

# Forest Cover Type Project Report

*Kayode Taiwo*

*11/3/2019*

## INTRODUCTION

The forest cover type project aims to build a machine learning model that predicts what types of trees grow in an area based on the surrounding characteristics using the Forest Cover Type Dataset. This dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. All observations are cartographic variables (no remote sensing) from 30 meter x 30 meter sections of forest. This dataset includes information on tree type, shadow coverage, distance to nearby landmarks (roads, water bodies, etc), soil type, and local topography.

## Dataset

The cover type data set has 581012 rows and 55 columns. The table below shows the first 10 rows and first 10 columns of the cover type data set.

Table 1: Forest Cover Type Dataset

elevation	aspect	slope	x_hydro	y_hydro	x_road	hs_9am	hs_12pm	hs_3pm	x_fire
2596	51	3	258	0	510	221	232	148	6279
2590	56	2	212	-6	390	220	235	151	6225
2804	139	9	268	65	3180	234	238	135	6121
2785	155	18	242	118	3090	238	238	122	6211
2595	45	2	153	-1	391	220	234	150	6172
2579	132	6	300	-15	67	230	237	140	6031
2606	45	7	270	5	633	222	225	138	6256
2605	49	4	234	7	573	222	230	144	6228
2617	45	9	240	56	666	223	221	133	6244
2612	59	10	247	11	636	228	219	124	6230

The entire 55 columns of the cover type data set, their descriptions and their R data types are shown in the table below.

Table 2: Forest Cover Type Dataset Columns

name	description	type
elevation	Elevation in meters	double
aspect	Aspect in degrees azimuth	double
slope	Slope in degrees	double
x_hydro	Horizontal distance to nearest surface water features	double
y_hydro	Vertical distance to nearest surface water features	double
x_road	Horizontal distance to nearest roadway	double
hs_9am	Hillshade index at 9am, summer solstice	double
hs_12pm	Hillshade index at noon, summer solstice	double
hs_3pm	Hillshade index at 3pm, summer solstice	double
x_fire	Horizontal distance to nearest wildfire ignition points	double
wa[1:4]	Wilderness area designation (4 binary columns)	factor
st[1:40]	Soil Type designation (40 binary columns)	factor

name	description	type
y	Forest Cover Type designation (integer 1 to 7)	factor

## Inputs

The first 54 columns of the forest cover type dataset form the **features** or **input** of this analysis. Each row is a distinct **observation** or **example**. When all column values of a row are composed together in a **vector** this **vector** is known as the **feature vector** and is designated as  $\mathbf{x}$ . The complete list of **inputs** are shown in the table below.

Table 3: Inputs or Features

name	description
elevation	Elevation in meters
aspect	Aspect in degrees azimuth
slope	Slope in degrees
x_hydro	Horizontal distance to nearest surface water features
y_hydro	Vertical distance to nearest surface water features
x_road	Horizontal distance to nearest roadway
hs_9am	Hillshade index at 9am, summer solstice
hs_12pm	Hillshade index at noon, summer solstice
hs_3pm	Hillshade index at 3pm, summer solstice
x_fire	Horizontal distance to nearest wildfire ignition points
wa[1:4]	Wilderness area designation (4 binary columns)
st[1:40]	Soil Type designation (40 binary columns)

## Output

The 55<sup>th</sup> column of the forest cover type dataset form the **label** or **output** of this analysis. The **output** value can be any one of: **Spruce**, **Lodgepole**, **Panderosa**, **Cottonwood**, **Aspen**, **Douglas**, or **Krummholz** and is designated as  $y$ . The single **output** column is shown in the table below.

Table 4: Output or Label

name	description
y	Forest Cover Type designation

## Objective

The **goal** is to find a **machine learning function**  $h$  that takes a feature vector  $\mathbf{x}$  as **input** and computes a label  $y$  as **output**. Expressed as an equation:

$$y = h(\mathbf{x})$$

where,

- $y$  is the **output** cover type
- $\mathbf{x}$  is the feature vector of **input** attributes
- $h$  is the trained **machine learning algorithm** or **hypothesis function**

For this purpose we will examine and fit several machine learning algorithms including *k-Nearest Neighbors*, *Random Forest*, *Linear Discriminant Analysis*, *Penalized Multinomial Regression* and *Support Vector Machines*

with *Linear Kernel*. We will also attempt to reduce the dimension of the dataset using *Principal Component Analysis* as a *Matrix Factorization* method.

## Steps

We start off the analysis by **pre-processing** the entire forest cover type dataset to mutate some columns from their intrinsic data type to a data type that is more appropriate for our analysis. Next we **partition** our dataset into a *train set* and a *test set* using a random 60:40 split respectively. Next we explore the possibility of applying the principal component analysis (PCA) method for **dimension reduction**. Then we step right into the meat of the analysis and try out several **machine learning algorithms** by training on the *train set* and evaluating on the *test set*. Finally we build an **ensemble** model using all of the machine learning models already built and examine its accuracy vis-a-vis each individual model.

## Problems Encountered

Extended training run times due to large dataset. The entire forest cover type dataset has 581012 rows. Even at 60:40 split between **train set** and **test set**, the **train set** still has at least 348607 rows. This proves to be a challenge for the built in model evaluation and parameter tuning of the **caret** *train()* function, which uses a 25 round bootstrap resampling for model parameter optimization by default. Many of the models could not be trained successfully because either the *train()* function ran for days on end without completion or the training could not run at all due to memory exhaustion. So to solve this issue we configured the *train()* function to use k-fold cross validation using code similar to the following.

```
library(caret)
control <- trainControl(method = "cv", number = 2)
train <- train(y ~ ., method = "my_method", data = dat$train, trControl = control)
```

We found that k-fold cross validation with  $k = 2$  or  $k = 3$  reduced the training run time of each model to at most 24 hours or a little over that. This was more bearable for this analysis

## METHODS

### Preprocessing

Columns wa1 to wa4 of the forest cover type dataset are used to describe the wilderness area designation of each dataset row. Similarly, columns st1 to st40 are used to describe the soil type designation. These columns (wa1 to wa4 and st1 to st40) are encoded as 0 or 1 in the original dataset, where 0 means **absence** and 1 means **presence**. A pre-processing was run to mutate these columns to a 2 level factor with levels (absence, presence).

A similar preprocessing was also run on column y of the forest cover type dataset. This column describes the cover type of each row. It was encoded as an integer from 1 through 7, where 1 means spruce, 2 means lodgepole, 3 means panderosa, 4 means cottonwood, 5 means aspen, 6 means douglas, and 7 means krummholz. This column (y) was mutated to a 7 level factor with levels (spruce, lodgepole, panderosa, cottonwood, aspen, douglas, and krummholz).

### Partitioning

The forest cover type dataset was partitioned into two sets, a **train** set for training our models, and a **test** set for evaluating them. 60% of the entire dataset (348607 rows) was randomly allocated for the **train** set while the remaining 40% of the entire dataset (232404 rows) was randomly allocated for the **test** set, thus splitting the entire dataset in a 60:40 ratio. We had sought to do a more traditional 80:20 split between **train** and **test** but because the size of our entire dataset is large and our models encountered memory and time (could not run to conclusion even after as long as seven days of non-stop execution) problems during training we decided to go with a 60:40 split. The **caret** *createDataPartition()* function was used to do the partitioning using code similar to the following.

```
set.seed(2, sample.kind="Rounding")
test_index <- createDataPartition(y = covtype$y, times = 1, p = 0.4, list = FALSE)
train <- covtype[-test_index,]
test <- covtype[test_index,]
```

## Dimension reduction

Because we had 54 columns on our input we decided to employ a matrix factorization to determine the principal components of our dataset. Doing this would let us review each columns contribution to the total variability of our dataset and enable us to perform our analysis with just those columns that are responsible for majority of the variability in our dataset. So a principal component analysis was performed using the following code.

```
x <- dat$train[,1:54] %>% as.matrix()
pca <- prcomp(x)
summary(pca)
```

A summary of the principal component analysis results for the first 5 principal components is shown in the table below. It shows clearly that 0.99917 of total variability is captured by the first 5 principal components.

Table 5: Importance of components

col	PC1	PC2	PC3	PC4	PC5
Standard deviation	1693.503	1152.1665	276.76365	192.49089	115.17437
Proportion of Variance	0.663	0.3069	0.01771	0.00857	0.00307
Cumulative Proportion	0.663	0.9698	0.98754	0.99610	0.99917

So we ran a k-NN with the first 2 principal components (PC1 and PC2) which together capture 0.9698 of total variability using code similar to that shown below.

```
fit <- knn3(pca$x[,1:2], dat$train$y, k = 5)
x_test <- dat$test[,1:54] %>% as.matrix()
pca_test <- prcomp(x_test)
y_hat <- predict(fit, pca_test$x[,1:2], type = "class")
cm <- confusionMatrix(data = y_hat, reference = dat$test$y)
cm$overall["Accuracy"]
```

```
Accuracy
0.4992
```

We examined the accuracy returned through the confusion matrix and found it to be **0.4992**. This is simply not good enough. So Principal Component Analysis was **not used** in this analysis. Tried and tested, but **not used**.

## Modeling

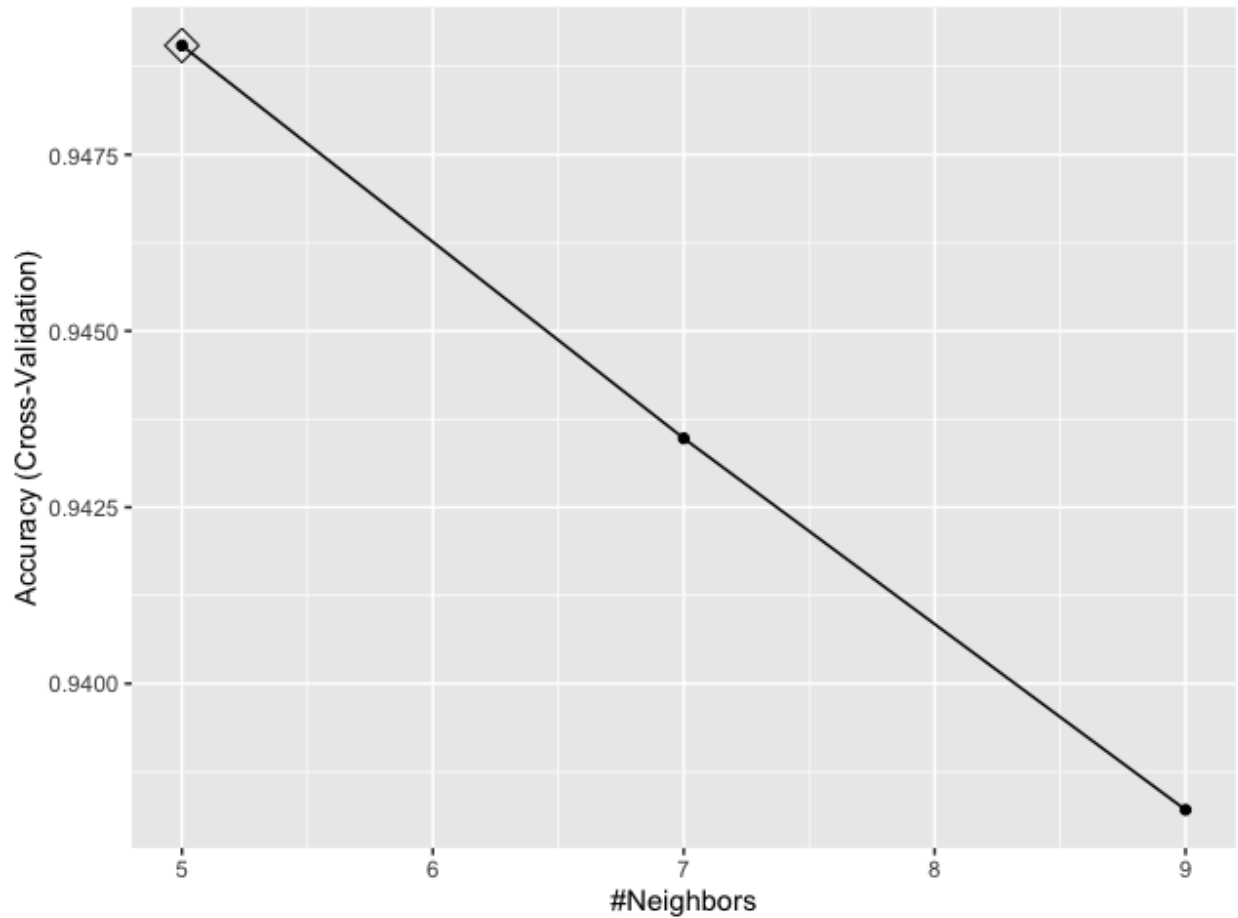
Five different machine learning models were built using five different machine learning algorithms: *k-Nearest Neighbors*, *Random Forest*, *Linear Discriminant Analysis*, *Penalized Multinomial Regression* and *Support Vector Machines with Linear Kernel*. The following sections discuss use of each algorithm in this analysis and how we selected required model parameters for each algorithm.

### k-Nearest Neighbors

A k-Nearest Neighbors model was trained with the **caret** *train()* function. The *train()* function was configured to use k-fold cross-validation for parameter tuning with  $k = 2$  using code similar to the following.

```
library(caret)
control <- trainControl(method = "cv", number = 2)
train_knn <- train(y ~ ., method = "knn", data = dat$train, trControl = control)
```

The plot function can be used to examine the relationship between the estimates of performance and the tuning parameters.



Accuracy was used to select the optimal model using the largest value. The final value used for the model was  $k = 5$ . This can be seen in the following table.

Table 6: Resampling results across tuning parameters

k	Accuracy	Kappa
5	0.9490477	0.9180877
7	0.9434776	0.9090698
9	0.9382109	0.9005325

Here is the accuracy of the best model when applied to the test set, which we have not used at all yet because the cross validation was done on the training set:

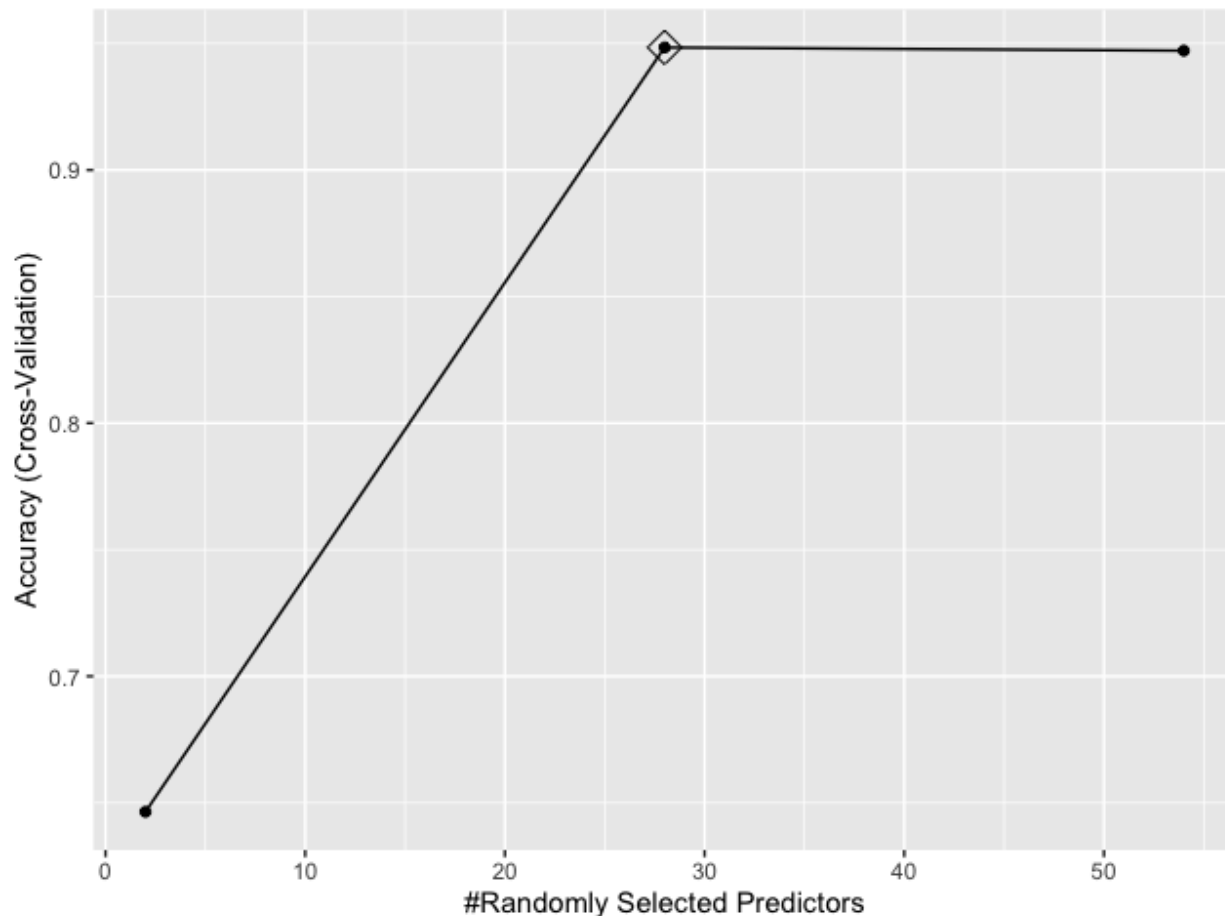
```
# cm_knn$overall["Accuracy"]
```

## Random Forest

A Random Forest model was trained with the **caret** `train()` function. The `train()` function was configured to use k-fold cross-validation for parameter tuning with  $k = 2$  using code similar to the following.

```
library(caret)
control <- trainControl(method = "cv", number = 2)
train_rf <- train(y ~ ., method = "rf", data = dat$train, trControl = control)
```

The plot function can be used to examine the relationship between the estimates of performance and the tuning parameters.



Accuracy was used to select the optimal model using the largest value. The final value used for the model was  $mtry = 28$ . This can be seen in the resampling results as shown in the following table.

Table 7: Resampling results across tuning parameters

mtry	Accuracy	Kappa
2	0.6464308	0.3774060
28	0.9484144	0.9168181
54	0.9471494	0.9148296

Here is the accuracy of the best model when applied to the test set, which we have not used at all yet because the cross validation was done on the training set: 0.9649. Class specific summary statistics are shown in the table below.

Table 8: Statistics by Class: Random Forest

Statistic	Aspen	Cottonwood	Douglas	Krummholz	Lodgepole	Panderosa	Spruce
Sensitivity	0.83228	0.864422	0.92515	0.96014	0.9775	0.96735	0.9586
Specificity	0.99897	0.999658	0.99833	0.99888	0.9651	0.99700	0.9837
Pos Pred Value	0.93053	0.923226	0.94473	0.96924	0.9638	0.95486	0.9712
Neg Pred Value	0.99722	0.999356	0.99770	0.99854	0.9783	0.99786	0.9764
Prevalence	0.01634	0.004729	0.02989	0.03530	0.4876	0.06154	0.3646
Detection Rate	0.01360	0.004088	0.02765	0.03389	0.4766	0.05953	0.3495
Detection Prevalence	0.01462	0.004428	0.02927	0.03497	0.4945	0.06234	0.3599
Balanced Accuracy	0.91562	0.932040	0.96174	0.97951	0.9713	0.98217	0.9711

### Linear Discriminant Analysis

A Linear Discriminant model was trained with the **caret** *train()* function. The *train()* function was configured to use k-fold cross-validation for parameter tuning with  $k = 2$  using code similar to the following.

```
library(caret)
control <- trainControl(method = "cv", number = 2)
train_lda <- train(y ~ ., method = "lda", data = dat$train, trControl = control)
```

There are no tuning parameters for this model. The resampling results are shown in the following table.

Table 9: Resampling results

Accuracy	Kappa
0.6791449	0.4958998

Here is the accuracy of the best model when applied to the test set, which we have not used at all yet because the cross validation was done on the training set: 0.6788. Class specific summary statistics are shown in the table below.

Table 10: Statistics by Class: Linear Discriminant Analysis

Statistic	Aspen	Cottonwood	Douglas	Krummholz	Lodgepole	Panderosa	Spruce
Sensitivity	0.231174	0.571429	0.50223	0.80400	0.7599	0.53706	0.6180
Specificity	0.990254	0.992149	0.97108	0.95717	0.7672	0.97930	0.8493
Pos Pred Value	0.282679	0.256956	0.34859	0.40719	0.7565	0.62980	0.7018
Neg Pred Value	0.987266	0.997952	0.98445	0.99256	0.7705	0.96993	0.7949
Prevalence	0.016342	0.004729	0.02989	0.03530	0.4876	0.06154	0.3646
Detection Rate	0.003778	0.002702	0.01501	0.02838	0.3705	0.03305	0.2253
Detection Prevalence	0.013364	0.010516	0.04307	0.06970	0.4898	0.05248	0.3211
Balanced Accuracy	0.610714	0.781789	0.73666	0.88058	0.7635	0.75818	0.7337

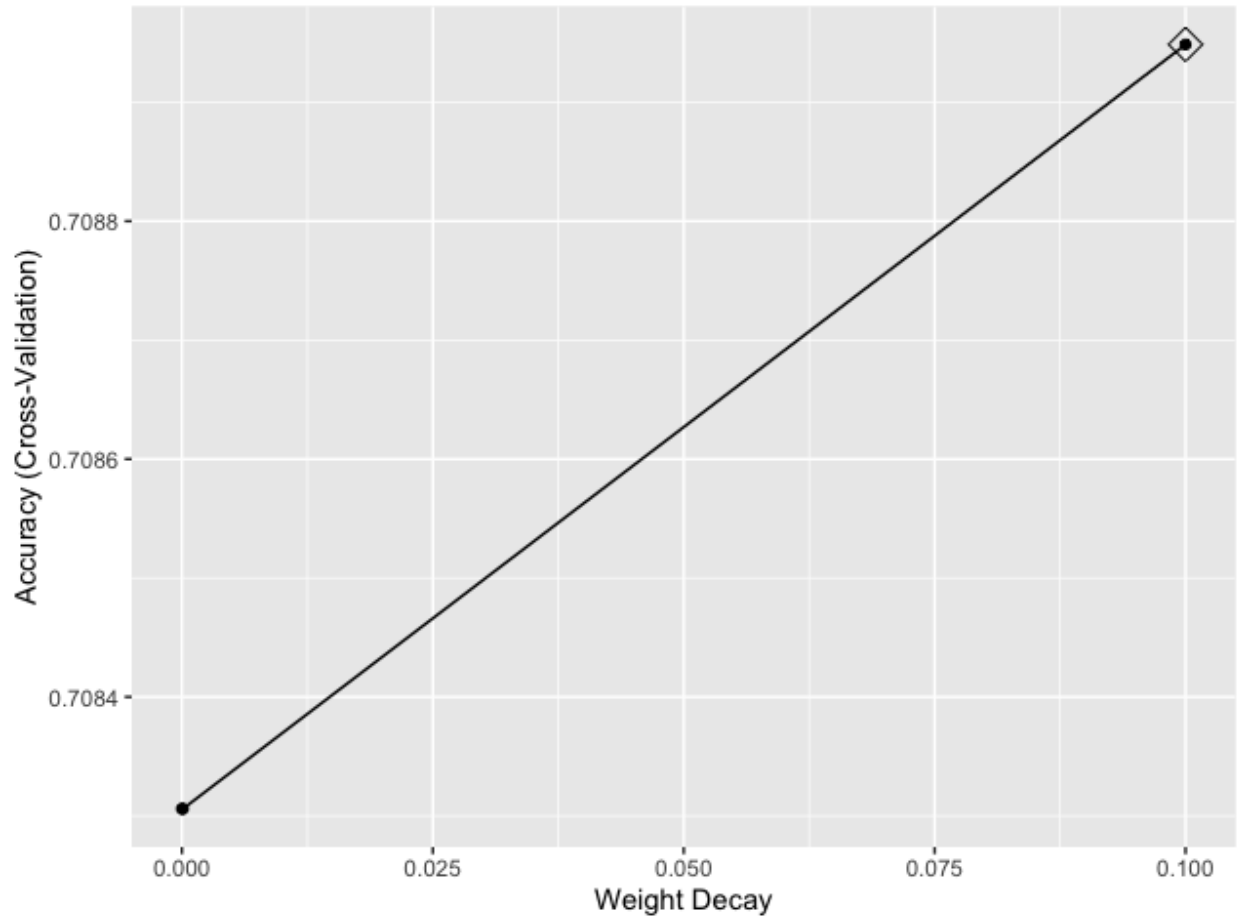
### Penalized Multinomial Regression

A Penalized Multinomial Regression model was trained with the **caret** *train()* function. The *train()* function was configured to use k-fold cross-validation for parameter tuning with  $k = 2$  using code similar to the following.

```
library(caret)
control <- trainControl(method = "cv", number = 2)
```

```
train_mn <- train(y ~ ., method = "multinom", data = dat$train, trControl = control)
```

The plot function can be used to examine the relationship between the estimates of performance and the tuning parameters.



Accuracy was used to select the optimal model using the largest value. The final value used for the model was  $\text{decay} = 0.1$ . This can be seen in the resampling results as shown in the following table.

Table 11: Resampling results across tuning parameters

decay	Accuracy	Kappa
0e+00	0.7083060	0.5240412
1e-04	0.7083060	0.5240419
1e-01	0.7089485	0.5251393

Here is the accuracy of the best model when applied to the test set, which we have not used at all yet because the cross validation was done on the training set: 0.6982. Class specific summary statistics are shown in the table below.

Table 12: Statistics by Class: Penalized Multinomial Regression

Statistic	Aspen	Cottonwood	Douglas	Krummholz	Lodgepole	Panderosa	Spruce
Sensitivity	0.0007899	0.0536852	0.191737	0.51036	0.8207	0.76793	0.6219



Statistic	Aspen	Cottonwood	Douglas	Krummholz	Lodgepole	Panderosa	Spruce
Specificity	0.9996000	0.9989408	0.990491	0.98944	0.6878	0.96999	0.8540
Pos Pred Value	0.0300000	0.1940789	0.383199	0.63885	0.7144	0.62660	0.7096
Neg Pred Value	0.9837000	0.9955192	0.975473	0.98221	0.8012	0.98455	0.7974
Prevalence	0.0163400	0.0047288	0.029892	0.03530	0.4876	0.06154	0.3646
Detection Rate	0.0000129	0.0002539	0.005731	0.01802	0.4002	0.04726	0.2267
Detection Prevalence	0.0004303	0.0013081	0.014957	0.02820	0.5602	0.07542	0.3195
Balanced Accuracy	0.5002000	0.5263130	0.591114	0.74990	0.7542	0.86896	0.7379

## Support Vector Machines with Linear Kernel

### Ensemble

Finally we build an ensemble model from the predictions of the models we have already built. When the ensemble is presented with an observation or example to classify, it examines the predictions of each of its component models for the same example and then predicts the class with the majority vote. When we construct the ensemble from all the models we have previously built, namely, *k-Nearest Neighbors*, *Random Forest*, *Linear Discriminant Analysis*, *Penalized Multinomial Regression* and *Support Vector Machines with Linear Kernel*, we obtain an overall ensemble accuracy of **0.7690087**, which is lower than the individual accuracies obtained by some of its constituent models. This is shown in the table below.

Table 13: Ensemble Accuracy vs. Component Model Accuracies:  
All

kNN	randomForest	LDA	Multinom	SVM	Ensemble
0.9686	0.9649	0.6788	0.6982	0.8141	0.769

But when we construct the ensemble from the top three individual models, *k-Nearest Neighbors*, *Random Forest* and *Support Vector Machines with Linear Kernel*, we obtain an overall ensemble accuracy of **0.9990087**, which is higher than the individual accuracies by using each model independently. This is shown in the table below.

Table 14: Ensemble Accuracy vs. Component Model Accuracies:  
Top3

kNN	randomForest	SVM	Ensemble
0.9686	0.9649	0.8141	0.999

## RESULTS

In this analysis we have trained several machine learning models to predict forest cover type from a combination of several measurable attributes. The models we have trained include: *k-Nearest Neighbors*, *Random Forest*, *Linear Discriminant Analysis*, *Penalized Multinomial Regression* and *Support Vector Machines with Linear Kernel*. We have also constructed Ensemble models by combining the predictions of each individual model to form a majority vote predictor. The accuracies obtained from this analysis are shown in the table below.

Table 15: Forest Cover Type Analysis Results

kNN	randomForest	LDA	Multinom	SVM	Ensemble
0.9686	0.9649	0.6788	0.6982	0.8141	0.769

## CONCLUSION

From the accuracies presented in the results section, we conclude that if we were to pick just one model to predict forest cover type with it should be the *k-Nearest Neighbors* model. *k-Nearest Neighbors* takes the longest time to train and also takes the longest time to predict. It's delivered accuracies make up for this however. The next best performing model for predicting forest cover type is the *Random Forest* model, which also takes a long time to train, albeit not as long as *k-Nearest Neighbors* model. Predicting from the *Random Forest* model is somewhat as time consuming as it is with *k-Nearest Neighbors*.

## Limitations

Infrastructure and time were a huge limitation during this analysis. Several of the machine learning algorithms trained on this project took days on end to train due to large dataset size. I think this will have been different if I had access to more infrastructure - faster hardware, more processing power, memory, and disk. This lack of infrastructure led me to reduce the time spent tuning model parameters.

## Future work

If I had to do this analysis again, I would certainly start with more infrastructure - faster hardware, more processing power, memory, and disk. Cloud processing is a strong contender for getting all three. Next I will spend more time tuning each individual model's parameters. For this project, it was sufficient to accept the default tuning grid of parameters defaulted by the `caret train()` function. Future work should build up on this by providing non-default tuning grid of parameters for training each model. Lastly I would increase the  $k$  in my k-fold cross-validation from  $k = 2$  used in this analysis, to perhaps  $k = 10$ .

## Finally

What do you mean finally? There is no finally :-). **That's all folks.** Thanks and Bye for now! Lolz. Just kidding. Ha ah. I have had loads of fun doing this analysis. Faced lots of frustrations and waiting for training to complete while tapping my fingers impatiently on my keyboard. I however enjoyed myself quite thoroughly and learnt quite a lot while pulling my hair off from their roots. Wish I I had more time and infrastructure to throw at this. Plus - I am not completely bald just yet...