Forest Cover Type Classification
Banatwalla, Muhammad
December 14th 2020

Introduction

Intro: The data set we used in this project comes from the University of Irvine Repository, The link of the Data set is as follows

Forest Cover_type Data set
The study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.

The Challenge for us with this data is to accurately predict forest cover type from cartographic variables only (no remotely sensed data). The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data.Our original data was in raw form (not scaled) and contained binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).

We have used a portion of the data to train our model with, the portion of the data we used comes from the following link:

Data used for training

The data has a total of 15120 cases and is well balanced with 2160 cases for each class. Furthermore we have in our data set 65 features, of these 44 are binary features which we will convert to numerical by number assignments, more details are given below.

Additional details

Cover_type - Our response variable, there are 7 distinct values from 1-7, the specific class these numbers represent are given below.

1 - Spruce/Fir

2 - Lodgepole Pine
3 - Ponderosa Pine
4 - Cottonwood/Willow
5 - Aspen
6 - Douglas-fir
7 - Krummholz

We will use the numbers (1-7) to represent the classes in our data set

Elevation - Elevation in meters of the area

Aspect - Aspect in degrees azimuth

Slope - Slope in degrees

Horizontal_Distance_To_Hydrology - The horizontal distance to the nearest surface water features

Vertical_Distance_To_Hydrology - The vertical distance to the nearest surface water features

Horizontal_Distance_To_Roadways - The horizontal distance to the to nearest roadway

Hillshade_9am (0 to 255 index) - Hillshade index at 9am, summer solstice

Hillshade_Noon (0 to 255 index) - Hillshade index at noon, summer solstice

Hillshade_3pm (0 to 255 index) - Hillshade index at 3pm, summer solstice

Horizontal_Distance_To_Fire_Points - The horizontal distance to the nearest wildfire ignition points

Wilderness_Area (4 binary columns, 0 = absence or 1 = presence) - Wilderness area designation

The wilderness areas represented by these numbers are:

1 - Rawah Wilderness Area

2 - Neota Wilderness Area

3 - Comanche Peak Wilderness Area

4 - Cache la Poudre Wilderness Area

Even though we have four binary columns for the data, we will convert them to a single column and set the values as 1,2,3,4 where each integer value represents the corresponding wilderness area as mentioned above.

Soil_Type (40 binary columns, 0 = absence or 1 = presence) - Soil Type designation

The soil types are:

1 Cathedral family - Rock outcrop complex, extremely stony.

2 Vanet - Ratake families complex, very stony.

3 Haploborolis - Rock outcrop complex, rubbly.

4 Ratake family - Rock outcrop complex, rubbly.

5 Vanet family - Rock outcrop complex complex, rubbly. 6

Vanet - Wetmore families - Rock outcrop complex, stony. 7

Gothic family.

8 Supervisor - Limber families complex.

9 Troutville family, very stony.

10 Bullwark - Catamount families - Rock outcrop complex,

rubbly. 11 Bullwark - Catamount families - Rock land complex,

rubbly. 12 Legault family - Rock land complex, stony.

13 Catamount family - Rock land - Bullwark family complex, rubbly.

14 Pachic Argiborolis - Aquolis complex.

15 unspecified in the USFS Soil and ELU Survey.

16 Cryaquolis - Cryoborolis complex.

17 Gateview family - Cryaquolis complex.

18 Rogert family, very stony.

19 Typic Cryaquolis - Borohemists complex.

20 Typic Cryaquepts - Typic Cryaquolls complex.

21 Typic Cryaquolls - Leighcan family, till substratum complex.

22 Leighcan family, till substratum, extremely bouldery. 23

Leighcan family, till substratum - Typic Cryaquolls complex. 24

Leighcan family, extremely stony.

25 Leighcan family, warm, extremely stony.

26 Granile - Catamount families complex, very stony. 27 Leighcan

family, warm - Rock outcrop complex, extremely stony. 28 Leighcan

family - Rock outcrop complex, extremely stony. 29 Como - Legault

families complex, extremely stony.

30 Como family - Rock land - Legault family complex, extremely stony.

31 Leighcan - Catamount families complex, extremely stony.

32 Catamount family - Rock outcrop - Leighcan family complex, extremely stony.

33 Leighcan - Catamount families - Rock outcrop complex, extremely stony.

34 Cryorthents - Rock land complex, extremely stony.

35 Cryumbrepts - Rock outcrop - Cryaquepts complex.

36 Bross family - Rock land - Cryumbrepts complex, extremely stony.

37 Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony.

38 Leighcan - Moran families - Cryaquolls complex, extremely stony.

39 Moran family - Cryorthents - Leighcan family complex, extremely stony.

40 Moran family - Cryorthents - Rock land complex, extremely stony.

Even though we have forty binary columns for the data, we will convert them to a single column with values 1-40 where each integer value represents the corresponding soil level as defined above.

PCA

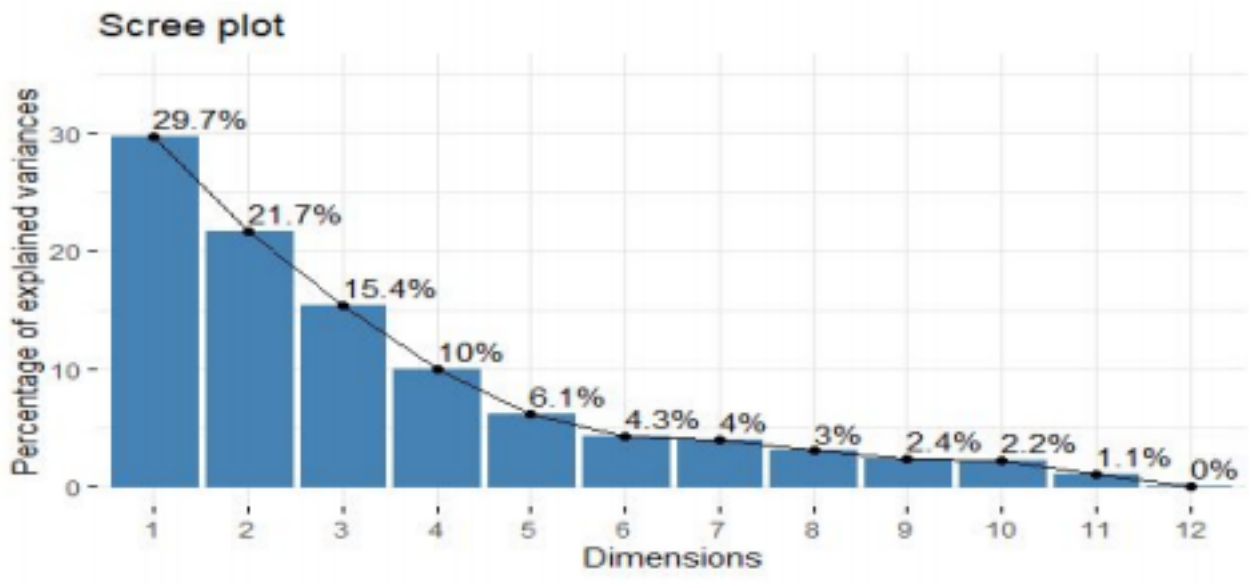
We standardized our data and performed Principal Component Analysis to find the features that account for the most variance in our data PCA works by computing principal components which are new variables that are constructed as linear combinations of the initial variables. PCA tries to put maximum possible information in the first component then the maximum remaining in the second component and so on Some useful plots are given below;

	eigenvalue	variance.percent	cumulative.variance.percent
Dim.1	3.57	29.73	29.73
Dim.2	2.30	21.69	51.41
Dim.3	1.85	15.44	66.86

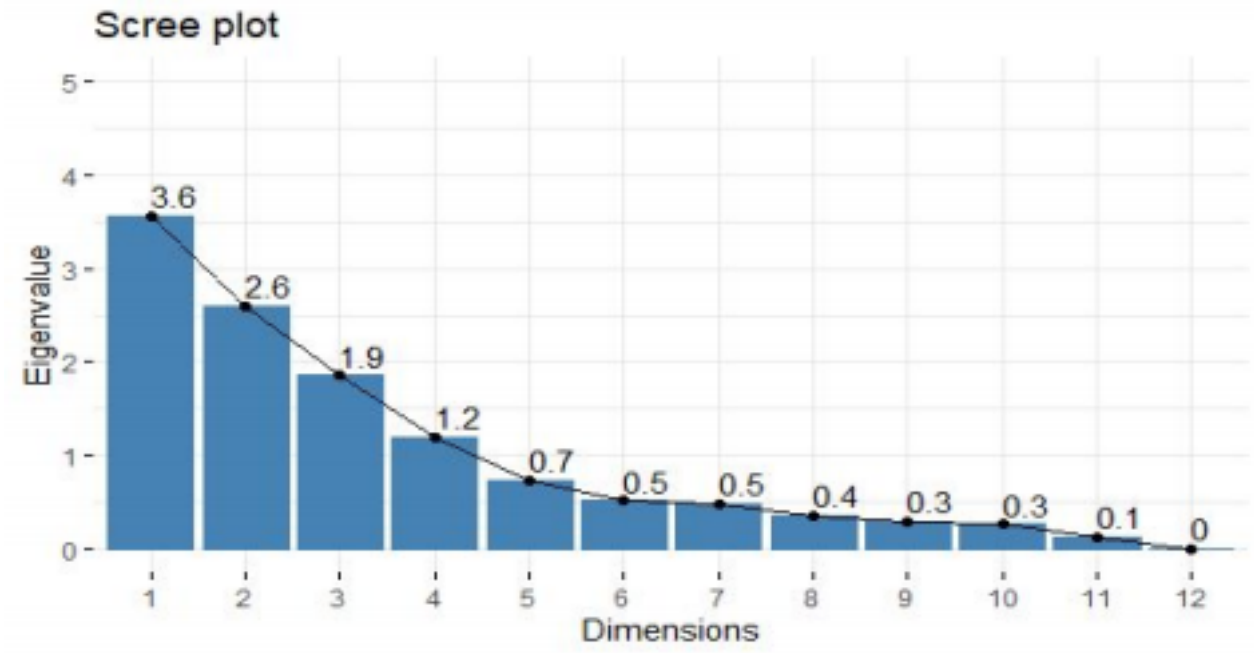
Table 1

Table 1 shows the eigenvalues of our top 3 features as well as the percent of variance they contribute in our model, from the cumulative variance column we can tell

that our top 3 features contribute to 66.86% variance in our data.



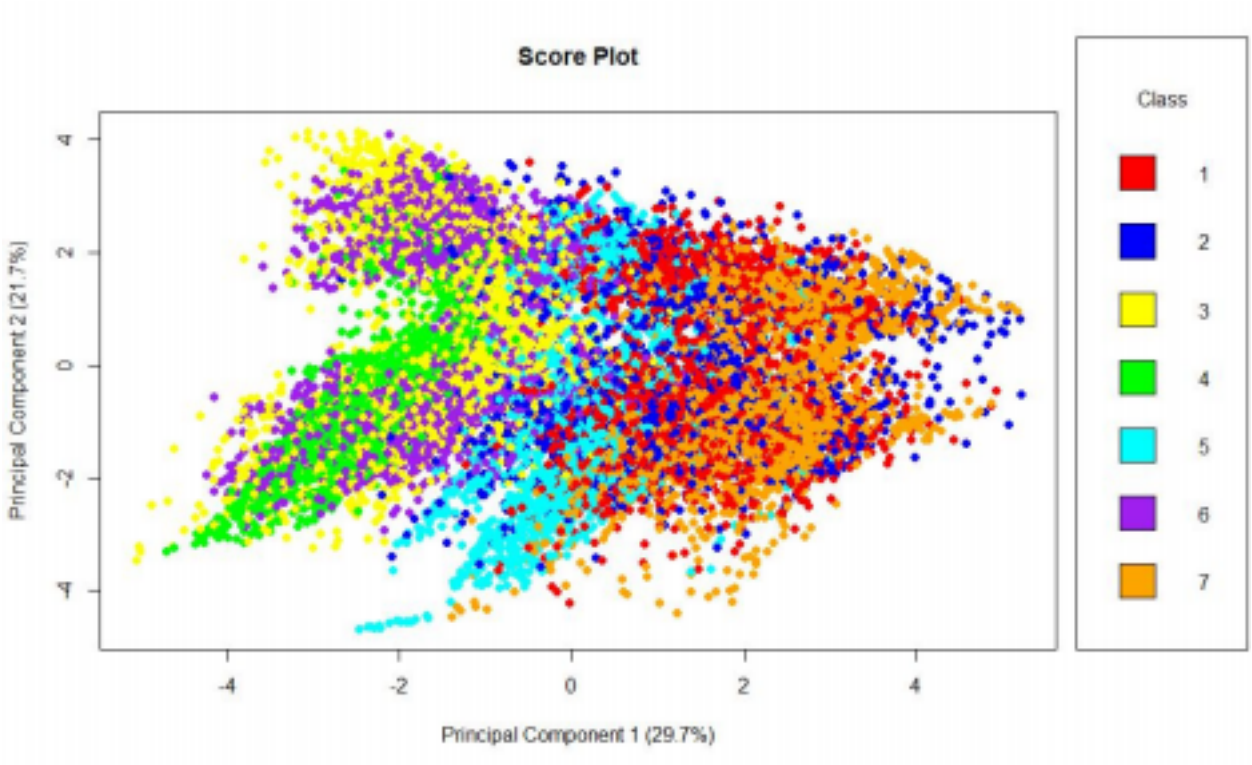
Graph 1



Graph 2

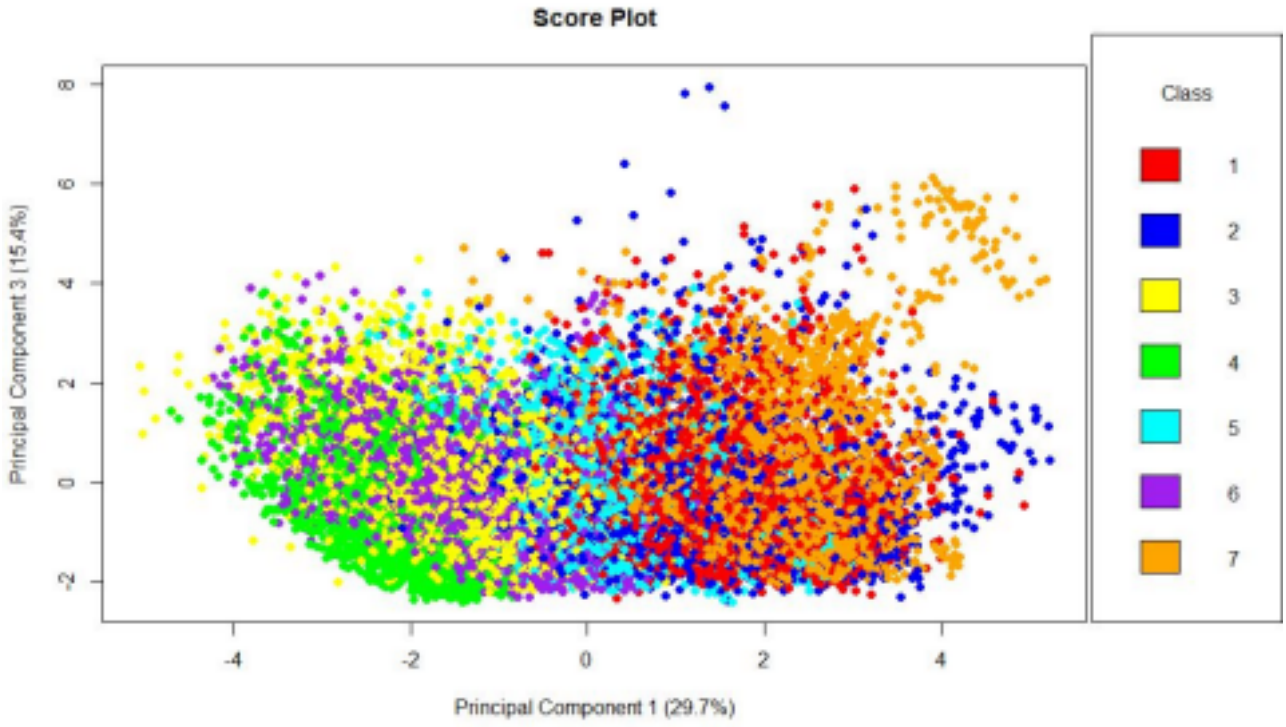
Graphs 1 and 2 represent the reduction in variance explained as percentages and as values respectively. From the graphs we can tell that 96.7% of the variance in our data is explained by the top 9 features.

We wanted to plot the results from our PCA on a score plot to visualize the separation in our data, This will help us get a rudimentary sense of how well the K_mean algorithm might perform on our data. We have plotted the score plots comparing the performances below.



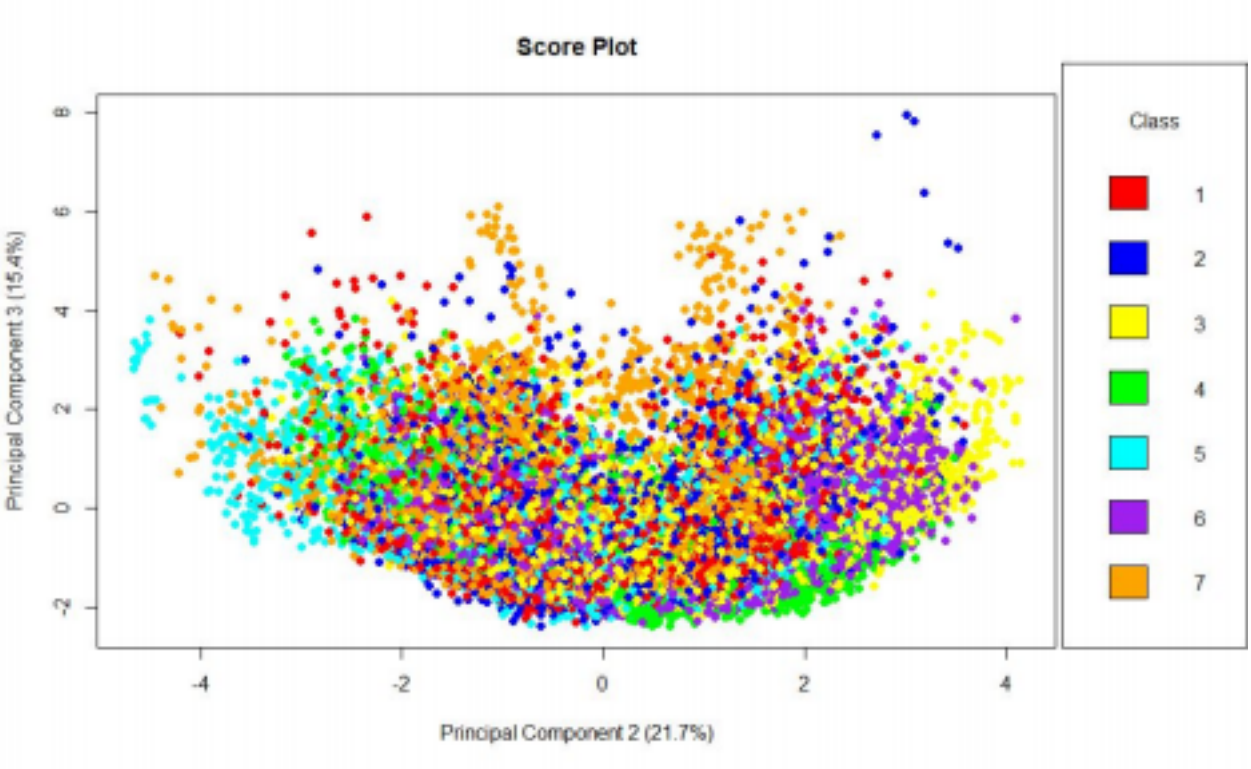
Graph 3- Score plot PC1 vs PC2

Based on the score plot we can see that our data is clumped close together and there aren't any visible clusters, however by looking at the spread of classes we can see that our clustering algorithm might yield useful results. For PC1 we get the highest scores for classes 1 and 7 and the lowest for classes 3,4 and 6. Class 5 is around the middle and revolves between 0 and -2 and class 2 is spread out the most with its values being the most varied of all the classes. PC2 does not look like a good discriminator of the classes as all classes seem to overlap each other in the axis. There seems to be some discrimination however with class 5 and class 7 having some cases lower then the other classes. Overall due to the good dicrimination of PC1 we can see clusters separated to some extent on our data but due to overlap we are not very confident of the classification ability of our k- means algorithm based on this information.



Graph 4

Once again based on the score plot we can see that our data is clumped close together and there aren't any visible clusters, however by looking at the spread of classes we can see that our clustering algorithm might yield useful results. For PC1 we get the highest scores for classes 1 and 7 and the lowest for classes 3,4 and 6. Class 5 is around the middle and revolves between 0 and -2 and class 2 is spread out the most with its values being the most varied of all the classes. PC3 seems to have all classes mixed up but amongst the highest scores there are cases with classes 7 and 2. Overall due to the good dicrimination of PC1 we can see clusters separated to some extent on our data but due to overlap we are not very confident of the classification ability of our k means algorithm based on this information.

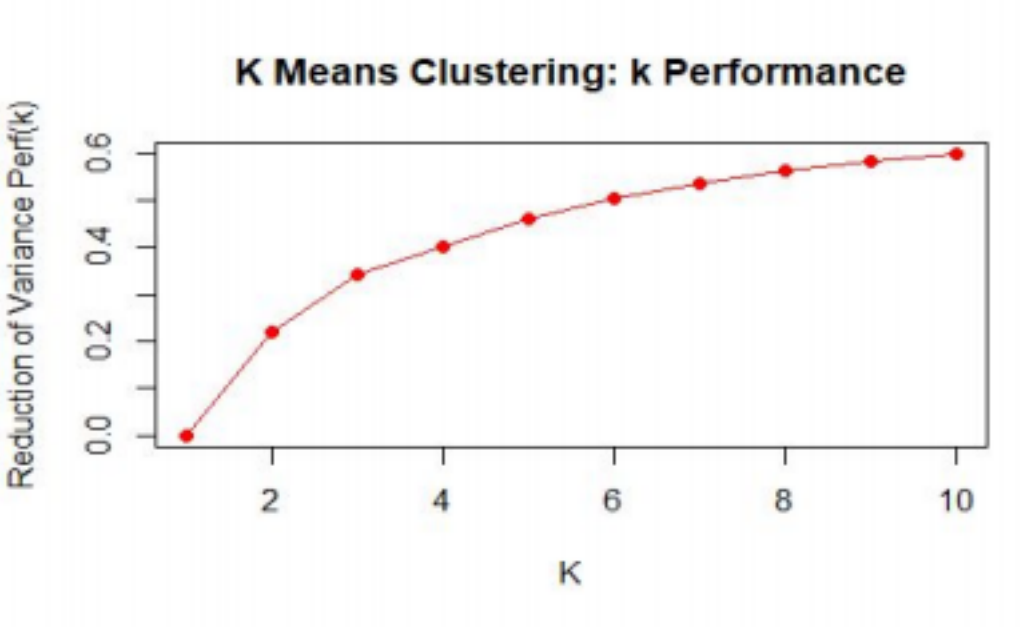


Graph 5

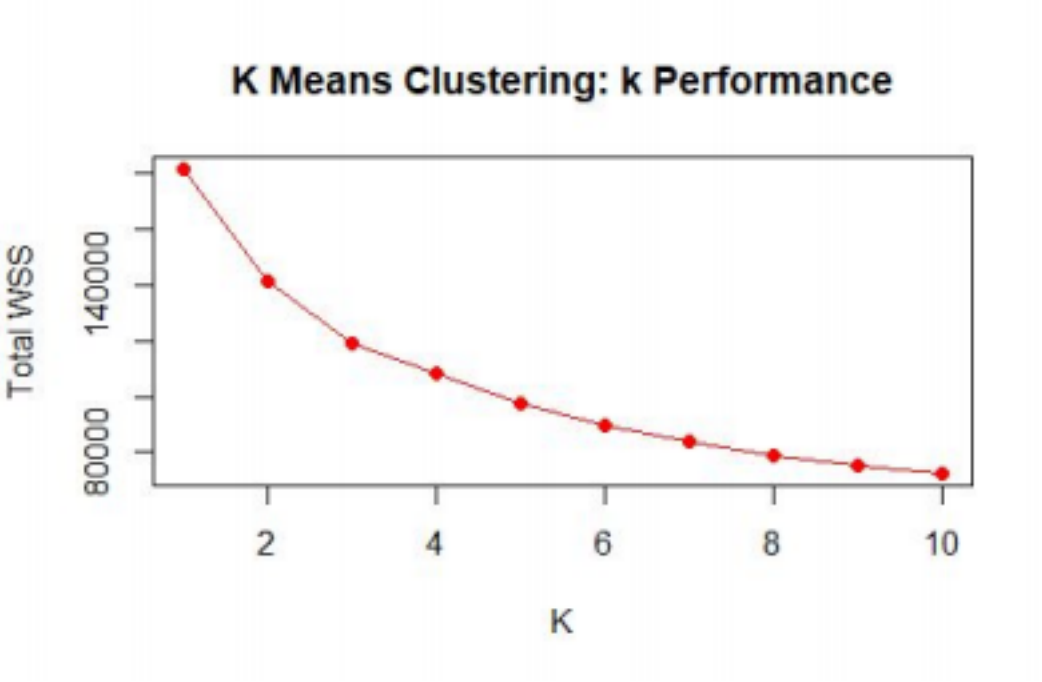
PC2 does not look like a good discriminator of the classes as all classes seem to overlap each other in the axis. There seems to be some discrimination however, with class 5 and class 7 having some cases lower then the other classes. PC3 seems to have all classes mixed up but amongst the highest scores there are cases with classes 7 and 2. Overall since both PC2 and PC3 are not good discriminators therefore the overall results are not great and our classes are all mixed up.

K Means

K Means Clustering splits the data into k groups that are similar to each other in terms of their placement in the data. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum.Thus, the notion of similarity is derived by how close a data point is to the centroid of the cluster. Here we will attempt to use K-Means on our cover_type data set to attempt to separate the 7 classes. Here are the performance levels for the different values of k:



Graph 6



Graph 7

K Means Clustering is an unsupervised machine learning algorithm, it will not have reference to the known cover type classes . In order to determine the optimal number of clusters for our data we used the elbow method. The “elbow” is the point where the sum of squared errors between the data points and their centers begins to flatten out as k

increases. It is not clear from our plots what the optimal level of k should be so we will do the k-means clustering for a few values of k and compare our results using gini impurity values. The gini impurity is a way to measure the quality of our clusters. It measures how well the clusters reveal the underlying true classes of the data.

3 Clusters

Cover Type	Cluster 1	Cluster 2	Cluster 3
1	1763	253	144
2	1540	295	325
3	32	971	1157
4	0	498	1662
5	641	446	1073
6	66	1069	1025
7	2025	42	93
Sum	6067	3574	5479
Gini Impurity	0.73	0.79	0.79

Table 2: 3 clusters Class quantities and Gini Impurity

Total gini impurity= 0.77

Table 2 shows us the distribution of cases in each cluster, since we had 7 classes and only 3 clusters and since based on the PCA score plots we noted our data is clumped together, the gini impurity was understandably high. The impurity for cluster 1 is 0.73, the impurity for cluster 2 is 0.79, the impurity for cluster 3 is also 0.79. The total gini impurity, which is the weighted average of the impurities of the cluster came out to be 0.77.

Cover_Type	Cluster 1	Cluster 2	Cluster 3	Cluster4
1	1423	202	101	434
2	1273	247	246	394

3	36	971	1129	24
4	0	503	1656	1
5	715	381	871	193
6	94	1065	993	8
7	1211	31	65	853
Sum	4752	3400	5061	1907
Gini Impurity	0.75	0.78	0.77	0.70

Table 3: 4 clusters Class quantities and Gini Impurity

total gini impurity= 0.76

Table 3 shows us the distribution of cases in each cluster, since we had 7 classes and only 4 clusters; and since based on the PCA score plots we noted our data is clumped together, the gini impurity was understandably high. The impurity for cluster 1 is 0.75, the impurity for cluster 2 is 0.78, the impurity for cluster 3 is also 0.77, and the impurity for cluster 4 is 0.70. The total gini impurity, which is the weighted average of the impurities of the cluster came out to be 0.76.

Cover_Type	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	81	96	1381	400	202
2	216	131	1205	375	233
3	859	456	5	18	822
4	1081	716	0	1	362
5	488	580	579	172	341
6	656	462	31	5	1006
7	2	105	1194	825	34
Sum	3383	2546	4395	1796	3000
Gini Imp	0.77	0.80	0.73	0.67	0.77

Table 4: 4 clusters Class quantities and Gini Impurity

total gini impurity= 0.76

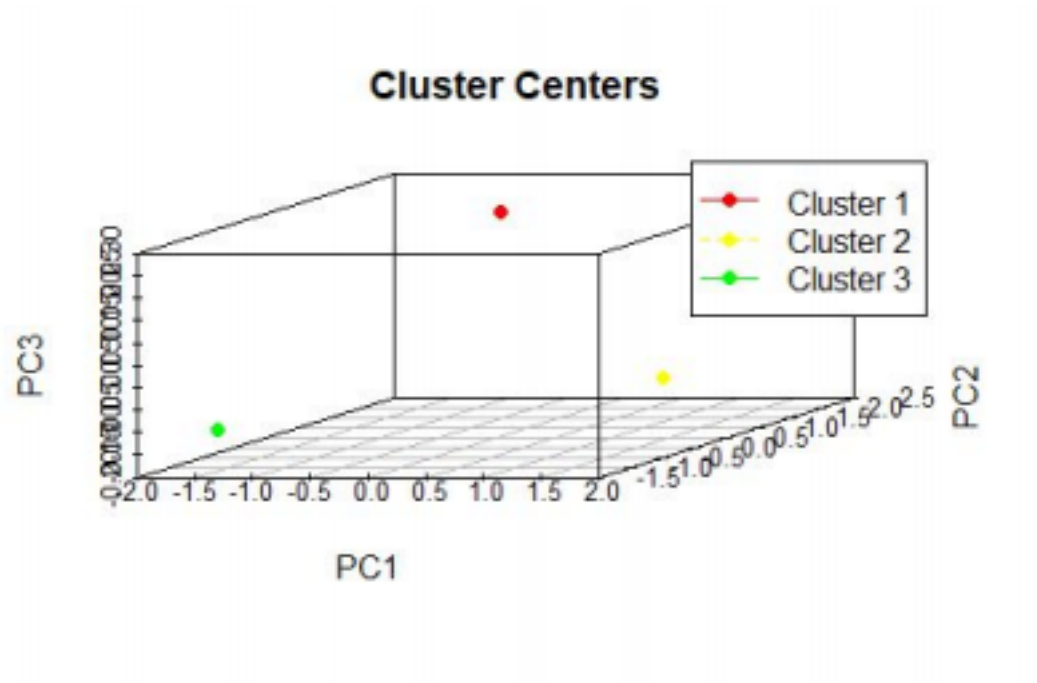
Table 4 shows us the distribution of cases in each cluster, since we had 7 classes and only 5 clusters; and since based on the PCA score plots we noted our data is clumped together, the gini impurity was understandably high. The impurity for cluster 1 is 0.77, the impurity for cluster 2 is 0.80, the impurity for cluster 3 is also 0.73, the impurity for cluster 4 is 0.67 and the impurity for cluster 5 is 0.77. The total gini impurity, which is the weighted average of the impurities of the cluster came out to be 0.76.

Since changing the clusters did not have a massive impact on our gini impurity we chose to go with 3 clusters additional details about cluster 3 are given below.

Clusters Cover Type	Cluster1	Cluster 2	Cluster 3
1	12%	82%	7%
2	14%	71%	15%
3	45%	1%	54%
4	23%	0%	77%
5	21%	30%	50%
6	49%	3%	47%
7	2%	94%	4%

Table 5: class percentages in each cluster

Table 5 shows us the percentages of cases each cluster has of any class, for example from the table we can see that 12 percent of the cases with class 1 are in cluster 1; 82% of the cases with class 1 are in cluster 2; and 7% of the cases with class 1 are in cluster 3 and so on. Overall our cluster 2 does a good job at isolating cases of class 7, 1 and 2 and has barely any cases of class 6,4,and 3. Cluster 3 does a good job at isolating class 4 and does not have many cases for class 1 and class 7. Cluster 1 does not do a good job at separating any class, even though it has the lowest impurity.



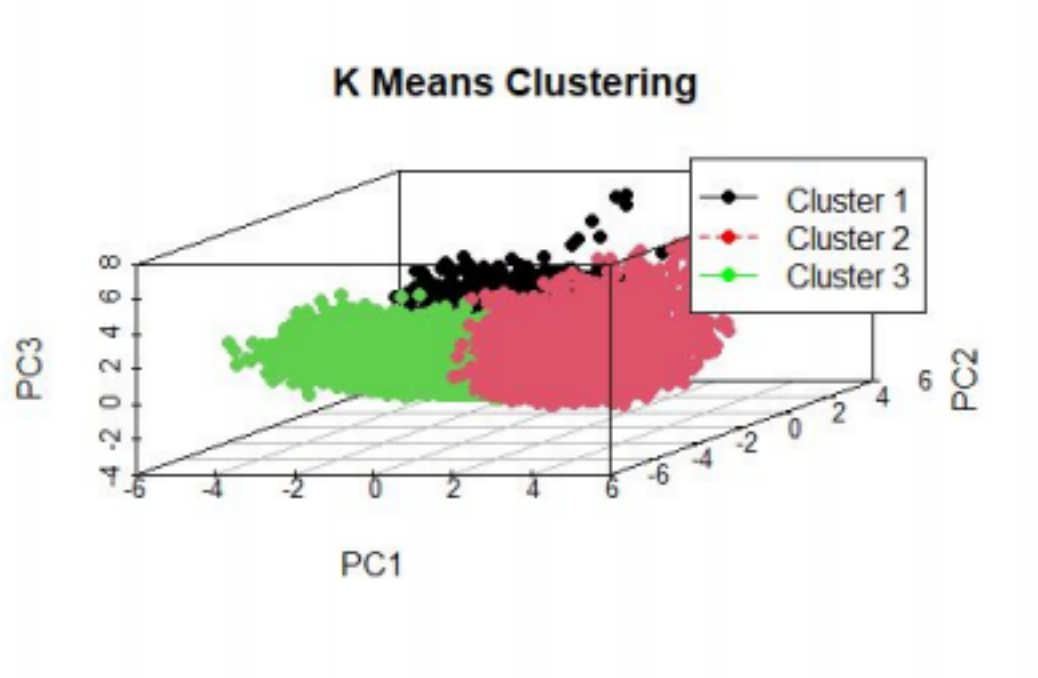
Graph 8

	PC1	PC2	PC3
Cluster 1	-0.88	2	0.23
Cluster 2	1.90	-0.29	-0.03
Cluster 3	-1.53	-1.09	-0.11

Table 6 : Centroid coordinates

Table 6 and graph 8 give us the position of the centroids on a 3d graph with PC1 PC2 and PC3 The points represent our three centeroids, they give us a rough idea about where our clusters will be. We can see that they are reasonably distanced and help us get an estimation of the shape of our data when taking into account the three top PCA's.

In dimension reduction by principal component analysis, principal component one explains the largest amount of variation in our data (29.7%), principal component 2 explains the second most variation (21.7%), and principal component 3 explains the third most variation in our data (15.4%). Therefore, a lot of the information from our dataset can be found in the first three principal components. We plotted below each data point in principal component 1, 2, and 3 colored according to its cluster assignment:



Graph 9: All cases colored according to cluster on 3d Plot

Graph 9 Gives us all the data points in our clusters, from table 2 we know that we have 6067 cases in cluster 1, 3575 cases in cluster 2 and 5479 cases in cluster 3.

Random Forests

Before one talks about random forest it is important to understand decision trees, decision trees are a decision making algorithm that can be used for both regression and classification tasks. While they are easy to build, use and interpret, they are also unfortunately very inaccurate. They are very inflexible making them bad at interpreting new data that comes in.The random forest algorithm is in fact the decision tree algorithm applied again and again. Random forest makes 100s of decision trees for any particular case and then applies a voting like system to determine the class that the case will fall into. Thus random Forest uses the simplicity of decision trees but the variability of trees also adds flexibility for all new incoming data!

The randomForest() function in R can take in a various number of parameters depending on the structure of the data. This function implements Breiman's random forest algorithm. The function can be used for both regression and classification tasks. For our purposes, we will use the parameters formula, data, ntree, mtry and importance. There are many other parameter options such as sampsize, nodesize, maxnodes, etc. This function returns an object of class randomForest with a list of values. The following table shows the outputs with explanations:

Output Value	Description
call	the original call to randomForest
type	either regression, classification, or unsupervised
predicted	predicted values of the input data
importance	a matrix with mean decrease values
importanceSD	standard errors
localImp	a matrix containing casewise importance measures
ntree	number of trees grown
mtry	number of predictors sampled for spliting at each node
forest	a list that contains the entire forest
err.rate	vector error rates of the prediction
confusion	the confusion matrix of the prediction
votes	gives the fraction or number of (OOB) ‘votes’ from the random forest
oob.times	number of times cases are out of bag

Table 7:randomForest() Output Description

For our implementation of the random forest algorithm we will use a training data set that will amount to around 80% of our data and will consist of 80% of cases form each class. The number of features in our data was 12, so we set our mtry value to the square root of 12 which is approximately 3. We set the importance to be true as we wanted to find which features perform the best in discriminating between the classes.

We ran the random forest alogarithm with different number of trees and have posted our results below. The tables below show the confusion matrices with different number of trees

Random Forest confusion matrix 100 Trees

Actual	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7
Actual 1	75.23%	17.13%	0.00%	0.00%	2.08%	0.69%	4.86%
Actual 2	12.50%	70.60%	1.39%	0.00%	11.57%	2.55%	1.39%
Actual 3	0.00%	0.69%	80.79%	4.40%	2.55%	11.57%	0.00%
Actual 4	0.00%	0.00%	2.31%	95.14%	0.00%	2.55%	0.00%
Actual 5	0.00%	3.01%	0.23%	0.00%	95.83%	0.93%	0.00%
Actual 6	0.00%	0.23%	8.80%	2.08%	0.23%	88.66%	0.00%
Actual 7	2.78%	0.00%	0.00%	0.00%	0.00%	0.00%	97.22%

Table 8:Random Forest confusion matrix 100 Trees Test Set(computation time 42 seconds)

Actual	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7
Actual 1	77.55%	15.51%	0.00%	0.00%	1.85%	0.69%	4.40%
Actual 2	14.58%	69.21%	1.62%	0.00%	10.65%	2.55%	1.39%
Actual 3	0.00%	0.46%	80.09%	5.09%	2.55%	11.81%	0.00%

Actual 4	0.00%	0.00%	1.85%	95.83%	0.00%	2.31%	0.00%
Actual 5	0.00%	2.55%	0.23%	0.00%	96.06%	1.16%	0.00%
Actual 6	0.00%	0.23%	9.03%	2.31%	0.23%	88.19%	0.00%
Actual 7	3.01%	0.00%	0.00%	0.00%	0.00%	0.00%	96.99%

Table 9:Random Forest confusion matrix 200 Trees Test Set(computation time 42 seconds)

Actual	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7
Actual 1	78.01%	15.28%	0.00%	0.00%	1.85%	0.69%	4.17%
Actual 2	15.05%	69.44%	1.16%	0.00%	10.19%	3.01%	1.16%
Actual 3	0.00%	0.46%	80.32%	4.63%	2.31%	12.27%	0.00%
Actual 4	0.00%	0.00%	2.08%	95.60%	0.00%	2.31%	0.00%
Actual 5	0.00%	2.78%	0.23%	0.00%	95.83%	1.16%	0.00%
Actual 6	0.00%	0.23%	8.80%	2.08%	0.46%	88.43%	0.00%
Actual 7	3.24%	0.00%	0.00%	0.00%	0.00%	0.00%	96.76%

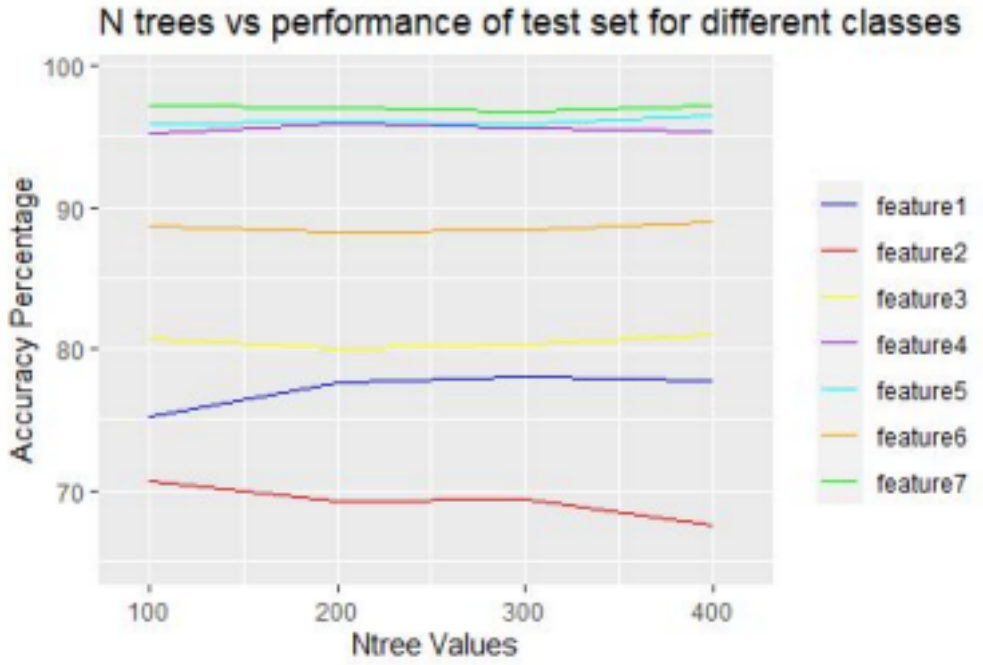
Table 10:Random Forest confusion matrix 300 Trees Test Set(computation time 45 seconds)

Actual	Predicted 1	Predicted 2	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7
Actual 1	77.78%	15.51%	0.00%	0.00%	1.85%	0.69%	4.17%
Actual 2	16.44%	67.59%	0.93%	0.00%	10.88%	2.78%	1.39%
Actual 3	0.00%	0.46%	81.02%	4.63%	2.31%	11.57%	0.00%
Actual 4	0.00%	0.00%	2.31%	95.37%	0.00%	2.31%	0.00%
Actual 5	0.00%	2.31%	0.23%	0.00%	96.53%	0.93%	0.00%

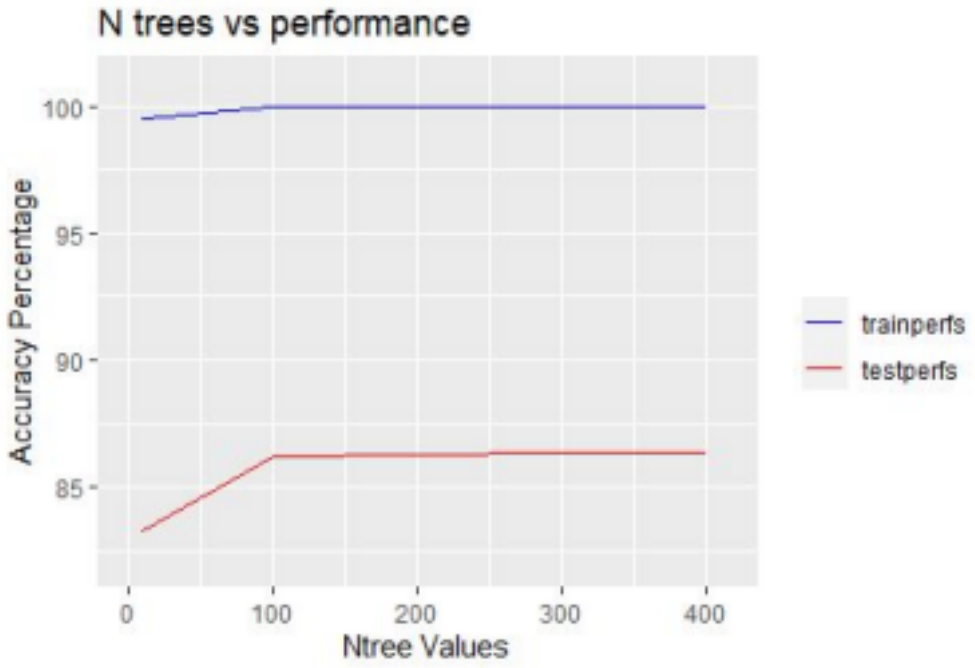
Actual 6	0.00%	0.23%	8.56%	2.08%	0.23%	88.89%	0.00%
Actual 7	2.78%	0.00%	0.00%	0.00%	0.00%	0.00%	97.22%

Table 11 :Random Forest confusion matrix 400 Trees Test Set(computation time 48 seconds)

Tables 8,9,10 and 11 shows us the confusion matrices for tree values 100,200 ,300 and 400 respectively, the diagonal element of our matrices represent our class accuracies, we have also represented these accuracies in graph 10 below, the overall accuracies for both our test and training set are in graph 11



Graph 10: Test set accuracies by different classes



Graph 11: Test and Training set accuracies

Graph 11 shows us the overall accuracy of our models with different tree levels, usually as the number of tree increases the better the performance and the more the computation cost, so we
should stop when it is not longer worth increasing trees, For us we got the same results for when trees are 300 and 400 so we chose to go with 300 as our BESTRF.

Importance of variables

We wanted to compute which variables have the largest impact on our random forest classifiers, our findings are documented below.

	1	2	3	4	5	6	7
Elevation	105.55	38.08	39.94	73.78	88.11	63.48	64.08
Aspect	19.14	14.88	27.22	21.95	32.97	30.97	23.03
Slope	15.69	10.46	20.14	19.34	28.85	29.24	19.04
Horizontal_Distance_To_H ydrology	21.98	29.43	31.71	56.52	42.21	50.60	34.24
Vertical_Distance_To_H yd rology	22.37	13.29	34.18	23.92	40.87	39.83	27.07

Horizontal_Distance_To_Roadways	26.65	30.08	41.56	30.45	65.47	43.27	38.92
Hillshade_9am	21.12	16.65	36.08	29.77	40.66	40.05	28.47
Hillshade_Noon	15.69	24.64	27.97	25.97	40.94	41.63	29.43
Hillshade_3pm	21.99	14.56	24.80	18.11	33.16	32.09	26.46
Horizontal_Distance_To_Fire_Points	20.10	24.92	33.05	17.40	65.80	50.61	45.48
Wildnerness_Area	18.23	25.78	25.22	39.70	30.63	31.35	22.73
Soil.Type	25.34	29.53	48.55	22.92	52.98	42.14	71.19

Table 12

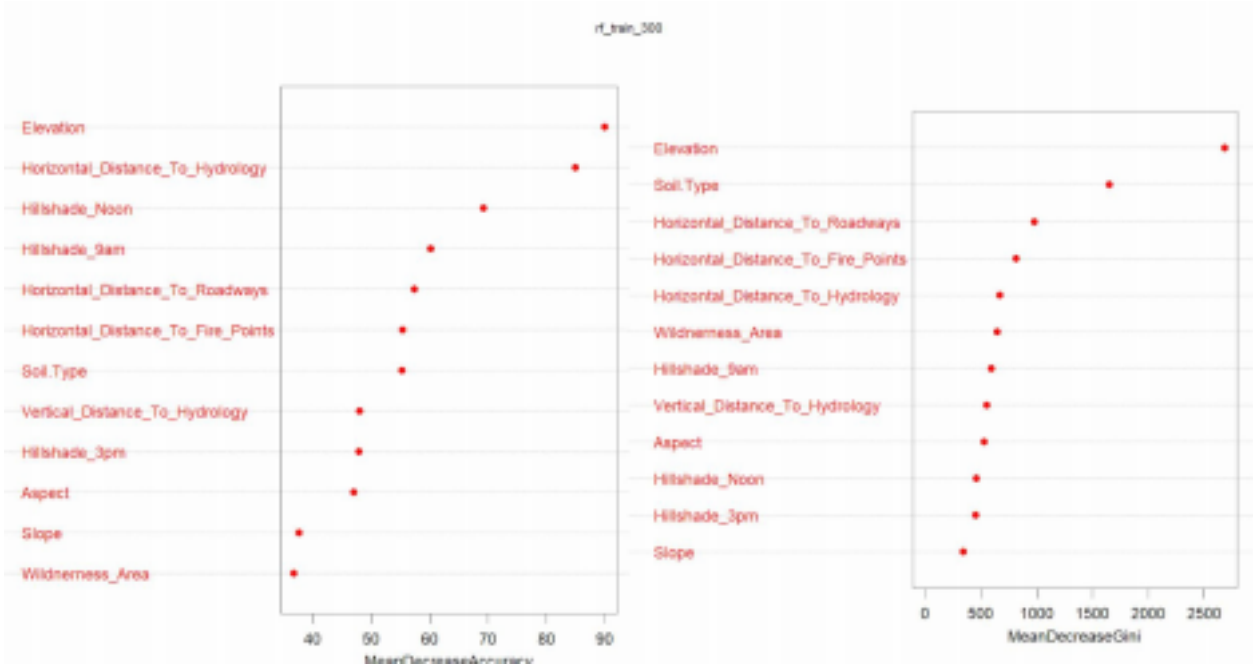
Table 12 shows the impact each variable has to each class, the columns have are classes and the rows have the different features from our dataset.

	MeanDecreaseAccuracy	MeanDecreaseGini
Elevation	89.97	2685.10
Aspect	47.01	528.58
Slope	37.61	338.87
Horizontal_Distance_To_Hydrology	84.98	665.16
Vertical_Distance_To_Hydrology	47.98	552.06
Horizontal_Distance_To_Roadways	57.37	979.79
Hillshade_9am	60.20	589.98
Hillshade_Noon	69.25	460.25
Hillshade_3pm	47.93	451.01
Horizontal_Distance_To_Fire_Points	55.34	816.74
Wildnerness_Area	36.71	645.39

Soil.Type	55.20	1651.70

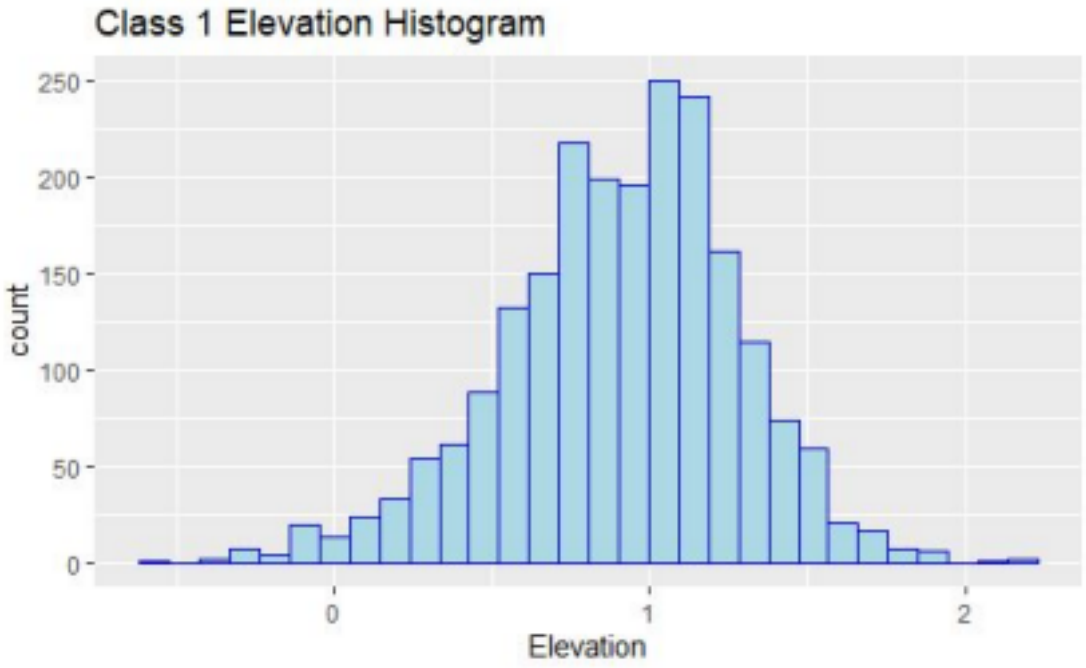
Table 13

Table 13 shows us the mean values of the impact on our variables with two parameters, Mean Decrease Gini And Mean Decrease Accuracy

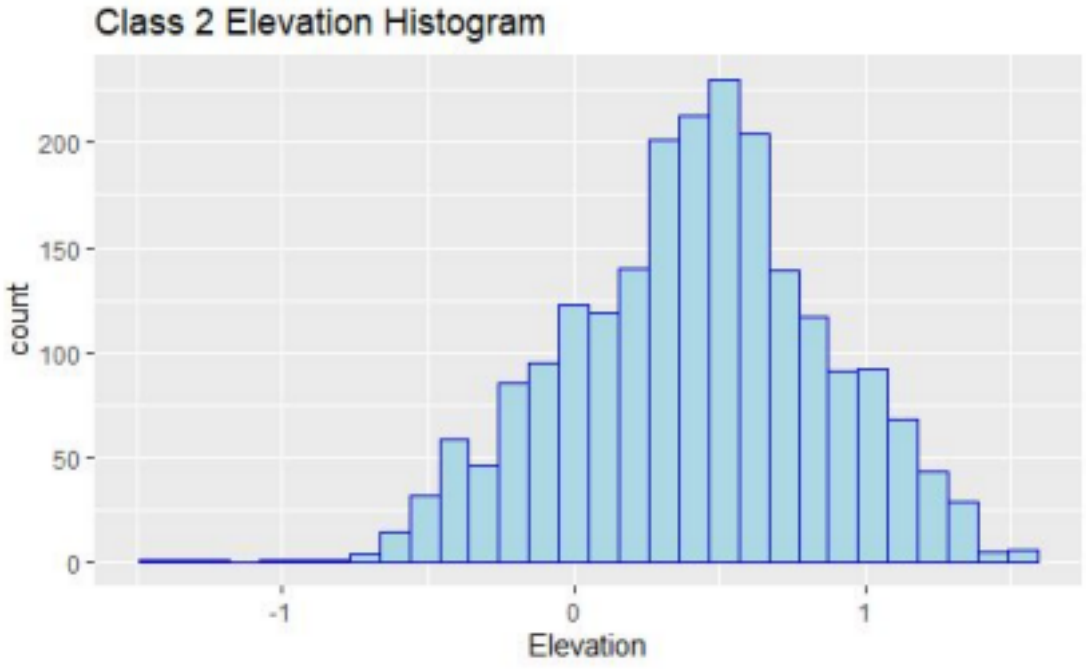


Graph 12

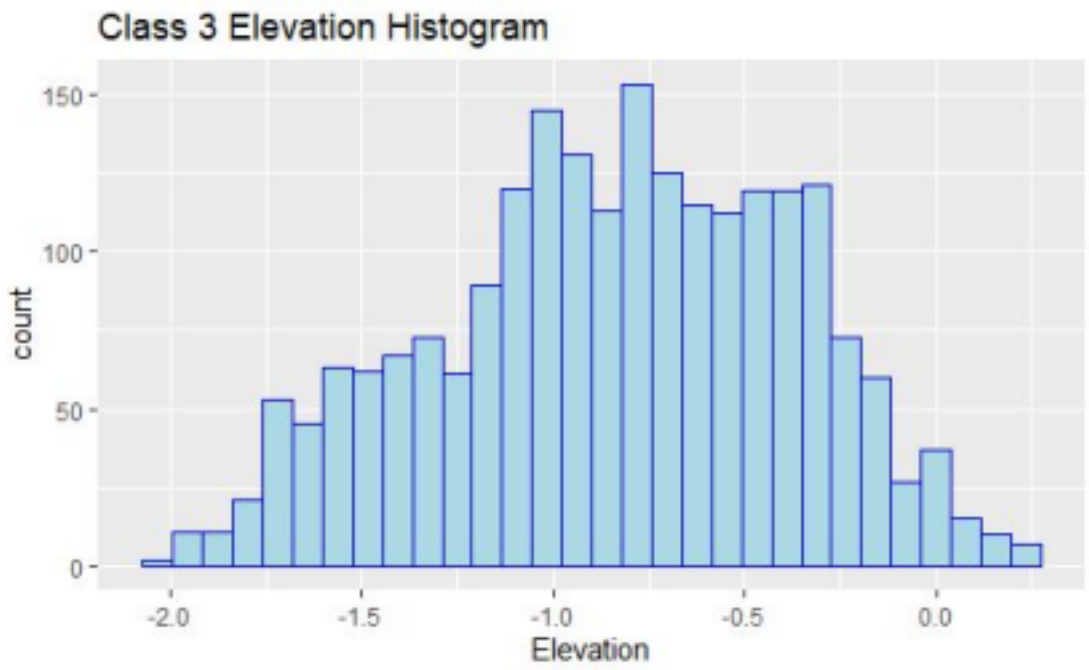
Graph 12, graphs the same information in table 13



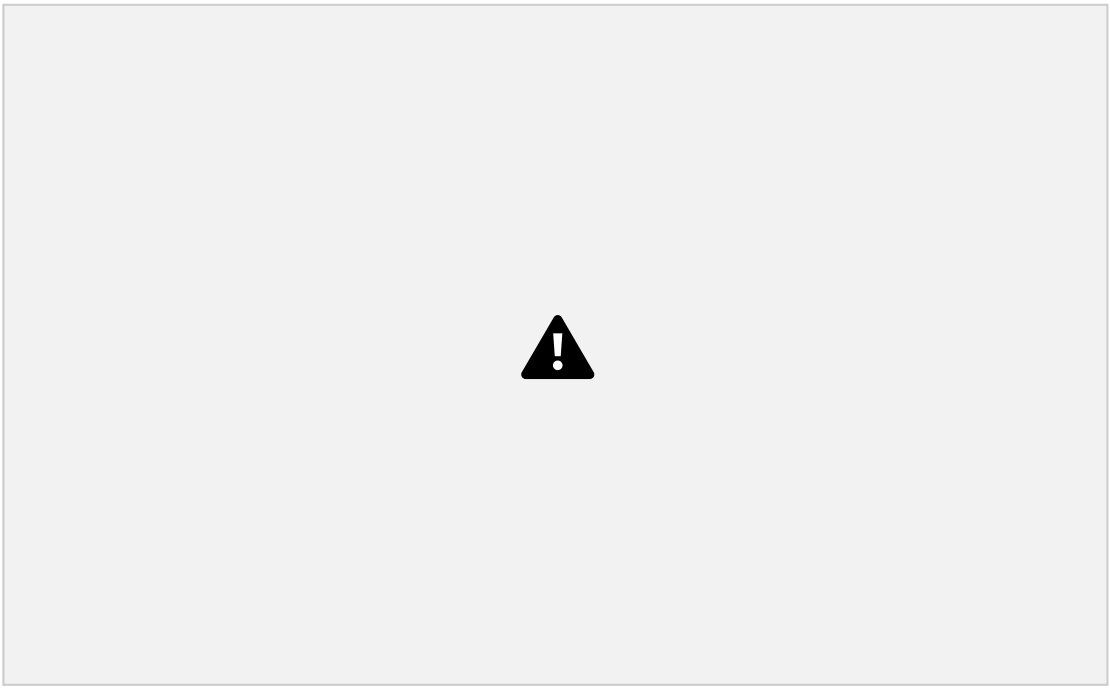
Graph 13



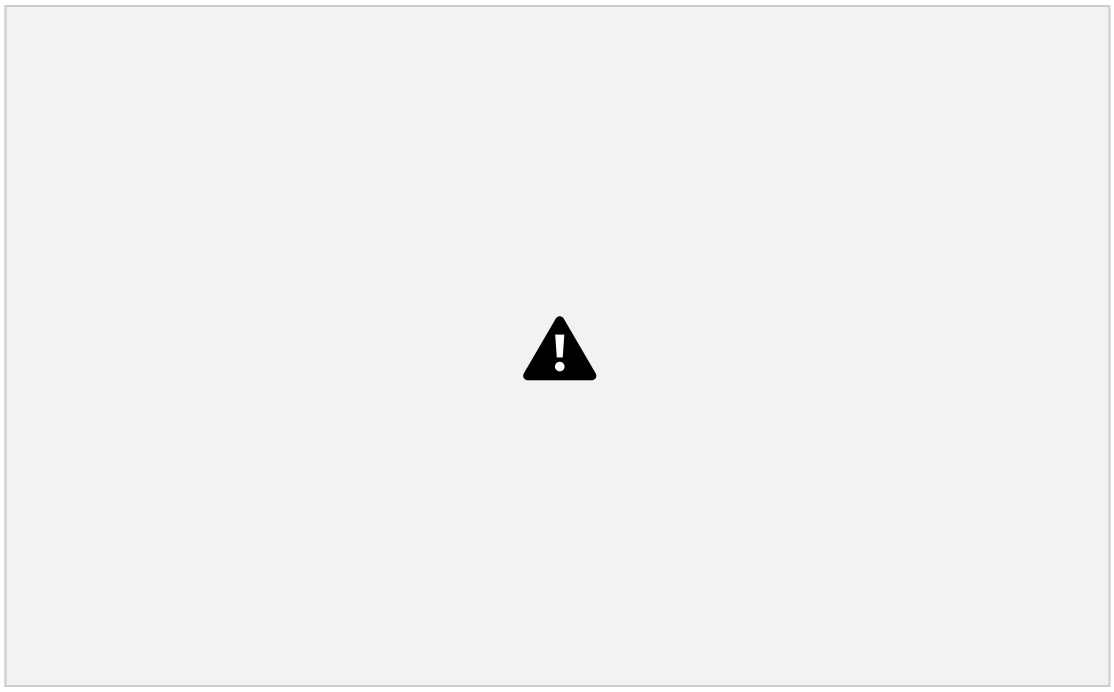
Graph 14



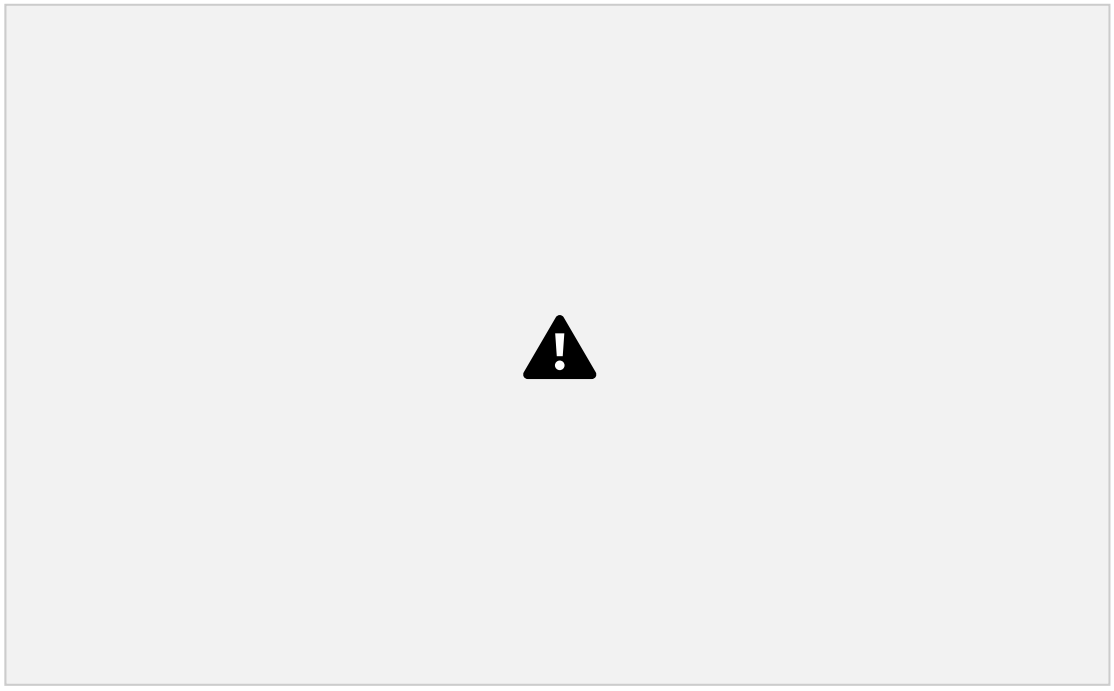
Graph 15



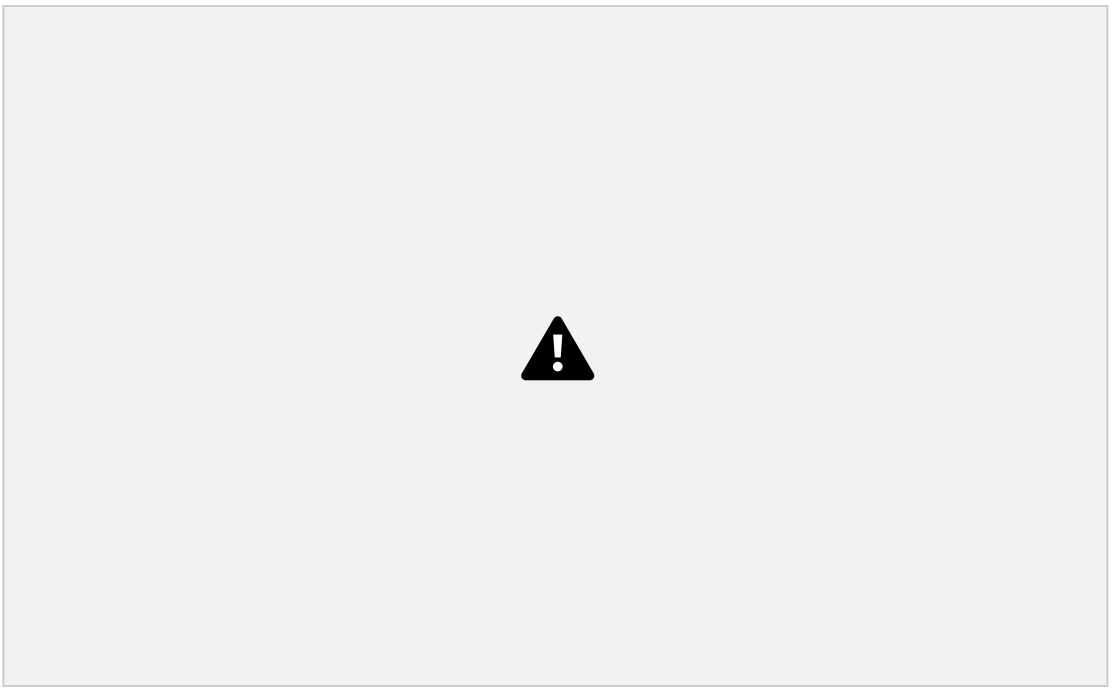
Graph 16



Graph 17



Graph 18



Graph 19

Graph 13 to graph 19 show us our histograms for our most important feature elevation, on different classes we are going to use KS tests to compare our features.

First Class	Second class	D	P value
1	2	0.49	0
1	3	0.97	0
1	4	1	0
1	5	0.85	0
1	6	0.96	0
1	7	0.67	0
2	3	0.83	0
2	4	0.99	0
2	5	0.47	0
2	6	0.8	0

2	7	0.9	0
3	4	0.51	0
3	5	0.51	0
3	6	0.01	0
3	7	1	0
4	5	0.97	0
4	6	0.61	0
4	7	1	0
5	6	0.81	0
5	7	1	0
6	7	1	0

Table 14 ks tests elevation

First Class	Second class	D	P Value
1	2	0.21	0
1	3	0.97	0
1	4	0.97	0
1	5	0.38	0
1	6	0.89	0
1	7	0.72	0
2	3	0.88	0
2	4	0.76	0
2	5	0.18	0

2	6	0.71	0
2	7	0.81	0
3	4	0.23	0
3	5	0.82	0
3	6	0.32	0
3	7	0.99	0
4	5	0.63	0
4	6	0.39	0
4	7	1	0
5	6	0.67	0
5	7	0.81	0
6	7	0.94	0

Table 14 ks tests wilderness area

Or ks tests show low values fo P which means that the distribution of our variable was different in our classes which explains how we could dicriminate

Matrix on cluster 1 overall accuracy 84% missing value 4

Class	**1**	**2**	**3**	**5**	**6**	**7**
1	277	51	0	3	0	22
2	69	221	0	15	0	3
3	0	0	2	3	1	0
5	0	8	1	118	1	0
6	0	0	0	1	12	0

7	13	3	0	0	0	389

Table 15

We performed Random forests on our poorest cluster and got an overall accuracy 82%, we were also missing our class 4, since the overall accuracy was lower then the random forest performed on the entire data set and there were no missing values in the original we conclude it is not useful to do random forest for each cluster individually.

	Actual 1	Actual 2
Predicted 1	1707	527
Predicted 2	453	1633

Table 16

Table 16 shows the confusion matrix when we used the SVM Learning algorithm on 2 classes of our data and noticed that the results were not great. We will need to look into it further to understand why.(computation time 30 seconds)

3 Conclusion and Further Suggestions

In conclusion, the random forest model was able to classify the cases of the specific areas with its corresponding cover_type really well. Random forest uses many methods that help increase results. Random Forest uses the simplicity of decision trees and adds flexibility by making many trees through bootstrapping the dataset. Then, the random forest model makes a decision by checking all the trees and seeing which result received the most votes. After finding which best number of trees yielded the best results, we verified that the results that we got on the test set was not just a matter of chance. We verified this by resplitting the data with a new training and test set. This yielded similar results. Even though we had good results with our best model there is always room for improvement. We then built three separate binary classifiers to classify each font individually but this didn't yield better results.

Since the performance of random forest often deteriorates when the number of features increases to further investigate we could try different methods of feature selection. In this project, we reduced the number of features through principal component analysis but we did notice that the order of importance

wasn't exactly the order of the principal components. We could try creating a model that only uses only features of high importance calculated

from our best random forest model. Then evaluate the performance on the test set.

Due to limitations on time we were not able to implement adaptive boosting to try to improve our results. Adaptive boosting is an algorithm very similar to random forest that can help improve our classification performance. Instead of using multiple trees it uses stumps which is a tree composed of just one variable instead of all of them. Using just one feature makes stumps weak learners but that is something Adaptive boosting relies on. In Random Forest each tree's vote is the same but in ada boost in the forest of stumps each stump's vote had different weightage to it. The weightage of the stump is calculated by the accuracy of its classification. Features that are better discriminators result in better stumps thus improving the weights of the stump. Another important distinction is that in random forests each tree is made independently so it does not matter which tree is made first. In contrast, in Ada boost the order is important as the errors one stump makes influences how the second stump is made, and the errors made by the second stump influence the third stump and so on. . . The way that works is that each case is initially given an equal amount of weight, then after the first stump is run, which is the stump based on the feature that is the best discriminator, the cases that are correctly classified get a reduction in weight and the cases that are incorrectly classified get an increase. The amount of increase or decrease the weights get is dependent on how good of the classifier the first stump was. The new weights are normalized and we can then compute a weighted gini index to determine which feature should be used to create the next stump. The rest of the process continues and can be summarized by the following steps:

1. Choose the best classifier based on the gini weighted indexes computed from the pre vious stump
2. Compute the accuracy of the stump
3. The cases that get correctly classified by this stump get a reduction in their weights and the ones incorrectly classified get an increase. The amount of increase or decrease is dependent on the accuracy of the stump
4. These new weights are normalized and we go back to step 1

The above steps are continued for a specified number of stumps, Generally the more stumps the better our classification but it also increases computing time so we have to make a decision for when it is no longer worth increasing the amount of stumps. At the end, for any particular case the weights of the stumps that put the case in different classes are added and the class that has the largest sum is

predicted for that case. This the power of vote for a stump depends on its weight/Accuracy.

R Script

```
standardize=function(x){
  return((x-mean(x))/sd(x))}
Data=ForestTypes[,2:14]
View(Data)
SDATA=as.data.frame(lapply(Data[,1:12],standardize))


res.pca <- prcomp(DATA, scale = TRUE)
#eigen values
eig.val <- get_eigenvalue(res.pca)
#Plot of Eigenvalues Lr versus r
fviz_eig(res.pca, choice = "eigenvalue", addlabels=TRUE,ylim = c(0, 5),ncp=35)
#Percentage of Variance explained versus r
fviz_eig(res.pca, ncp = 23,addlabels=TRUE,ylim = c(0, 35))


newdata = res.pca$x[,1:3]




#PCA
res.pca <- prcomp(SDATA, scale = FALSE)

?prcomp

#Eigenvector w1,w2,w3
eigenvec3 <- res.pca$rotation[,1:3]
library(factoextra)
library("FactoMineR")


#eigen values
eig.val <- get_eigenvalue(res.pca)
#Plot of Eigenvalues Lr versus r
fviz_eig(res.pca, choice = "eigenvalue", addlabels=TRUE,ylim = c(0, 5),ncp=35)
#Percentage of Variance explained versus r
fviz_eig(res.pca, ncp = 23,addlabels=TRUE,ylim = c(0, 35))
```

```
pca_vals=res.pca$x[,1:3]

newdata1 <- cbind(ForestTypes[,14],newdata)

newdata.df <- as.data.frame(newdata1)
#3 colors representing 3 fonts
treat_col <- c()
for (i in 1:15120) {
  if(newdata.df$Cover_Type[i] == '1') {
    treat_col <- append(treat_col, 'red')
  }
  else if(newdata.df$Cover_Type[i] == '2'){
    treat_col <- append(treat_col, 'blue') }
  else if(newdata.df$Cover_Type[i] == '3') {
    treat_col <- append(treat_col, 'yellow')
  }
  else if(newdata.df$Cover_Type[i] == '4'){
    treat_col <- append(treat_col, 'green') }
  else if(newdata.df$Cover_Type[i] == '5') {
    treat_col <- append(treat_col, 'cyan')
  }
  else if(newdata.df$Cover_Type[i] == '6'){
    treat_col <- append(treat_col, 'purple') }
  else if(newdata.df$Cover_Type[i] == '7') {
    treat_col <- append(treat_col, 'orange')
  }
}
#PCA plot PC1 vs PC2
plot(newdata.df$PC1, newdata.df$PC2, col=treat_col,
     pch=16, xlab="Principal Component 1 (29.7%)",ylab="Principal Component 2
(21.7%)",main="Score Plot")
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
legend("bottomright",inset=c(-0.2,0), legend=c("1", "2","3","4",'5','6','7'),title="Class",
       fill=c("red","blue","yellow",'green','cyan','purple','orange'))

#PCA plot PC1 vs PC3
plot(newdata.df$PC1, newdata.df$PC3, col=treat_col,
     pch=16, xlab="Principal Component 1 (29.7%)",ylab="Principal Component 3
(15.4%)",main="Score Plot")
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
legend("bottomright",inset=c(-0.2,0), legend=c("1", "2","3","4",'5','6','7'),title="Class",
       fill=c("red","blue","yellow",'green','cyan','purple','orange'))
```

```
#PCA plot PC2 vs PC3
plot(newdata.df$PC2, newdata.df$PC3, col=treat_col,
     pch=16, xlab="Principal Component 2 (21.7%)",ylab="Principal Component 3
(15.4%)",main="Score Plot")
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)
legend("bottomright",inset=c(-0.2,0), legend=c("1", "2","3","4",'5','6','7'),title="Class",
       fill=c("red","blue","yellow",'green','cyan','purple','orange'))

#k means clustering with k=1 to 10
sm= c()
s1=c()
for (i in 1:10){
  km.out=kmeans(SDATA,i, nstart=50)
  sm = c(sm,sum(km.out$tot.withinss)) #Total within-cluster sum of squares
  s1 = c(s1,km.out$totss) #the total variance in the data
  qm = sm/s1
  perfk1 = 1-qm
}
qm = sm/s1
perfk1 = 1-qm
par(mar = c(5, 4, 4, 2) + 0.1)
plot(1:10,perfk1,xlab="K",ylab="Reduction of Variance Perf(k)",main="K Means Clustering: k
Performance",type='o',pch=19,col='red')

wssplot(SDATA)


#k means clustering with k=1 to 10
sm= c()
s1=c()
for (i in 1:10){
  km.out=kmeans(SDATA,i, nstart=50)
  sm = c(sm,sum(km.out$tot.withinss)) #Total within-cluster sum of squares
  s1 = c(s1,km.out$totss) #the total variance in the data
  qm = sm/s1
  perfk1 = 1-qm
}
qm = sm/s1
perfk1 = 1-qm
par(mar = c(5, 4, 4, 2) + 0.1)
plot(1:10,sm,xlab="K",ylab="Total WSS",main="K Means Clustering: k
Performance",type='o',pch=19,col='red')
#k means with 3 clusters
```

```
km.out=kmeans(SDATA,3, nstart=50)
clusters = km.out$cluster

#Scalar product of w1,w2,w3 and the centers
centers = km.out$centers
ck = centers%*%eigenvec3

#plot of centers
#install.packages("scatterplot3d")
library("scatterplot3d")
colors <- c("red", "yellow", "green")
colors <- colors[as.numeric(ck)]
scatterplot3d(ck,color=colors,main="Cluster Centers",pch = 16)
legend("topright",legend=c("Cluster 1","Cluster 2","Cluster 3"), col=c("red", "yellow", "green"),
       pch=c(16,16,16),lty=c(1,2), ncol=1)

#pc scores with cluster assignment
newdata2 <- cbind(newdata,clusters)
colors <- c("red", "yellow", "green")
colors <- colors[as.numeric(newdata2[, "clusters"])]
scatterplot3d(newdata2,color= colors,pch = 16,main="K Means Clustering")
legend("topright",legend=c("Cluster 1","Cluster 2","Cluster 3"), col=c("Black", "Red", "green"),
pch=c(16,16,16),lty=c(1,2), ncol=1)

#sizes of 3 clusters
km.out$size

Cover_Type=ForestTypes$Cover_Type
#regular data with standardized data, clusters, and fonts
DATA <- cbind(SDATA,clusters,Cover_Type)

#cluster1
clus1 <- subset(DATA, clusters == 1)
#cluster2
clus2 <- subset(DATA, clusters == 2)
#cluster3
clus3 <- subset(DATA, clusters == 3)

library(plyr)
#get a count of each font in cluster 1
count1 <- count(clus1$Cover_Type)
de<-data.frame("4","0")
names(de)<-c("x","freq")
#get a count of each font in cluster 2
```

```
count2 <- count(clus2$Cover_Type)

count4=rbind(count2[1:3,],de,count2[4:6,])
rownames(count4) <- 1:nrow(count4)
count2=count4
#get a count of each font in cluster 3
count3 <-count(clus3$Cover_Type)

#put counts together
counts <- cbind(count1,count2$freq,count3$freq)
names(counts) <- c("Cover_Type","Cluster 1", "Cluster 2","Cluster 3")


#data frame with frequencies of Covertype in each cluster
freq_CTs=cbind(count1$freq,count2$freq,count3$freq)
colnames(freq_CTs)=c("Cluster 1","Cluster 2","Cluster 3")


#k means with 4 clusters
km.out=kmeans(SDATA,4, nstart=50)
clusters = km.out$cluster

#Scalar product of w1,w2,w3 and the centers
centers = km.out$centers
ck = centers%*%eigenvec3

#sizes of 4 clusters
km.out$size

Cover_Type=ForestTypes$Cover_Type
#regular data with standardized data, clusters, and fonts
DATA <- cbind(SDATA,clusters,Cover_Type)

#cluster1
clus1 <- subset(DATA, clusters == 1)
#cluster2
clus2 <- subset(DATA, clusters == 2)
#cluster3
clus3 <- subset(DATA, clusters == 3)
#cluster4
clus4 <- subset(DATA, clusters == 4)
library(plyr)
#get a count of each font in cluster 1
count1 <- count(clus1$Cover_Type)
```

```
missval=data.frame(x="4",freq="0")
count1=rbind(count1[1:3,],missval,count1[4:6,])
rownames(count1) <- 1:nrow(count1)
#get a count of each font in cluster 2
count2 <- count(clus2$Cover_Type)
#get a count of each font in cluster 3
count3 <-count(clus3$Cover_Type)
#get a count of each font in cluster 4
count4 <-count(clus4$Cover_Type)


#put counts together
counts <- cbind(count1,count2$freq,count3$freq,count4$freq)
names(counts) <- c("Cover_Type","Cluster 1", "Cluster 2","Cluster 3","Cluster4")

#install.packages("writexl")
#library("writexl")
write_xlsx(counts,"C:\\Users\\haroo\\OneDrive\\Documents\\R\\Azencott H.W\\counts.xlsx")


#k means with 4 clusters
km.out=kmeans(SDATA,5, nstart=50)
clusters = km.out$cluster

#Scalar product of w1,w2,w3 and the centers
centers = km.out$centers
ck = centers%*%eigenvec3

#sizes of 5 clusters
km.out$size

Cover_Type=ForestTypes$Cover_Type
#regular data with standardized data, clusters, and fonts
DATA <- cbind(SDATA,clusters,Cover_Type)

#cluster1
clus1 <- subset(DATA, clusters == 1)
#cluster2
clus2 <- subset(DATA, clusters == 2)
#cluster3
clus3 <- subset(DATA, clusters == 3)
#cluster4
clus4 <- subset(DATA, clusters == 4)
#cluster5
```

```
clus5 <- subset(DATA, clusters == 5)
#library(plyr)
#get a count of each font in cluster 1
count1 <- count(clus1$Cover_Type)

#get a count of each font in cluster 2
count2 <- count(clus2$Cover_Type)
#get a count of each font in cluster 3
count3 <-count(clus3$Cover_Type)

missval=data.frame(x="4",freq="0")
count3=rbind(count3[1:3,],missval,count3[4:6,])
rownames(count3) <- 1:nrow(count3)
#get a count of each font in cluster 4
count4 <-count(clus4$Cover_Type)
#get a count of each font in cluster 5
count5 <-count(clus5$Cover_Type)

#put counts together
counts <- cbind(count1,count2$freq,count3$freq,count4$freq,count5$freq) colnames(counts)
<- c("Cover_Type","Cluster 1", "Cluster 2","Cluster 3","Cluster 4","Cluster 5")

#install.packages("writexl")
#library("writexl")
write_xlsx(counts,"C:\\Users\\haroo\\OneDrive\\Documents\\R\\Azencott H.W\\counts1.xlsx")


library(dplyr)
DATARF <- as.data.frame(DATA[,1:12])
DATARF %>% mutate_if(is.character,as.numeric)
cty <- factor(DATA[,14])


##Q4 train test set split for random forests
DATARF=cbind(DATARF,cty)
colnames(DATARF)[13]="Cover_Type"
#seven classes
class1 <- subset(DATARF, Cover_Type== '1')
class2 <- subset(DATARF, Cover_Type == '2')
class3 <- subset(DATARF, Cover_Type == '3')
class4 <- subset(DATARF, Cover_Type == '4')
class5 <- subset(DATARF, Cover_Type == '5')
class6 <- subset(DATARF, Cover_Type == '6')
class7 <- subset(DATARF, Cover_Type == '7')
```

```
#Train Test Split for cl1
set.seed(101)
n = nrow(class1)
trainIndex = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl1 = class1[trainIndex ,]
testcl1 = class1[-trainIndex ,]

#Train Test Split for cl2
set.seed(102)
n = nrow(class2)
trainIndex1 = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl2 = class2[trainIndex1 ,]
testcl2 = class2[-trainIndex1 ,]

#Train Test Split for cl3
set.seed(103)
n = nrow(class3)
trainIndex2 = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl3 = class3[trainIndex2 ,]
testcl3 = class3[-trainIndex2 ,]

#Train Test Split for cl4
set.seed(104)
n = nrow(class4)
trainIndex = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl4 = class4[trainIndex ,]
testcl4 = class4[-trainIndex ,]

#Train Test Split for cl5
set.seed(105)
n = nrow(class5)
trainIndex1 = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl5 = class5[trainIndex1 ,]
testcl5 = class5[-trainIndex1 ,]

#Train Test Split for cl6
set.seed(106)
n = nrow(class6)
trainIndex2 = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl6 = class6[trainIndex2 ,]
testcl6 = class6[-trainIndex2 ,]

#Train Test Split for cl7
```

```
set.seed(107)
n = nrow(class7)
trainIndex2 = sample(1:n, size = round(0.8*n), replace=FALSE)
traincl7 = class7[trainIndex2 ,]
testcl7 = class7[-trainIndex2 ,]

#Full training set and test set
trainset <- rbind(traincl1,traincl2,traincl3,traincl4,traincl5,traincl6,traincl7)
testset <- rbind(testcl1,testcl2,testcl3,testcl4,testcl5,testcl6,testcl7)

##extract font label column of training and test set
test_category <- testset[,13]
train_category <- trainset[,13]

trainset1 <- trainset[,1:12]
testset1 <- testset[,1:12]

##load trees library
library(randomForest)

#Random Forest for trainset (ntree = 10)
rf_train_10 = randomForest(train_category ~., data=trainset1, ntree=10,
mtry=3,importance=TRUE)


#Random Forest for testset (ntree = 10)
rf_test_10 = predict(rf_train_10,newdata=testset1)
rf_train_T10=predict(rf_train_10,newdata=trainset1)
#Confusion matrix for Test set
conf_10 <- table(test_category,rf_test_10)
conf_10

conf_T10=table(train_category,rf_train_T10)
conf_T10
#Random Forest for trainset (ntree = 100)
rf_train_100 = randomForest(train_category ~., data=trainset1, ntree=100,
mtry=3,importance=TRUE)


#Random Forest for testset (ntree = 100)
rf_test_100 = predict(rf_train_100,newdata=testset1)
rf_train_T100=predict(rf_train_100,newdata=trainset1)
#Confusion matrix for Test set
conf_100 <- table(test_category,rf_test_100)
```

```
conf_100
#confurion matrix for training set
conf_T100=table(train_category,rf_train_T100)
conf_T100

#Random Forest for trainset (ntree = 200)
rf_train_200 = randomForest(train_category ~., data=trainset1, ntree=200,
mtry=3,importance=TRUE)


#Random Forest for testset (ntree = 200)
rf_test_200 = predict(rf_train_200,newdata=testset1)
rf_train_T200=predict(rf_train_200,newdata=trainset1)

#Confusion matrix for Test set
conf_200 <- table(test_category,rf_test_200)
conf_200

#Confusion matrix for Training set
conf_T200=table(train_category,rf_train_T200)
conf_T200

#Random Forest for trainset (ntree = 300)
rf_train_300 = randomForest(train_category ~., data=trainset1, ntree=300,
mtry=3,importance=TRUE)


#Random Forest for testset (ntree = 300)
rf_test_300 = predict(rf_train_300,newdata=testset1)
rf_train_T100=predict(rf_train_300,newdata=trainset1)
#Confusion matrix for Test set
conf_300 <- table(test_category,rf_test_300)
conf_300
#Confusion matrix for training set
conf_T300=table(train_category,rf_train_T100)
conf_T300

#Random Forest for trainset (ntree = 400)
rf_train_400 = randomForest(train_category ~., data=trainset1, ntree=400,
mtry=3,importance=TRUE)
#Random Forest for testset (ntree = 400)
rf_test_400 = predict(rf_train_400,newdata=testset1)
rf_train_T400=predict(rf_train_400,newdata=trainset1)
#Confusion matrix for Test set
```

```
conf_400 <- table(test_category,rf_test_400)
conf_400

conf_T400=table(train_category,rf_train_T400)
conf_T400


#Function to calculate accuracy
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}

##Accuracy of RF ntree = 10, test set
testperf_10 = accuracy(conf_10)
testperf_10

trainperf_10=accuracy(conf_T10)
trainperf_10

##Accuracy of RF ntree = 100, test set
testperf_100 = accuracy(conf_100)
testperf_100

trainperf_100=accuracy(conf_T100)
trainperf_100

##Accuracy of RF ntree = 200, test set
testperf_200 = accuracy(conf_200)
testperf_200

trainperf_200=accuracy(conf_T200)
trainperf_200
##Accuracy of RF ntree = 300, test set
testperf_300 = accuracy(conf_300)
testperf_300

trainperf_300=accuracy(conf_T300)
trainperf_300

##Accuracy of RF ntree = 400, test set
testperf_400 = accuracy(conf_400)
testperf_400

trainperf_400=accuracy(conf_T400)
trainperf_400
##Plot Accuracy v. ntrees
```

```
ntrees = c(10,100,200,300,400)
testperfs = c(testperf_10,testperf_100, testperf_200, testperf_300, testperf_400) trainperfs =
c(trainperf_10,trainperf_100, trainperf_200, trainperf_300, trainperf_400) plot(ntrees,
testperfs, xlim = c(0,415),ylim = c(82,87) ,type = "b", col = "blue", xlab = "N Trees", ylab =
"Accuracy")

library(ggplot2)
library(dplyr)
testperfs.df=as.data.frame(testperfs)
ntrees.df=as.data.frame(ntrees)
tbl=cbind(ntrees.df,testperfs.df)
tbl2=cbind(testperfs,trainperfs)
tbl2=as.data.frame(tbl2)
tbl2=cbind(ntrees,tbl2)

ggplot(data=tbl2,aes(x=ntrees))+
  geom_line(aes(y = trainperfs, colour = "trainperfs")) +
  geom_line(aes(y = testperfs, colour = "testperfs")) +
  scale_colour_manual("",
                      breaks = c("trainperfs", "testperfs"),
                      values = c("trainperfs"="blue", "testperfs"="red"
                              )) +
  xlim(0,415)+scale_y_continuous("Accuracy Percentage", limits = c(82,101))+
  xlab("Ntree Values")+ ylab("Performance")+
  labs(title="N trees vs performance")



tbl$feature1=c(75.23,77.55,78.01,77.78)
tbl$feature2=c(70.60,69.21,69.44,67.59)
tbl$feature3=c(80.79,80.09,80.32,81.02)
tbl$feature4=c(95.14,95.83,95.60,95.37)
tbl$feature5=c(95.83,96.06,95.83,96.53)
tbl$feature6=c(88.66,88.19,88.43,88.89)
tbl$feature7=c(97.22,96.99,96.76,97.22)

ggplot(data=tbl,aes(x=ntrees,y=feature1))+
  geom_line(color="red")+xlim(90,415)+ylim(65,99)+
  geom_line(y=tbl$feature2,color="blue")+
  geom_line(y=tbl$feature3,color="green")+
  geom_line(y=tbl$feature4,color="purple")+
  geom_line(y=tbl$feature5,color="yellow")+
  geom_line(y=tbl$feature6,color="cyan")+
  geom_line(y=tbl$feature7,color="orange")+ ylab("classes")
```

```
ggplot(data=tbl,aes(x=ntrees))+
  geom_line(aes(y = feature1, colour = "feature1")) +
  geom_line(aes(y = feature2, colour = "feature2")) +
  geom_line(aes(y = feature3, colour = "feature3")) +
  geom_line(aes(y = feature4, colour = "feature4")) +
  geom_line(aes(y = feature5, colour = "feature5")) +
  geom_line(aes(y = feature6, colour = "feature6")) +
  geom_line(aes(y = feature7, colour = "feature7")) +
  scale_colour_manual("",
                      breaks = c("feature1", "feature2","feature3","feature4",
                                 "feature5","feature6","feature7"),
                      values = c("feature1"="blue", "feature2"="red","feature3"="yellow",
                                 "feature4"="purple","feature5"="cyan","feature6"="orange",
                                 "feature7"="green")) +
  xlim(90,415)+scale_y_continuous("Accuracy Percentage", limits = c(65,99))+
  xlab("Ntree Values")+ ylab("Performances")+
  labs(title="N trees vs performance of test set for different classes ")
```