (paste("Searching for threshold producing best",bestLabel))
}

# Calculate increment value to iterate through the threshold values
if ( thresholdParts ==0) { thresholdParts = 1 }
if ( thresholdParts < 0) { thresholdParts = - thresholdParts }
thresholdInc = (thresholdEnd - thresholdStart) / thresholdParts
if (thresholdStart==thresholdEnd | thresholdParts < 2) {
  thresholdEnd=thresholdStart
  thresholdInc=1
}
threshold=thresholdStart

if (findThreshold) {
  print(paste("start=",thresholdStart,"end=",thresholdEnd,"inc=",thresholdInc))
}

funcStat="OK"

workPerformance = table(actualValues, predictedValues > threshold)

for (row in rownames(workPerformance)) {
  if(row != "0" & row != "1") {
    funcStat=paste("ERROR:Bad row name:",row,",must be '0' or '1'")}
}
for (col in colnames(workPerformance)) {
  if(col != "TRUE" & col != "FALSE") {
    funcStat=paste("ERROR:Bad column name:",col,", must be 'TRUE' or 'FALSE'")}
}

if (funcStat=="OK") {
  while (more) {
  repeat {

    if (thresholdParts>1 & printLevel > 1) { print("----------")}

    workPerformance = table(actualValues, predictedValues > threshold)
```

```
# create a modelPerformance table and set all the values to zero.
# This ensures a 2x2 matrix in case the threshold causes all values predicted
# to be TRUE or FALSE values and produces a 2x1 vector.
# The table of actual and predicted values with be copied into the
# modelPerformance table later.
Actual = c(0, 1)
Predicted = c(FALSE, TRUE )
modelPerformance = table(Actual,Predicted)
modelPerformance["0","TRUE"]=0
modelPerformance["0","FALSE"]=0
modelPerformance["1","FALSE"]=0
modelPerformance["1","TRUE"]=0

# Descriptions          | Predict Good Care (0) | Predict Poor Care (1)
# ----------------------|-----------------------|----------------------
# Actual Good Care (0) |     TN (true neg)     |   FP (false pos)
# Actual Poor Care (1) |     FN (false neg)    |   TP (true pos)

# Remember: 0 means negative which means Good care,
#           1 means positive which means Poor care
#   (Opposite of intuition)

# Sensitivity = TP / (TP + FN) = percent of true positives identified

# Specificity = TN / (TN + FP) = percent of true negatives identified

# transfer the workPerformance table to the final performance table
for (row in rownames(workPerformance)) {
  for (col in colnames(workPerformance)) {
    modelPerformance[row,col]=workPerformance[row,col]
    if (printLevel > 3) { print(paste("workPerformance[",row,",",col,"]=",
                                  workPerformance[row,col]))}
  }
}

if (printLevel > 3) {print(modelPerformance) }

#                  Actual,Prediction
TP = modelPerformance["1","TRUE"]  # Predicted True (1),  and actually TRUE (1) = True Positive
FN = modelPerformance["1","FALSE"] # Predicted False (0), but actually TRUE (1) = False 0/Negativ

TN = modelPerformance["0","FALSE"] # Predicted False (0), and actually False (0) = True Negative
FP = modelPerformance["0","TRUE"] # Predicted True (1), but actually False (0) = False 1/Positive

# Prevent and report divide by zero error
if (TP+FN == 0) {
  sensitivity="ERROR:TP+FN=0"
  funcStat=sensitivity
} else { sensitivity = TP / (TP + FN ) }

# Prevent and report divide by zero error
if (TN+FP == 0) {
  specificity="ERROR:TN+FP=0"
```

```
    funcStat=specificity
  } else { specificity = TN / (TN + FP) }

  if (funcStat == "OK") { # calc SensSpec
    if (sensitivity > 0.0 & sensitivity < 1.0) {
        SensSpec=sensitivity^2 + specificity^2
    } else { SensSpec = -2.0 }
        printSensSpec=as.integer(SensSpec*1000)/1000
  }

  retVal = c(modelPerformance, sensitivity,specificity) # TN, FN, FP, TP, sens, spec

  if (printLevel > 1) {
    modelPerformance["1","TRUE"]  = paste("   ",modelPerformance["1","TRUE"], "(TP)")
    modelPerformance["1","FALSE"] = paste("   ",modelPerformance["1","FALSE"],"(FN)")
    modelPerformance["0","FALSE"] = paste("   ",modelPerformance["0","FALSE"],"(TN)")
    modelPerformance["0","TRUE"]  = paste("   ",modelPerformance["0","TRUE"], "(FP)")
  }

  c1=paste("FALSE=Predict:",negativeLabel,sep="")
  c2=paste("TRUE=Predict:",positiveLabel,sep="")
  r1=paste("0=Actual:",negativeLabel,sep="")
  r2=paste("1=Actual:",positiveLabel,sep="")
  colnames(modelPerformance) <- c(c1,c2)
  rownames(modelPerformance) <- c(r1,r2)

  if (printLevel > 1) {
    print(paste("Model Performance for threshold=", threshold))
    print("predicted performance=")
    print(modelPerformance)

    sensPrint=paste("Sensitivity=",sensitivity,"(True positive rate of",positiveLabel)

    specPrint=paste("Specificity=",specificity,"(True negative rate of",negativeLabel)

    if (printLevel > 2 ) {
      sensPrint=paste(sensPrint,"= TP/(TP+FN) =",TP,"/(",TP,"+",FN,"))")
      specPrint=paste(specPrint,"= TN/(TN+FP) =",TN,"/(",TN,"+",FP,"))")
    }

    print(sensPrint)
    print(specPrint)
  }

  # Calculate actual true and actual false totals to calculate baseline accuracy
  # and logistic model accuracy
  totSamples=TP+FN+TN+FP
  actTrue=TP+FN
  actFalse=TN+FP

  # double check there were actually some non-zero values
  if (totSamples>0) {
    if (actTrue > actFalse) {
      baseline = actTrue / totSamples
      baseModel= positiveLabel
    } else {
      baseline = actFalse / totSamples
      baseModel=negativeLabel
    }

    # the accuracy is the number of TRUE positives and True negatives
    # divided by the number of samples
    accuracy=(TP+TN)/totSamples
    if (findThreshold)  {
      if (findThreshold == 2) {
        if (accuracy > bestAccuracy) {
          bestAccuracy=accuracy
          bestThreshold=threshold
        }
      } else {
        if (SensSpec > bestAccuracy) {
          bestAccuracy=SensSpec
          bestThreshold=threshold
        }
      }
    }
  } else {
    baseModel="ERROR:0 samples"
    baseline="ERROR:0 samples"
    accuracy="ERROR:0 samples"
    funcStat=accuracy
  }

  if (printLevel > 0) {

    printAcc=(as.integer(accuracy*1000000))/1000000
    printbaseline=(as.integer(baseline*1000000))/1000000

    if (printLevel > 1) {
      print(paste("Sens^2+Spec^2=",printSensSpec,sep=""))
      print(paste("Baseline (",baseModel,") Accuracy=",printbaseline,sep=""))
      print(paste("Logistic Accuracy=",printAcc,sep=""))
    } else {
      print(paste("Thresh=",threshold,
            ", Accuracy=",as.integer(accuracy*1000)/10,
            "%, BaseAcc(",baseModel,")=",as.integer(baseline*1000)/10,
            "%, Sens=",as.integer(sensitivity*1000)/10,
            "%, Spec=",as.integer(specificity*1000)/10,
            "%, Sens^2+Spec^2=",printSensSpec,
            sep=""))
    }
  }

  # c(funcStat,accuracy,baseline,retVal)

  #print(paste("threshold=",threshold,",End=",thresholdEnd,",Inc=",thresholdInc))
```

```
      threshold=threshold+thresholdInc
      if(thresholdEnd < thresholdStart) {
        if (threshold < thresholdEnd) { break}
      } else { if (threshold > thresholdEnd) { break} }

    } # end repeat

      more=more-1

      if (findThreshold & more) {
        print(paste("Best",bestLabel,"threshold=",bestThreshold,"inc=",thresholdInc))
        thresholdStart = bestThreshold - thresholdInc
        if (thresholdStart < 0.0) { thresholdStart = 0.0 }
        thresholdEnd = bestThreshold + thresholdInc
        if (thresholdEnd > 1.0) { thresholdEnd = 1.0 }
        thresholdInc = (thresholdEnd - thresholdStart) / 20.0
        threshold=thresholdStart
        print("========================================")
        print(paste("start=",thresholdStart,"end=",thresholdEnd,"inc=",thresholdInc))
      }
    } # end while

    if (findThreshold) {
      print("========================================")
      print(paste("Best Threshold=",bestThreshold,sep=""))
      print(paste("Best ",bestLabel,"=",bestAccuracy,sep=""))
    }
  } else {
    # Had an error, just return the error information
    print(funcStat)
  }

  # x <- data.frame('Description', 'TrueLabel', 'FalseLabel', 'BaselineLabel',
  #       'BaseLineAcc', 'Accuracy', 'Sensitivity', 'Specificity',
  #       'AIC', 'AUC', 'TP', 'FN', 'TP','TN', 'Count' )
  if (saveFile != "" & funcStat == "OK") {
    threshold=threshold-thresholdInc
    x <-data.frame(desc, positiveLabel,negativeLabel, baseModel,
      baseline, accuracy, sensitivity, specificity,
      AIC, AUC, TP, FN, FP,TN, totSamples, threshold)
    write.table(x, file=saveFile, sep=",", append=TRUE, quote=FALSE,
      col.names = FALSE, row.names = FALSE)
  }

  c(funcStat,accuracy,baseline,retVal,bestAccuracy,bestThreshold)
}
```

## Appendix D - Calculate 7x7 Confusion Matrix for Combined Logistic Model Function

A function to create a 7x7 Confusion Matrix for the combined logistic regression model is shown below. It calculates the accuracy and weighted sensitivity and specificity for the combined logistic regression model.

```r
calcConfusionMatrix<-function (
  df,         # dataset with Actual Coverage Type and Estimated Coverage Type set
  ccmDebug=0  # debug: 0=no printing, 1=print details
)
{
  treeNames=c("Aspen", "Cotton&Willow", "DouglasFir", "Krummholz",
              "Lodgepole", "Ponderosa", "Spruce&Fir")
  confusionMat=zeroMat

  # Create a confusion matrix
  for (drow in 1:7) {
    actLabel<-treeNames[drow]
    for (dcol in 1:7) {
      predLabel<-treeNames[dcol]

      # populate each cell of the confusion matrix comparing the actual coverage type
      # with the coverage type estimated by the model
      confusionMat[drow,dcol]=sum(df$CovName==actLabel & df$EstTreeType==predLabel)
    }
  }

  # Abbreviate the row and column names so the table is not split up by column
  confRows<-c("Aspen_Act", "Cot&Wil", "DougFir", "Krumm",
              "Lodge", "Ponder", "SprFir")
  confCols<-c("Aspen_Pre", "Cot&Wil", "DougFir", "Krumm",
              "Lodge", "Ponder", "Spr&Fir")
  rownames(confusionMat)<-confRows
  colnames(confusionMat)<-confCols

  if (ccmDebug) {
    print("Confusion Matrix (rows are actual, columns are predicted) =")
    print(confusionMat)
  }

  # create a 7x7 zero matrix to hold statistics
  statsMat=zeroMat
  rownames(statsMat)<-treeLabels
  colnames(statsMat)<-c("TP","FP","FN","TN","Acc","Sens","Spec")

  # initialize variables
  weightedSens=0.0
  weightedSpec=0.0
  accuracy=0.0

  # Calculate statistics from confusion matrix
  for(treeIndex in 1:7) { # calculate stats for each tree coverage type
    TP = confusionMat[treeIndex,treeIndex] # True Positive for class is on the diagonal
    accuracy=accuracy+TP # caclulate accuracy by first accumulating all True Positives
    totClass=sum(confusionMat[treeIndex,]) # total number of class is the row sum (all actual values fo
    FN = sum(confusionMat[treeIndex,])-TP # False Neg = totClass - True Pos
    # which is sum of the cells in the Actual class row not predicting the class value
    FP = sum(confusionMat[,treeIndex])-TP # False Pos = col sum of predicted values - True Pos
    # which is the sum of the cells in the predicted class that are not the actual class value
    TN =0 # Initialize True Negative
    for (drow in 1:7) { # True negative is sum of all cells not in row or col of the class
      for (dcol in 1:7) {
        if (drow != treeIndex & dcol != treeIndex) TN=TN+confusionMat[drow,dcol]
      }
    }
    statsMat[treeIndex,1]=TP
    statsMat[treeIndex,2]=FP
    statsMat[treeIndex,3]=FN
    statsMat[treeIndex,4]=TN
    statsMat[treeIndex,5]=(TP + TN)/(TP+TN+FP+FN) # Set accuracy
    statsMat[treeIndex,6]=TP/(TP+FN) # Set Sensitivity for feature - positive predicted%
    statsMat[treeIndex,7]=TN/(TN+FP) # Set Specificity for feature - negative predicted%

    # accumulate weighted sensitivity and specificity for later overall model to calculation
    weightedSens = weightedSens + (totClass * statsMat[treeIndex,6])
    weightedSpec = weightedSpec + (totClass * statsMat[treeIndex,7])
  }

  # complete weighted calculations by dividing by number of rows in data set
  weightedSens = weightedSens / nrow(df)
  weightedSpec = weightedSpec / nrow(df)
  accuracy=accuracy/nrow(df)

  if (ccmDebug) {
    print("Stats")
    print(statsMat)
    print(paste("Weighted Avg Sens=",weightedSens))
    print(paste("Weighted Avg Spec=",weightedSpec))
    print(paste("Accuracy         =",accuracy))
  }
  c(weightedSens, weightedSpec, accuracy)
}
```

## Appendix E - Calculate Tree Types for Combined Logistic Regression Models

A function to calculate tree types for the combined logistic regression model for a set of thresholds is shown below.

```
# Calculate tree types based on passed in threshholds.
# Probabilities were previously calculated
calcTreeTypes <-
function(tds,              # tree data set
         mode,
         AspenThresh,
         CotWillThresh,
         DougFirThresh,
         KrummThresh,
         LodgeThresh,
         PonderThresh,
         SprFirThresh
         )
{
```

```r
tds$EstTreeType="X" # set Estimated tree type to default
#tds$EstTreeType=as.character(tds$EstTreeType)

if(1 == 2) {
  print(AspenThresh)
  print(CotWillThresh)
  print(DougFirThresh)
  print(KrummThresh)
  print(LodgeThresh)
  print(PonderThresh)
  print(SprFirThresh)
}

print(paste("calcTreeType Mode=",mode))
# determine tree types applying logistic regression models in order described by mode
if (mode == 1) { # sensitivity order, highest to lowest, update all
  print(paste("calcTreeType(Mode 1)"))
  tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > PonderThresh] = "Ponderosa"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > DougFirThresh] = "DouglasFir"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > KrummThresh] = "Krummholz"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > CotWilThresh] = "Cotton&Willow"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > AspenThresh] = "Aspen"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > SprFirThresh] = "Spruce&Fir"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > LodgeThresh] = "Lodgepole"
} else if (mode == 2) { # specifcity order, highest to lowest, update unassigned only
  print(paste("calcTreeType(Mode 2)"))
  tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > CotWilThresh]="Cotton&Willow"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > PonderThresh] = "Ponderosa"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > KrummThresh] = "Krummholz"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > DougFirThresh] = "DouglasFir"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > LodgeThresh] = "Lodgepole"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > AspenThresh] = "Aspen"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > SprFirThresh] = "Spruce&Fir"
} else if (mode ==3) { # specifcity order, lowest to highest, update unassigned only
  print(paste("calcTreeType(Mode 3)"))
  tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > SprFirThresh] = "Spruce&Fir"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > AspenThresh] = "Aspen"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > LodgeThresh] = "Lodgepole"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > DougFirThresh] = "DouglasFir"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > KrummThresh] = "Krummholz"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > PonderThresh] = "Ponderosa"
  tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > CotWilThresh]="Cotton&Willow"
} else { # specifcity order, lowest to highest, update all
  print(paste("calcTreeType(Mode 4)"))
  tds$EstTreeType[tds$SprFirProb > SprFirThresh] = "Spruce&Fir"
  tds$EstTreeType[tds$AspenProb > AspenThresh] = "Aspen"
  tds$EstTreeType[tds$LodgeProb > LodgeThresh] = "Lodgepole"
  tds$EstTreeType[tds$DougFirProb > DougFirThresh] = "DouglasFir"
  tds$EstTreeType[tds$KrummProb > KrummThresh] = "Krummholz"
  tds$EstTreeType[tds$PonderProb > PonderThresh] = "Ponderosa"
  tds$EstTreeType[tds$CotWilProb > CotWilThresh]="Cotton&Willow"
}
```

```
  ccm=calcConfusionMatrix(tds,1) # report stats for the combined 7 logistic regression models
  ccm
}
```

## Appendix F - Find Best Threshold for Combined Logistic Model Function

A function to help determine a set of thresholds for best sensitivity and specificity of the combined logistic regression models is shown below.

```
# find Threshholds optimized for the seven combined logistic regression models
findModelThresholds <-
function(tds,
         printLevel,
         findThreshold,
         mode,
         iterations,
         initAspenThresh,
         initCotWillThresh,
         initDougFirThresh,
         initKrummThresh,
         initLodgeThresh,
         initPonderThresh,
         initSprFirThresh
         ) {

  if (printLevel > 1) print(table(tds$EstTreeType))

  # Reset data
  tds$EstTreeType="X"

  threshs =c(initAspenThresh, initCotWillThresh, initDougFirThresh, initKrummThresh,
              initLodgeThresh, initLodgeThresh, initSprFirThresh)

  for (i in 1:iterations) { # number of times to optimize complete set of thresholds

    for (j in 1:7) { # variables to optimize

      start=0.1
      end = 0.9
      increment = 0.1
      curThresh=start
      bestAccuracy = 0.0
      bestThresh = threshs[j]

      for (k in 1:2) { # optimize increments by 0.1, 0.01, 0.001
        more=TRUE
        #bestThresh=threshs[j] # save current threshold for kth tree type

        if (printLevel > 0) {
          print(paste("Start=",start,", end=",end, ", inc=",increment))
          print("---------------------------")
        }
        while(more) {
          threshs[j]= curThresh

          # determine tree types applying logistic regression models in order described
          # in comments below
          if (mode == 1) { # sensitivity order, highest to lowest, update only if unassigned
            tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > threshs[6]] = "Ponderosa"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > threshs[3]] = "DouglasFir"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > threshs[4]] = "Krummholz"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > threshs[2]]="Cotton&Willow"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > threshs[1]] = "Aspen"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > threshs[7]] = "Spruce&Fir"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > threshs[5]] = "Lodgepole"
          } else if (mode ==2) { # specifcity order, highest to lowest, update unassigned only
            tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > threshs[2]]="Cotton&Willow"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > threshs[6]] = "Ponderosa"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > threshs[4]] = "Krummholz"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > threshs[3]] = "DouglasFir"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > threshs[5]] = "Lodgepole"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > threshs[1]] = "Aspen"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > threshs[7]] = "Spruce&Fir"
          } else if (mode ==3) { # specifcity order, lowest to highest, update unassigned only
            tds$EstTreeType[tds$EstTreeType=="X" & tds$SprFirProb > threshs[7]] = "Spruce&Fir"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$AspenProb > threshs[1]] = "Aspen"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$LodgeProb > threshs[5]] = "Lodgepole"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$DougFirProb > threshs[3]] = "DouglasFir"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$KrummProb > threshs[4]] = "Krummholz"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$PonderProb > threshs[6]] = "Ponderosa"
            tds$EstTreeType[tds$EstTreeType=="X" & tds$CotWilProb > threshs[2]]="Cotton&Willow"
          } else { # specifcity order, lowest to highest, update all
            tds$EstTreeType[tds$SprFirProb > threshs[7]] = "Spruce&Fir"
            tds$EstTreeType[tds$AspenProb > threshs[1]] = "Aspen"
            tds$EstTreeType[tds$LodgeProb > threshs[5]] = "Lodgepole"
            tds$EstTreeType[tds$DougFirProb > threshs[3]] = "DouglasFir"
            tds$EstTreeType[tds$KrummProb > threshs[4]] = "Krummholz"
            tds$EstTreeType[tds$PonderProb > threshs[6]] = "Ponderosa"
            tds$EstTreeType[tds$CotWilProb > threshs[2]]="Cotton&Willow"
          }
          #accuracy = (sum(tds$EstTreeType == tds$CovName))/nrow(tds)

          result=calcConfusionMatrix(tds,0)
          # accuracy=result[1]^2 + result[2]^2 # sensitivity^2 + specificity^2
          accuracy=result[1] + result[2]       # sensitivity   + specificity

          # reset data
          tds$EstTreeType="X"

          # print thresholds
          if (printLevel > 0) {
            printAccuracy = as.integer(accuracy * 100000)/1000.0
            print(paste("Accuracy(",threshs[1],threshs[2],threshs[3],
                      threshs[4],threshs[5],threshs[6],threshs[7],")=",
                      printAccuracy, ", i=",i,", j=",j,", bestThresh=",bestThresh))
```

```
          }

          # if accuracy improves, save best accuracy and threshold
          if (accuracy > bestAccuracy) {
            bestAccuracy = accuracy
            bestThresh = curThresh
          }
          curThresh = curThresh + increment
          if (curThresh > end) more = FALSE
        }

        # set new start, end and increment
        start = bestThresh - increment
        end = bestThresh + increment
        increment = increment / 10.0
        if (start <= 0.0) start = 0.0 + increment
        if (end >= 1.0) end = 1.0 - increment

        curThresh = start
      }

      threshs[j]=bestThresh
    }
  }

  if (printLevel) print(table(tds$EstTreeType))

  c(bestAccuracy,threshs)
}
```