Homework 6

CS 498, Spring 2018, Xiaoming Ji

Problem 1

Linear regression with various regularizers The UCI Machine Learning dataset repository hosts a dataset giving features of music, and the latitude and longitude from which that music originates here. Investigate methods to predict latitude and longitude from these features, as below. There are actually two versions of this dataset. We will use the one with more independent variables. We will ignore outliers. We also regard latitude and longitude as entirely independent.

```
library(readr)

ALL_DF = read_csv("./Geographical Original of Music/default_plus_chromatic_features_1059_tracks.txt",
                  col_names = FALSE)
COL_NUM = dim(ALL_DF)[2]
colnames(ALL_DF)[1:(COL_NUM - 2)] = sapply(1:(COL_NUM - 2), FUN =
                                           function(x) {paste("F", x, sep = "")})
colnames(ALL_DF)[COL_NUM - 1] = "Lat"
colnames(ALL_DF)[COL_NUM] = "Long"

#Change origin in order to eliminate negative value
ALL_DF$Lat = 180 + ALL_DF$Lat
ALL_DF$Long =  180 + ALL_DF$Long

Lat_DF = ALL_DF[, -COL_NUM]
Long_DF = ALL_DF[, -(COL_NUM - 1)]
```

1.1

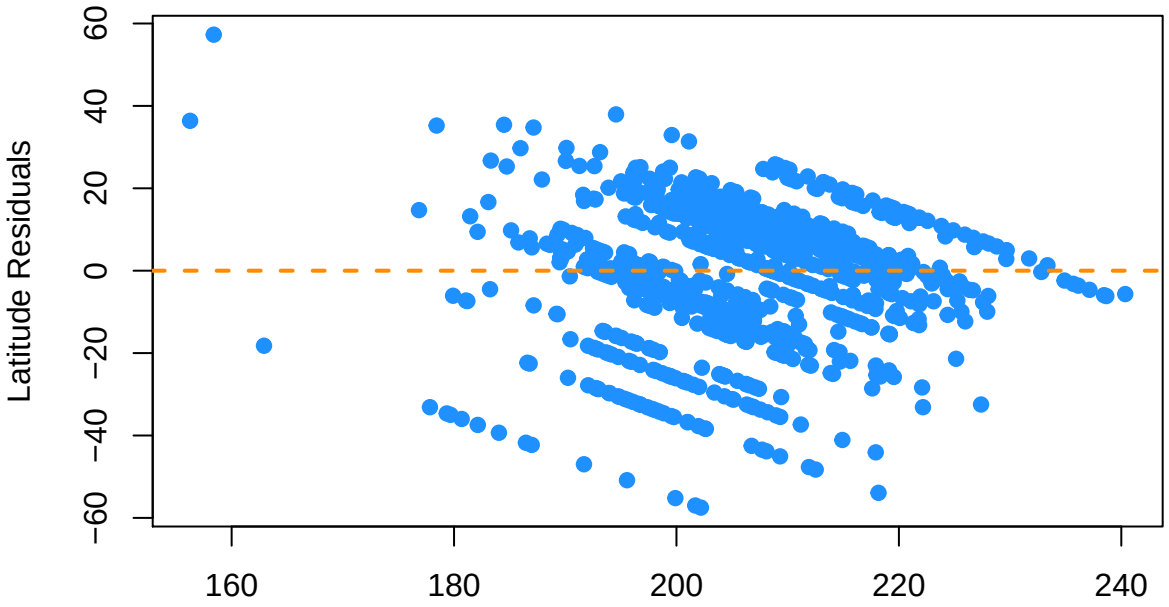
First, build a straightforward linear regression of latitude (resp. longitude) against features. What is the R-squared? Plot a graph evaluating each regression.

```
lat_model = lm(Lat ~ ., data = Lat_DF)
long_model = lm(Long ~ ., data = Long_DF)
```

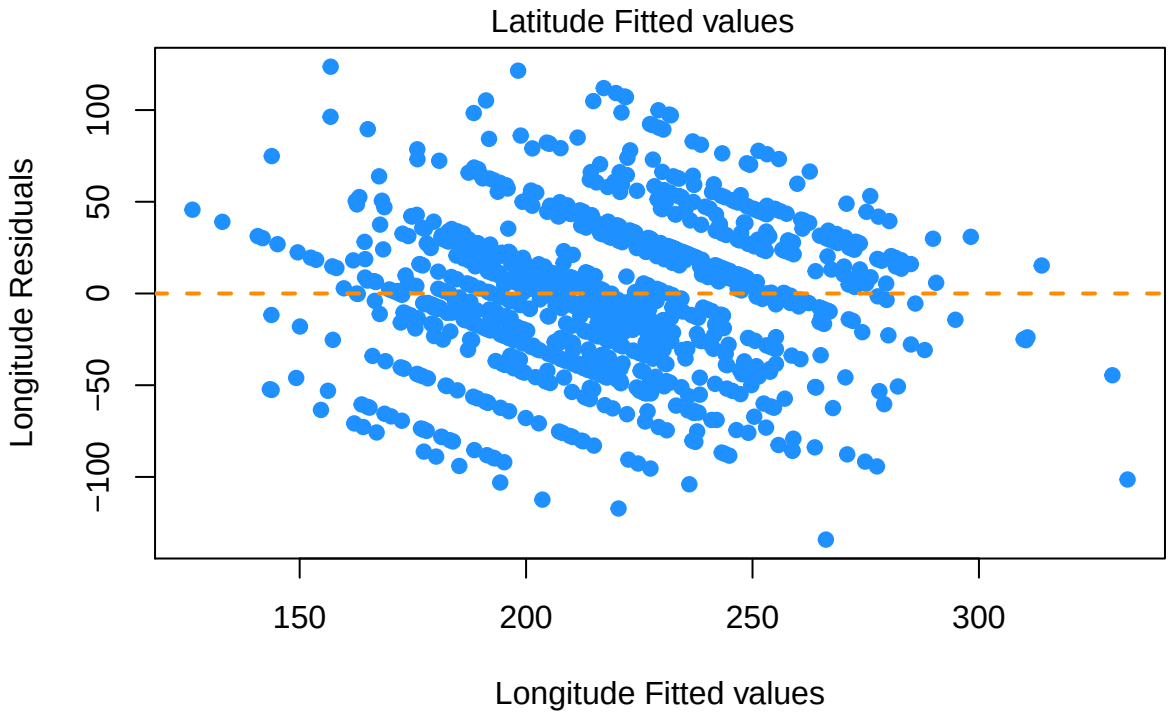
We get R-squared for each model as,

- Latitude: 0.2928092
- Longitude: 0.3645767

Plots of Residual vs Fitted Values as,



- Latitude:



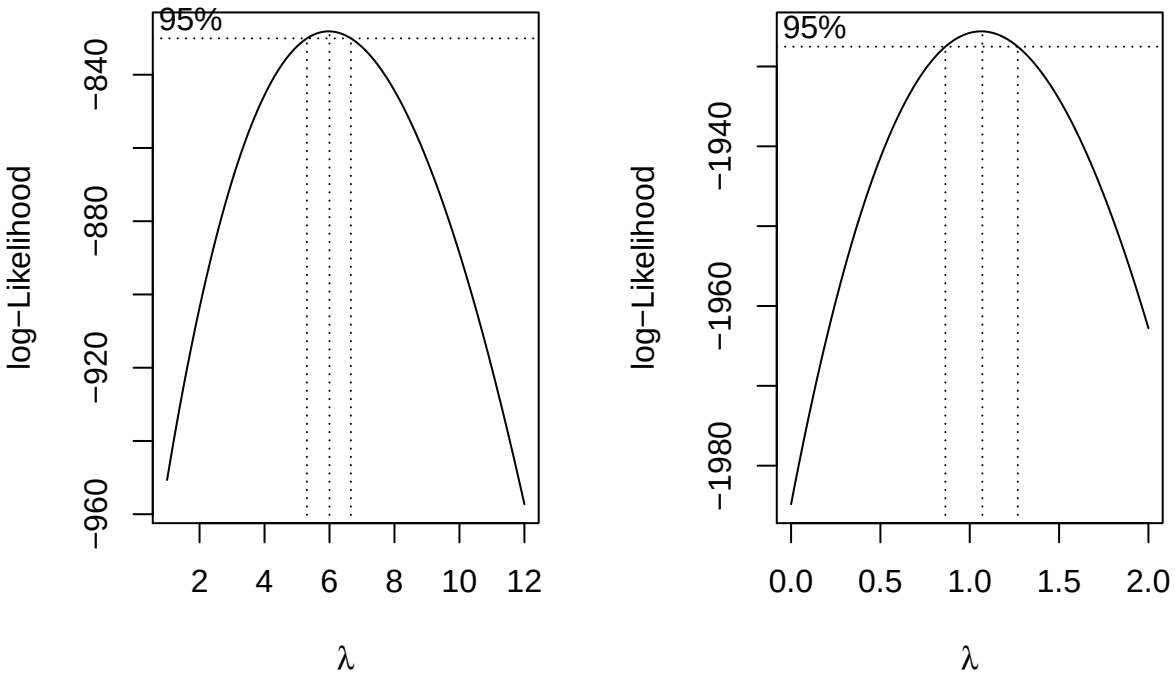
- Longitude:

1.2

Does a Box-Cox transformation improve the regressions?

We make Box-Cox plot to determine λ.

```
library(MASS)
par(mfrow=c(1, 2))
boxcox(lat_model, lambda = seq(1, 12, 0.1), plotit = TRUE)
boxcox(long_model, lambda = seq(0, 2, 0.1), plotit = TRUE)
```



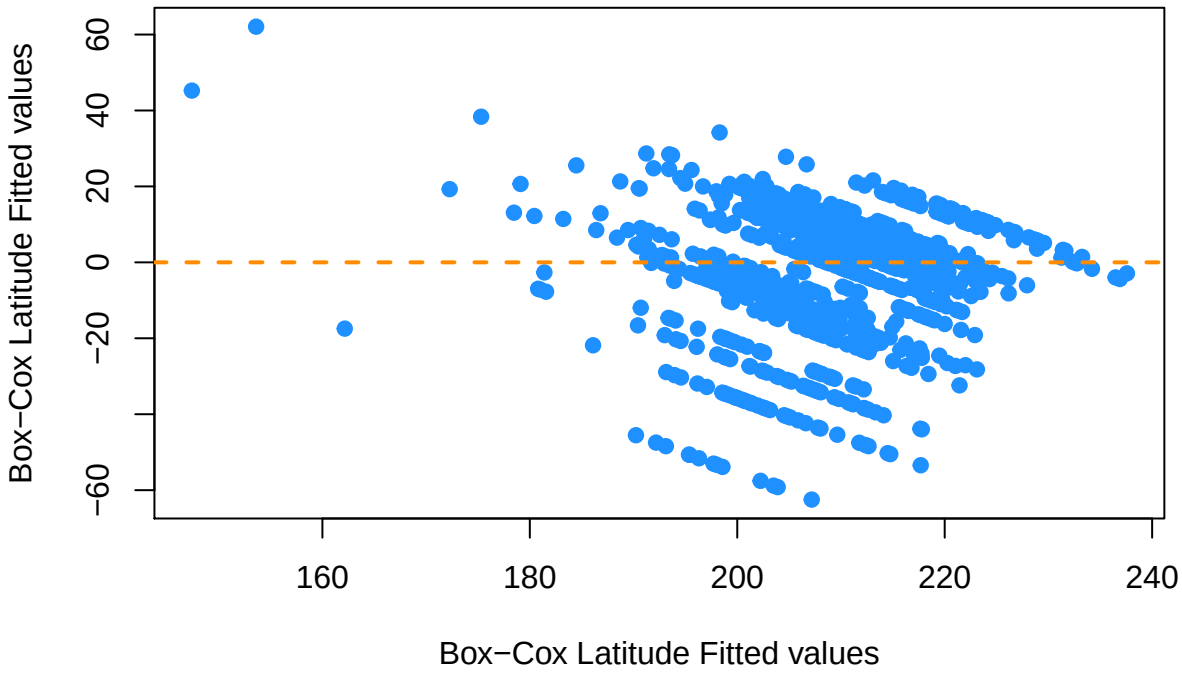
From the above plots, we see the best λ,

- Latitude: 6
- Longitude: 1, which means no need to do transformation.

We evaluate whether Box-Cox transformation for $\lambda = 6$ can give us better Latitude regression model.

We firstly make Residual vs Fitted Values plot.

```
#We make another dataframe and do the transform on the data.
#This is to benefit the later regularization comparison
BoxCox_Lat_DF = Lat_DF
BoxCox_Lat_DF$Lat = (BoxCox_Lat_DF$Lat ^ 6 - 1) / 6
boxcox_lat_model = lm(Lat ~ ., data = BoxCox_Lat_DF)

v = (6 * boxcox_lat_model$fitted.values + 1) ^ (1/6)
r = Lat_DF$Lat - v

plot_residule(v, r, "Box-Cox Latitude Fitted values",
              "Box-Cox Latitude Fitted values")
```



This plot doesn't seem to have much different with the regular one. We then check the R Squared, R Adjusted Squared and RMSE value of both models.

```
regular_rmse = sqrt(mean(resid(lat_model) ^ 2))
boxcox_rmse = sqrt(mean(r ^ 2))
```

Model	R.Squared	R.Adjusted.Squared	RMSE
Regular	0.2928092	0.2411685	15.516061
Box-Cox	0.3301234	0.2812074	16.0806264

Above info shows Box-Cox Latitude regression model has better R Squared (12.7% gain) and R Adjusted Squared (16.6% gain) value than the regular one. Although it has a bigger RMSE, the incremental is relatively small (3.6% bigger). Thus, we believe the Box-Cox model is better.

To conclude:

- Box-Cox transformation gives better Latitude regression model.
- We will not use any transformation for Longitude model.

1.3

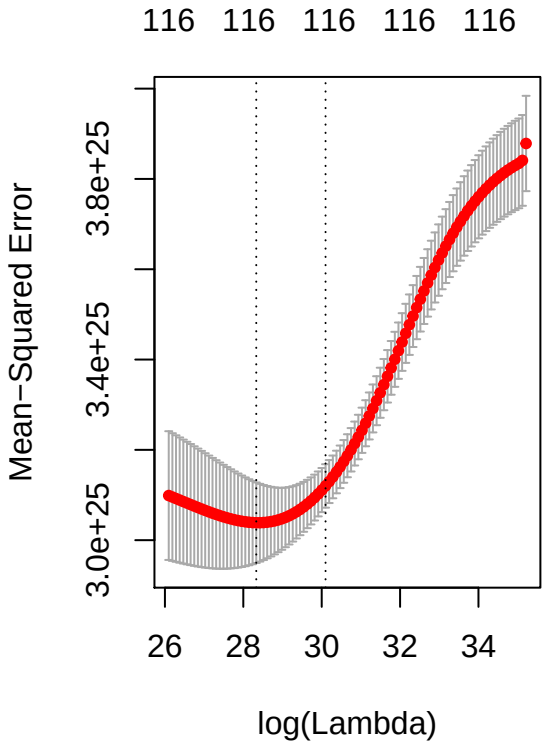
Use glmnet to produce:

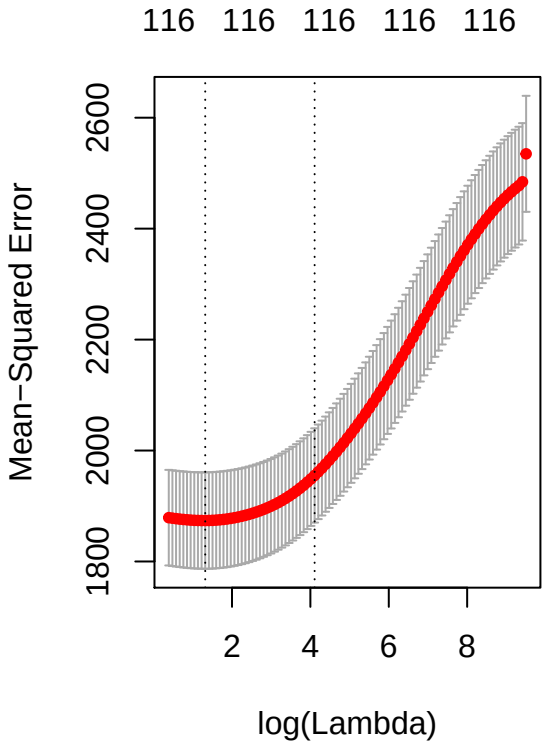
1.3.1

A regression regularized by L2 (equivalently, a ridge regression). We will estimate the regularization coefficient that produces the minimum error. Is the regularized regression better than the unregularized regression?

```
library(glmnet)

#Compare the regularized model with the unregularized one
```

```
compare_model = function(df, alpha, family = "gaussian") {
  col_num = dim(df)[2]
  x = as.matrix(df[, -col_num])
  y = as.matrix(df[,col_num])

  reg_model = cv.glmnet(x, y, alpha = alpha, family = family)
  reg_lambda = reg_model$lambda.min

  com_model = cv.glmnet(x, y, alpha = alpha, lambda = c(reg_lambda, 0),
                        family = family)

  return (list(reg_model = reg_model,
               best_lambda = com_model$lambda.min,
               cvm = com_model$cvm))
}

lat_result = compare_model(BoxCox_Lat_DF, 0)
long_result = compare_model(Long_DF, 0)
```





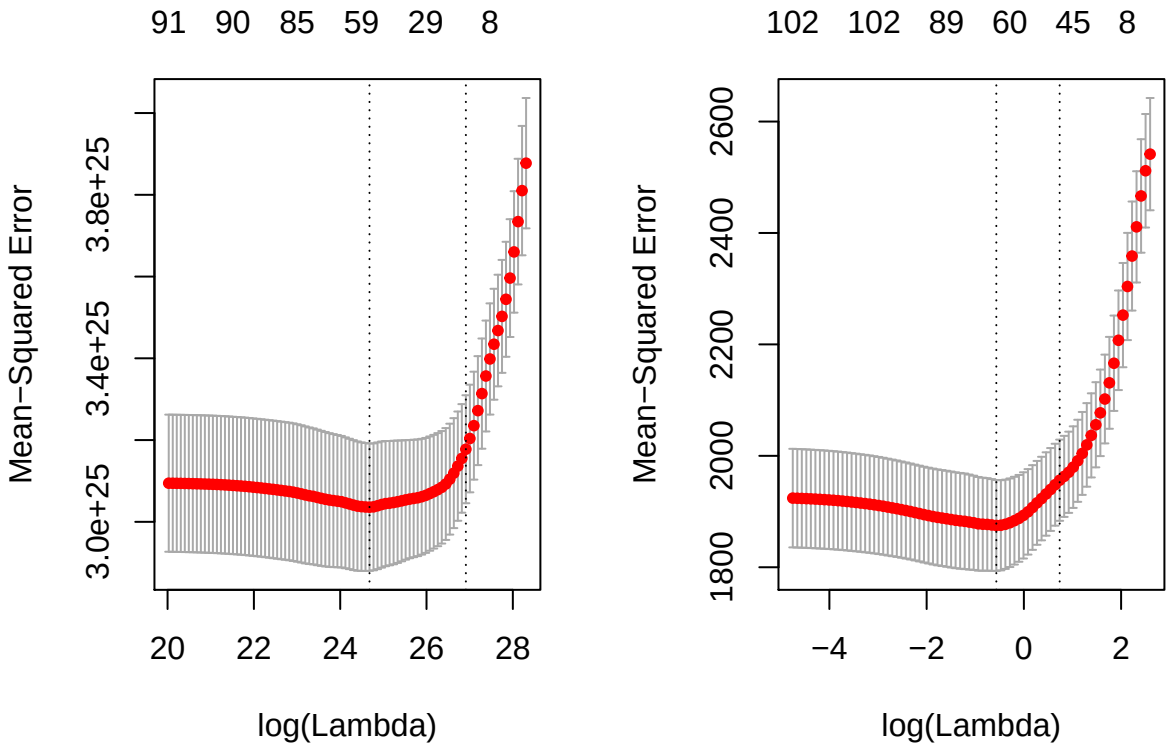
Model	Error (Regularized)	Error (UnRegularized)	Best λ
Latitude	3.0215436×10^{25}	3.1103474×10^{25}	2.0124594×10^{12}
Longitude	1908.9727623	1958.8159403	3.7332596

- From the comparison above, we see both Latitude and Longitude regularized L2 models have smaller cross-validation error (MSE) than the unregulized ones. Thus are better.

1.3.2

A regression regularized by L1 (equivalently, a lasso regression). We will estimate the regularization coefficient that produces the minimum error. How many variables are used by this regression? Is the regularized regression better than the unregularized regression?

```
lat_result = compare_model(BoxCox_Lat_DF, 1)
long_result = compare_model(Long_DF, 1)

lat_para_num = lat_result$reg_model$glmnet.fit$df[which.min(lat_result$reg_model$cvm)]
long_para_num = long_result$reg_model$glmnet.fit$df[which.min(long_result$reg_model$cvm)]
```



- See table below to find the number of variables used by each regression.

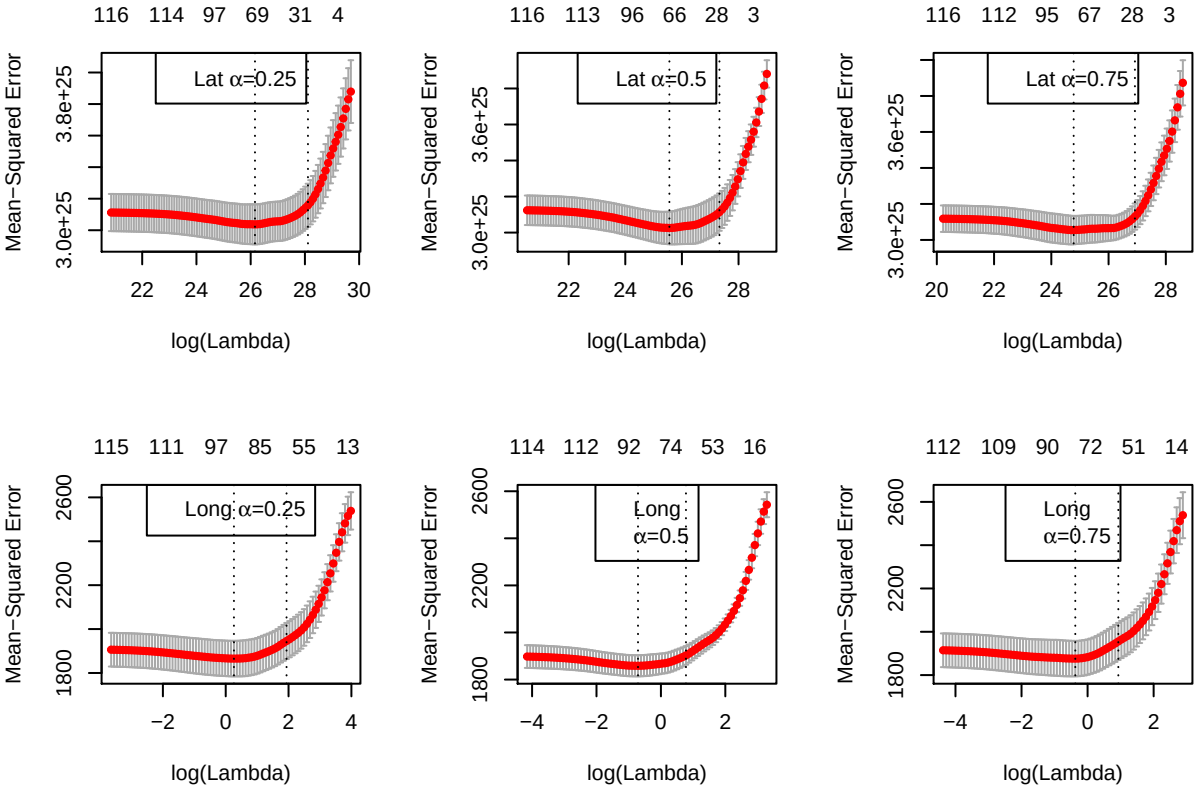
Model	Error (Regularized)	Error (UnRegularized)	Best λ	# of Variables
Latitude	2.9985332×10^{25}	3.0477783×10^{25}	5.222381×10^{10}	59
Longitude	1875.943407	1915.1809905	0.5674223	67

- From the comparison above, we see both Latitude and Longitude regularized L1 models have smaller cross-validation error (MSE) than the unregulized ones. Thus are better.

1.3.3

A regression regularized by elastic net (equivalently, a regression regularized by a convex combination of L1 and L2). Try three values of alpha, the weight setting how big L1 and L2 are. We will estimate the regularization coefficient that produces the minimum error. How many variables are used by this regression? Is the regularized regression better than the unregularized regression?

```
lat_result_25 = compare_model(BoxCox_Lat_DF, 0.25)
lat_para_num_25 = lat_result_25$reg_model$glmnet.fit$df[which.min(lat_result_25$reg_model$cvm)]
long_result_25 = compare_model(Long_DF, 0.25)
long_para_num_25 = long_result_25$reg_model$glmnet.fit$df[which.min(long_result_25$reg_model$cvm)]

lat_result_50 = compare_model(BoxCox_Lat_DF, 0.5)
lat_para_num_50 = lat_result_50$reg_model$glmnet.fit$df[which.min(lat_result_50$reg_model$cvm)]
long_result_50 = compare_model(Long_DF, 0.5)
long_para_num_50 = long_result_50$reg_model$glmnet.fit$df[which.min(long_result_50$reg_model$cvm)]

lat_result_75 = compare_model(BoxCox_Lat_DF, 0.75)
lat_para_num_75 = lat_result_75$reg_model$glmnet.fit$df[which.min(lat_result_75$reg_model$cvm)]
long_result_75 = compare_model(Long_DF, 0.75)
long_para_num_75 = long_result_75$reg_model$glmnet.fit$df[which.min(long_result_75$reg_model$cvm)]
```



- See table below to find the number of variables used by each regression.

Model	Error (Regularized)	Error (UnRegularized)	Best λ	# of Variables
Latitude (α=0.25)	3.0185066×10^{25}	3.0788538×10^{25}	2.2926227×10^{11}	72
Latitude (α=0.5)	3.0514176×10^{25}	3.0995763×10^{25}	1.2580753×10^{11}	68
Latitude (α=0.75)	3.0097184×10^{25}	3.0583671×10^{25}	5.7809499×10^{10}	81
Longitude (α=0.25)	1858.5661108	1895.6811462	1.2987995	87

Model	Error (Regularized)	Error (UnRegularized)	Best λ	# of Variables
Longitude (α=0.5)	1872.0007466	1913.7241039	0.4912471	88
Longitude (α=0.75)	1895.9246209	1923.7608492	0.689352	76

- From the comparison above, we see all Latitude and Longitude regularized L1 models have smaller cross-validation error (MSE) than the unregulized ones. Thus are better.

Problem 2

Logistic regression. The UCI Machine Learning dataset repository hosts a dataset giving whether a Taiwanese credit card user defaults against a variety of features here. Use logistic regression to predict whether the user defaults. We will ignore outliers, but try the various regularization schemes we have discussed.

```
set.seed(20180318)
CreditDF = read_csv("credit.csv", col_names = FALSE)
CreditDF = CreditDF[, -1] #Remove the ID column
RecordNum = dim(CreditDF)[1]
ColumnNum = dim(CreditDF)[2]

#Make 80%-20% training-test data split
TrainIndex = sample(1:RecordNum, RecordNum * 0.8)
TestIndex = (1:RecordNum)[-TrainIndex]
TrainCreditDF = CreditDF[TrainIndex,]
TestCreditDF = CreditDF[TestIndex,]

TrainCreditX = as.matrix(TrainCreditDF[,1:(dim(TrainCreditDF)[2] - 1)])
TrainCreditY = as.matrix(TrainCreditDF[,dim(TrainCreditDF)[2]])

best_alpha_bi_model = function(TrainDF, TestDF, alphas) {
  col_num = dim(TrainDF)[2]
  train_x = as.matrix(TrainDF[, -col_num])
  train_y = as.matrix(TrainDF[,col_num])
  test_x = as.matrix(TestDF[,-col_num])
  test_y = as.matrix(TestDF[,col_num])
  num_test = length(test_y)

  models = list()
  accuracies = rep(0, length(alphas))

  for (i in 1:length(alphas)) {
    models[[i]] = cv.glmnet(train_x, train_y, family="binomial", alpha = alphas[i])
    accuracies[i] = sum((predict(models[[i]], newx = test_x, s = "lambda.min",
                                 type = "response") > 0.5) == test_y) / num_test

    print(accuracies[i])
  }

  best_index = which.max(accuracies)
```

```
  return (list(accuracy = accuracies[best_index], alpha = alphas[best_index],
               model = models[[best_index]]))
}
```

We try different alpha and choose the alpha that gives us the best accuracy.

```
best = best_alpha_bi_model(TrainCreditDF, TestCreditDF, c(0, 0.25, 0.5, 0.75, 1))
```

```
## [1] 0.8018333
## [1] 0.804
## [1] 0.804
## [1] 0.8041667
## [1] 0.8043333
```

```
best$accuracy
```

```
## [1] 0.8043333
```

```
best$alpha
```

```
## [1] 1
```

We also make a unregularized model and calculate its accuracy.

```
model = glm(X25 ~ ., family = "binomial", data = TestCreditDF)
sum((predict(model, TestCreditDF[,-ColumnNum], type = "response") > 0.5)
    == TestCreditDF[,ColumnNum]) / dim(TestCreditDF)[1]
```

```
## [1] 0.8036667
```

We can see the best regularized model is slightly better than the unregularized one if measured by accuracy.