Code

R Notebook

Exoplanet Classification Problem

For this project, we decided to perform classification on our data set. Classification is a way to predict a label, in our case categorical, based on the characteristics of the data. The

The objective of our project is to use machine learning classification models to determine if a given observation of a star should be classified as an exoplanet (a planet outside our solar system). The original data set comes from the Kepler Space Observatory (https://www.kaggle.com/nasa/kepler-exoplanet-search-results), and details 9,564 observations of potential exoplanets, along with 50 descriptive features ranging from identifiers to specific measurements. The column "koi_disposition" labelled each of the potential exoplanets as either "CONFIRMED," "FALSE POSITIVE," or "CANDIDATE". Those labeled "CANDIDATE" have not yet been determined to be exoplanets or not; the goal of our analysis will be to label them as "CONFIRMED" or "FALSE POSITIVE."

NOTE: In this context, "false positive" does not indicate a false positive output from our model. Instead, it refers to the original observation, which was considered a candidate, to have been falsely identified as a candidate. A koi_disposition of "FALSE POSITIVE" indicates that the candidate is not a planet; it is a negative result.

To solve this problem, we will carry out the following process:

1. Load, preprocess, and clean up data
2. Data visualization and exploratory data analysis
3. Creation and testing of candidate models:
    - 3.1) Models using non-scaled data:
        - 3.1.1) Decision Tree, Random Forest, and Support Vector Machine
        - 3.1.2) XGBoost
        - 3.1.3) Logistic Regression
    - 3.2) Models using scaled data:
        - 3.2.1) Neural Network: Original Variables
        - 3.2.2) Neural Network: PCA w/15 Variables
        - 3.2.3) Neural Network: PCA w/20 Variables
        - 3.2.4) K-Nearest Neighbours
        - 3.2.5) K-Means Clustering
4. Select best model
5. Use best model to make predictions

1. Data Loading, Pre-Processing and Clean Up

Load Libraries:

Hide

```
library(Matrix)    #extra Matrix functionality
library(rpart)      #Decision trees
library(rpart.plot)
library(randomForest)#random forest
library(class)      #KNN
library(e1071)      #misc. stats functionary
library(xgboost)    #XGBoost Algorithm
library(FNN)        #KNN
library(factoextra) #PCA and clustering
library(ggplot2)    #extra plotting functionality
library(corrplot)   #extra plotting functionality
#visualization packages
library(dplyr)
library(reshape2)
library(tidyverse)
library(DiagrammeR) #plotting for XGBoost
library(formulaic) #automated formula creation
library(neuralnet) #Neural Networks
```

The first step was to load the data, and convert our output variable, “koi_disposition”, to a factor

Hide

```
#read data
originalKepplerData = read.csv("cumulative.csv") #read data
factored_keppler_data = originalKepplerData
factored_keppler_data$koi_disposition = factor(originalKepplerData$koi_disposition) #fac
tor character data
factored_keppler_data$koi_pdisposition = factor(originalKepplerData$koi_pdisposition)
head(factored_keppler_data)
```

	ro... <int>	kepid <int>	kepoi_na... <chr>	kepler_name <chr>	koi_disposition <fctr>	koi_pdisposition <fctr>	koi_score <dbl>
1	1	10797460	K00752.01	Kepler-227 b	CONFIRMED	CANDIDATE	1.000
2	2	10797460	K00752.02	Kepler-227 c	CONFIRMED	CANDIDATE	0.969
3	3	10811496	K00753.01		FALSE POSITIVE	FALSE POSITIVE	0.000
4	4	10848459	K00754.01		FALSE POSITIVE	FALSE POSITIVE	0.000
5	5	10854555	K00755.01	Kepler-664 b	CONFIRMED	CANDIDATE	1.000
6	6	10872983	K00756.01	Kepler-228 d	CONFIRMED	CANDIDATE	1.000

6 rows | 1-9 of 50 columns

Our next step was to conduct forms of dimensionality reduction on our 49 possible input variables. Dimensionality reduction is important for our large data set to reduce the computational requirements of models and reduce possible overfitting or bias from un-important features. First, we removed several entirely empty columns (with values of 0 or NA), unique row identifiers (observation numbers), as well as several which could only be filled once the outcome of the observation was already known. For example, “kepler_name” is the name given to a confirmed exoplanet, so it was removed from the data set. There were no unique outlying data points that had to be explored further.

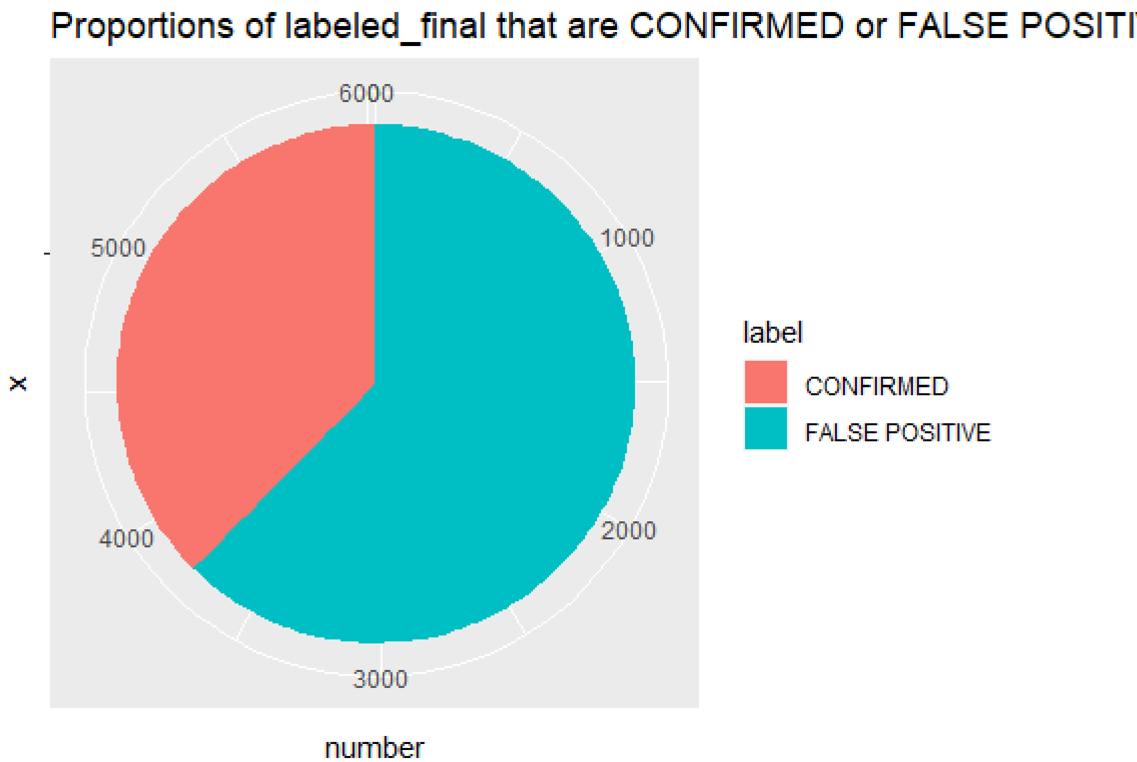
Hide

```
#remove all-NA columns:
remove_KOI_tech_factored = subset(factored_keppler_data, select = -c(koi_teq_err1)) #rem
ove
remove_KOI_tech_factored = subset(remove_KOI_tech_factored, select = -c(koi_teq_err2))
remove_NAs = na.exclude(remove_KOI_tech_factored) #remove any rows with NAs remaining

#Remove unique identifiers and rows that can only be known after the status of a planet
 is known. They therefore are not useful inputs to determine the status of a candidate.
identifiers_removed = subset(
  remove_NAs,
  select = -c(
    rowid,
    kepid,
    kepoi_name,
    kepler_name,
    koi_pdisposition,
    koi_score
  )
)

#Also needs to remove flags, these can only be known after status is confirmed
identifiers_removed = subset(
  identifiers_removed,
  select = -c(
    koi_fpflag_ss,
    koi_fpflag_ec,
    koi_fpflag_co,
    koi_fpflag_nt,
    koi_tce_plnt_num,
    koi_tce_delivname
  )
)

#Now, we will separate in labeled and unlabeled data. The labeled data will be used to t
rain our model; once the best model is selected, we will use it to predict the labels of
the unlabeled dataset
candidates_final = identifiers_removed[identifiers_removed$koi_disposition ==
                                         "CANDIDATE",] #separate out just the candidates
labeled_final = identifiers_removed[identifiers_removed$koi_disposition !=
                                      "CANDIDATE",]
labeled_final = droplevels(labeled_final)
candidates_final = droplevels(candidates_final)

numFalse = sum(labeled_final$koi_disposition=="FALSE POSITIVE")
numConfirmed = sum(labeled_final$koi_disposition=="CONFIRMED")
##check
numFalse + numConfirmed == dim(labeled_final)[1]
```

```
[1] TRUE
```

Hide

```
Proportions = data.frame(label = c("CONFIRMED","FALSE POSITIVE"), number = c(numConfirme
d,numFalse))
bp = ggplot(Proportions,aes(x = "",y=number,fill=label))+geom_bar(width = 1,stat = "iden
tity")
pie = bp+coord_polar("y", start = 0) + ggtitle("Proportions of labeled_final that are CO
NFIRMED or FALSE POSITIVE")
pie
```



Hide

```
head(labeled_final)
```

	koi_disposition <fctr>	koi_period <dbl>	koi_period_err1 <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0b
1	CONFIRMED	9.488036	2.775e-05	-2.775e-05	170.5387	0.0
2	CONFIRMED	54.418383	2.479e-04	-2.479e-04	162.5138	0.0
3	FALSE POSITIVE	19.899140	1.494e-05	-1.494e-05	175.8503	0.0
4	FALSE POSITIVE	1.736952	2.630e-07	-2.630e-07	170.3076	0.0
5	CONFIRMED	2.525592	3.761e-06	-3.761e-06	171.5956	0.0
6	CONFIRMED	11.094321	2.036e-05	-2.036e-05	171.2012	0.0

6 rows | 1-7 of 36 columns

Hide

```
head(candidates_final)
```

	koi_disposition <fctr>	koi_period <dbl>	koi_period_err1 <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0
38	CANDIDATE	4.959319	5.150e-07	-5.150e-07	172.2585	8
59	CANDIDATE	40.419504	1.139e-04	-1.139e-04	173.5647	2
63	CANDIDATE	7.240661	1.617e-05	-1.617e-05	137.7554	2
64	CANDIDATE	3.435916	4.729e-05	-4.729e-05	132.6624	1
73	CANDIDATE	1.626630	1.015e-06	-1.015e-06	169.8202	4
85	CANDIDATE	10.181584	6.188e-06	-6.188e-06	177.1419	4

6 rows | 1-7 of 36 columns

We are left with two datasets:

- candidates_final, which has 1772 data points and 35 features, which will be what we use the best model on
- labeled_final, which has 6031 data points and 35 features, which will be what we use to train and test candidate models, as well as build our final model.

2) Data Visualization and Exploratory Data Analysis

Select the variables to be used:

Hide

```
#All variables have errors - for exploratory data analysis we will only be looking at th
e variables themselves, not their errors
variablesonly = labeled_final %>%
  select(koi_period, koi_time0bk, koi_impact, koi_duration, koi_depth, koi_prad, koi_te
q, koi_insol, koi_model_snr, koi_steff, koi_slogg, koi_srad, ra, dec, koi_kepmag)
```

Create a correlation heat map with only the variables (excluding the errors).

Hide

```
cormat <- round(cor(variablesonly),2)
head(cormat)
```

```
            koi_period koi_time0bk koi_impact koi_duration koi_depth
koi_period        1.00        0.60      -0.03         0.33     -0.05
koi_time0bk       0.60        1.00       0.02         0.20     -0.05
koi_impact       -0.03        0.02       1.00         0.06      0.02
koi_duration      0.33        0.20       0.06         1.00      0.09
koi_depth        -0.05       -0.05       0.02         0.09      1.00
koi_prad         -0.01       -0.01       0.54         0.02      0.08
            koi_prad koi_teq koi_insol koi_model_snr koi_steff koi_slogg
koi_period     -0.01   -0.35     -0.02         -0.04      0.01     -0.04
koi_time0bk    -0.01   -0.28     -0.02         -0.04     -0.01      0.02
koi_impact      0.54    0.05     -0.01          0.03      0.08     -0.03
koi_duration    0.02   -0.19     -0.02          0.10      0.10     -0.13
koi_depth       0.08    0.06     -0.01          0.60      0.15     -0.03
koi_prad        1.00    0.12      0.05          0.05     -0.01     -0.17
            koi_srad    ra   dec koi_kepmag
koi_period      0.01 -0.05  0.02      -0.02
koi_time0bk    -0.01 -0.03  0.00       0.03
koi_impact      0.00  0.07 -0.03       0.02
koi_duration    0.02  0.04 -0.03      -0.10
koi_depth      -0.02  0.02 -0.01       0.00
koi_prad        0.19  0.03  0.00      -0.01
```

Hide

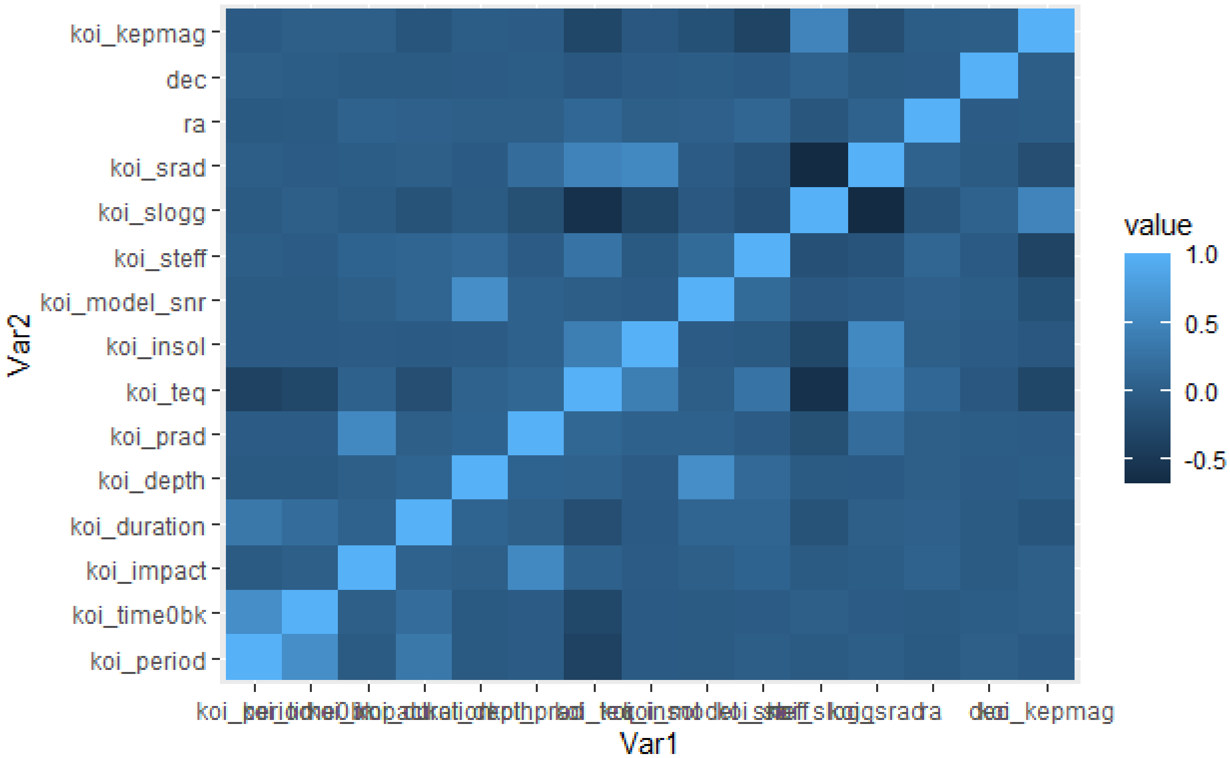
```
melted_cormat <- melt(cormat)
head(melted_cormat)
```

	Var1 <fctr>	Var2 <fctr>	value <dbl>
1	koi_period	koi_period	1.00
2	koi_time0bk	koi_period	0.60
3	koi_impact	koi_period	-0.03
4	koi_duration	koi_period	0.33
5	koi_depth	koi_period	-0.05
6	koi_prad	koi_period	-0.01

6 rows

Hide

```
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()
```



Defining significant correlations as those greater than .4, there are a few; this will need to be dealt with in Logistic Regression, but should not affect other models.

Plots by FALSE POSITIVE and CONFIRMED

We will now observe the distribution of several variables across false positive and confirmed exoplanets. This will help determine where there is a clear pattern in certain variables.

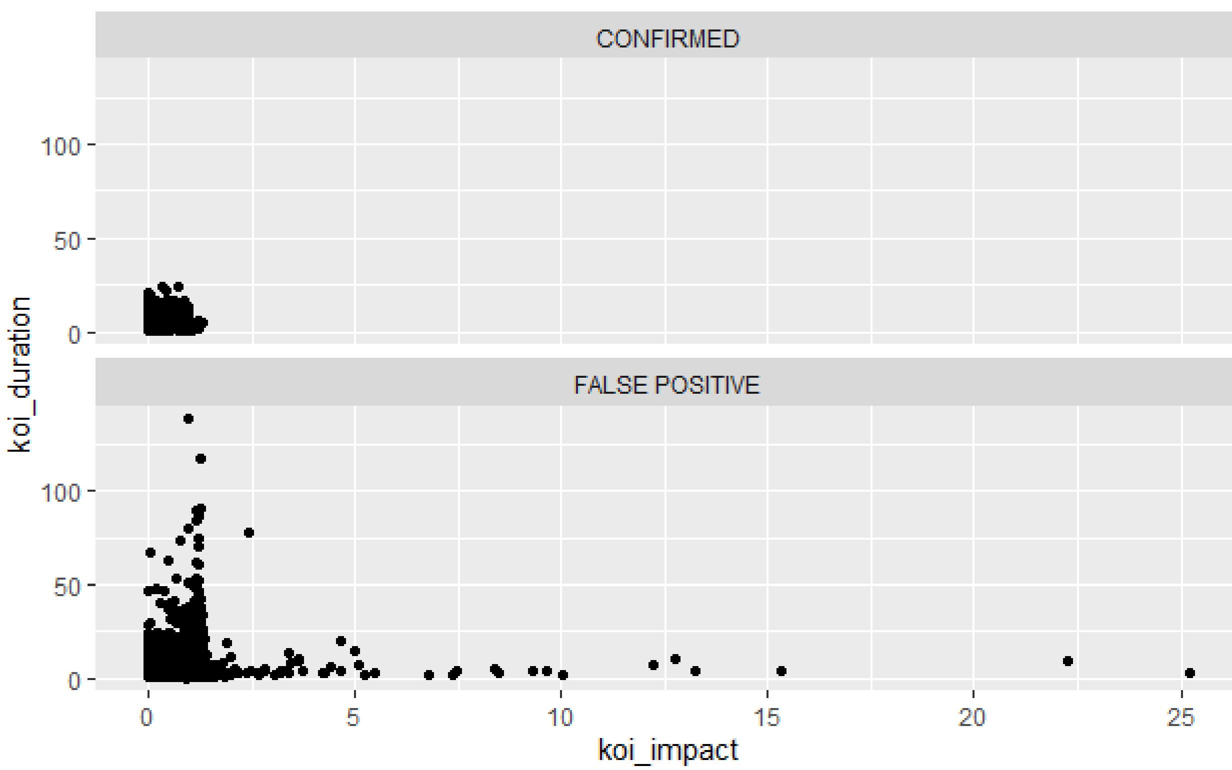
Hide

```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_period, y = koi_time0bk)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```

For koi_period and koi_time0bk, the distributions of the variables seem similars, with some outliers. That being said, FALSE POSITIVEs can have significantly higher koi_periods than CONFIRMEDs

Hide

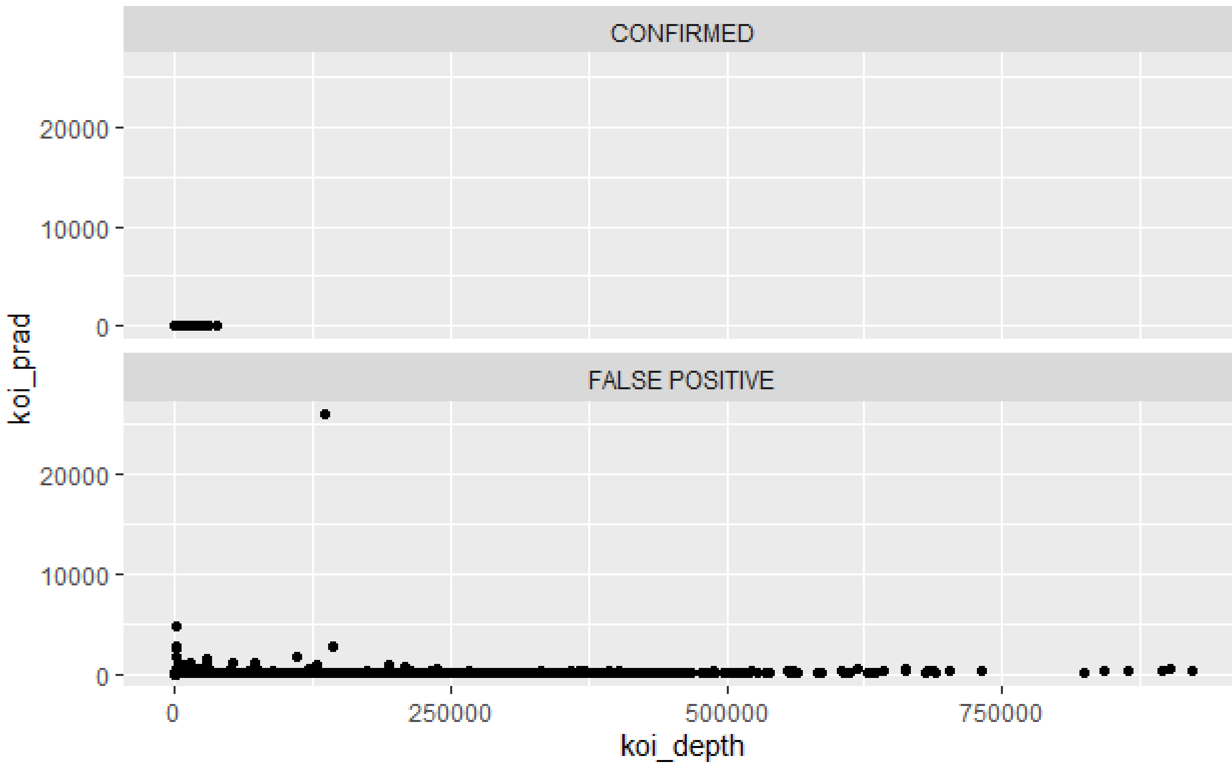
```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_impact, y = koi_duration)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```



The plot of koi_impact vs. koi_duration shows clear differences between their distributions - CONFIRMEDs have a significantly small range for both variables, although those candidates that fall inside that range may be difficult to classfiy.

Hide

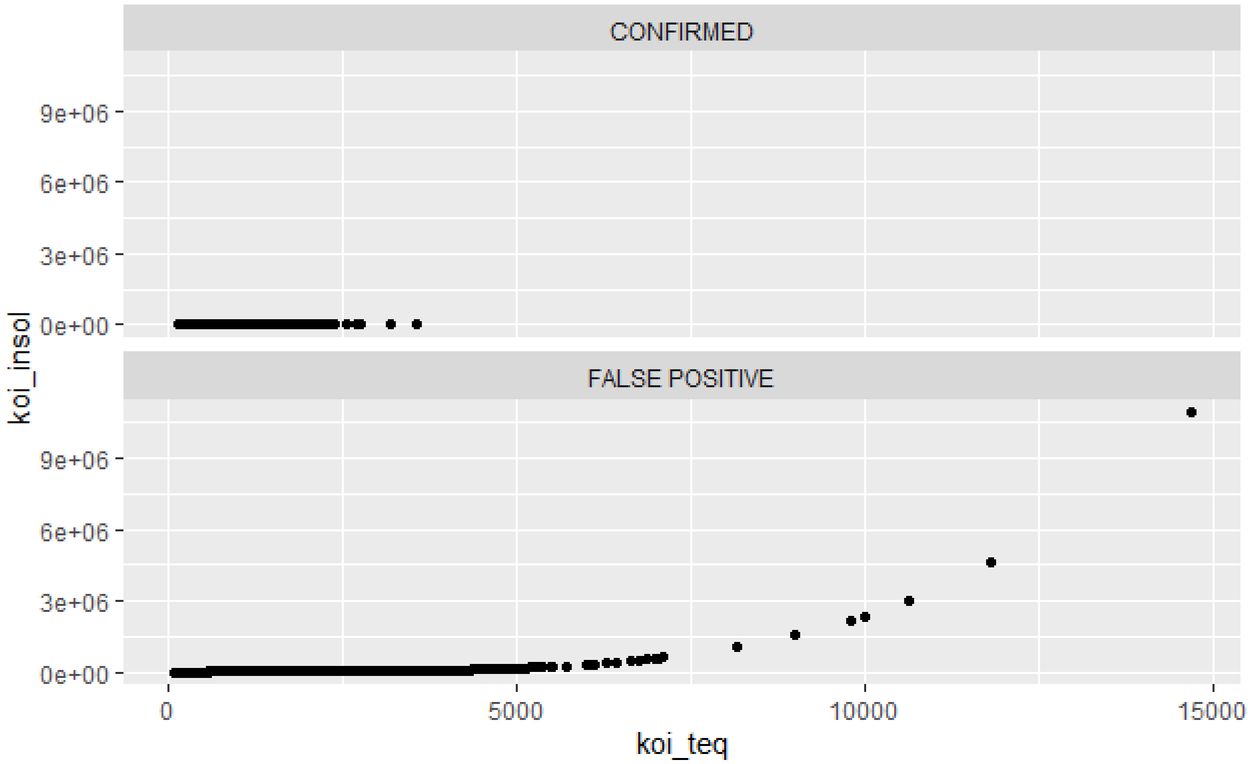
```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_depth, y = koi_prad)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```



koi_depth shows a clear different: high koi_depths almost always indicate FALSE POSITIVEs

Hide

```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_teq, y = koi_insol)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```



koi_teq has a significantly smaller range for CONFIRMED data points, and KOI_insol stays closer to zero. If we look at

Hide

```
summary(labeled_final$koi_insol)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
       0       36      219     8148     1332 10947555 
```

Hide

```
summary(labeled_final$koi_teq)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
     92     626     981    1192    1540   14667 
```

This confirms that there are very large values for FALSE POSITIVES that are significantly outside the normal data range. It will be easy to classify these as FALSE POSITIVEs. We can explore these graphs with a smaller range, to exclude the outliers:

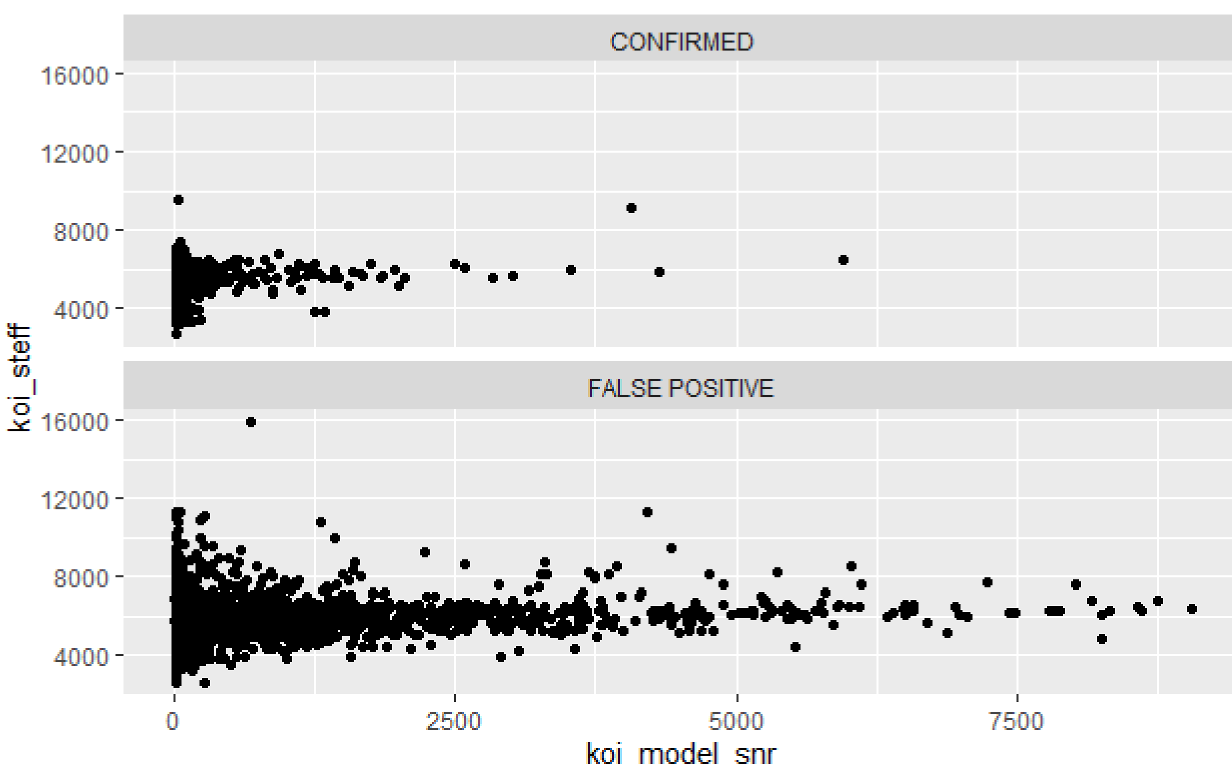
Hide

```
ggplot(data = labeled_final[labeled_final$koi_insol<250000,]) +
  geom_point(mapping = aes(x = koi_teq, y = koi_insol)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```

This shows that for values of koi_insol < 25000 and koi_teq<3000, it becomes difficult to identify whether a given data point is FALSE POSITIVE or CONFIRMED.

Hide

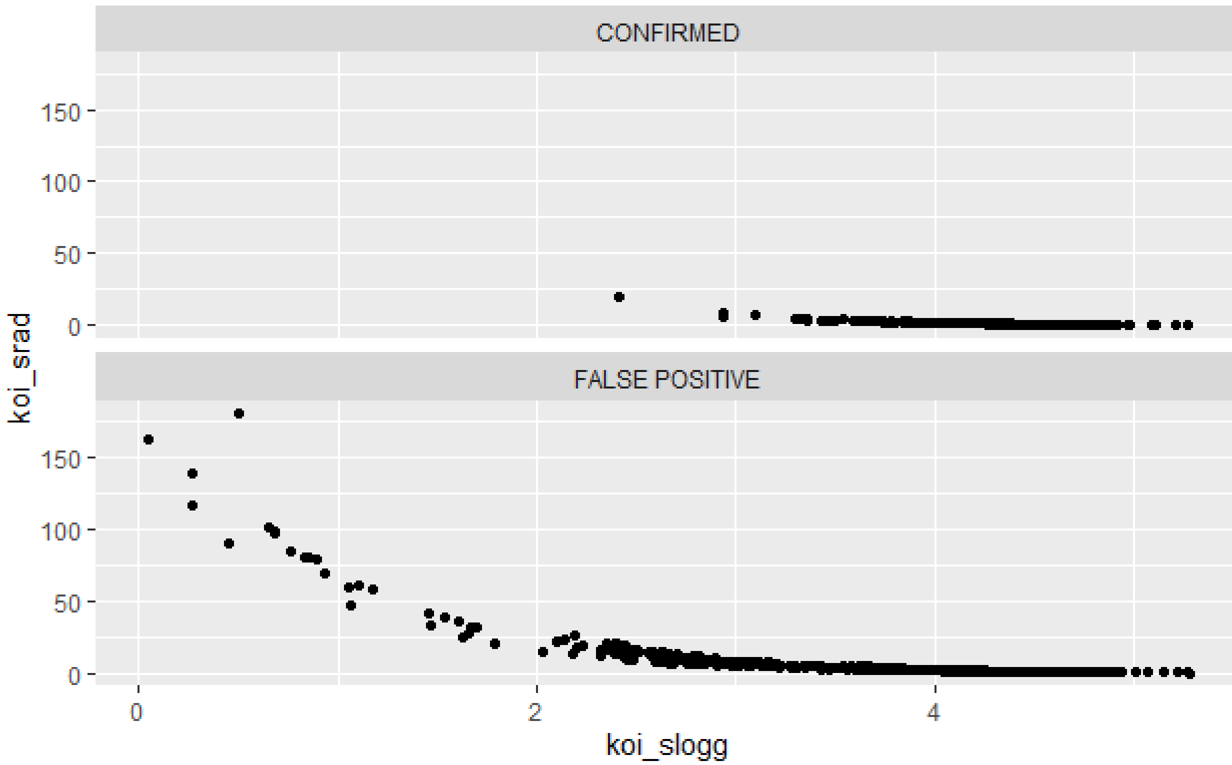
```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_model_snr, y = koi_steff)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```



Once again, FALSE POSITIVES show significantly more variability on both axes than CONFIRMEDs

Hide

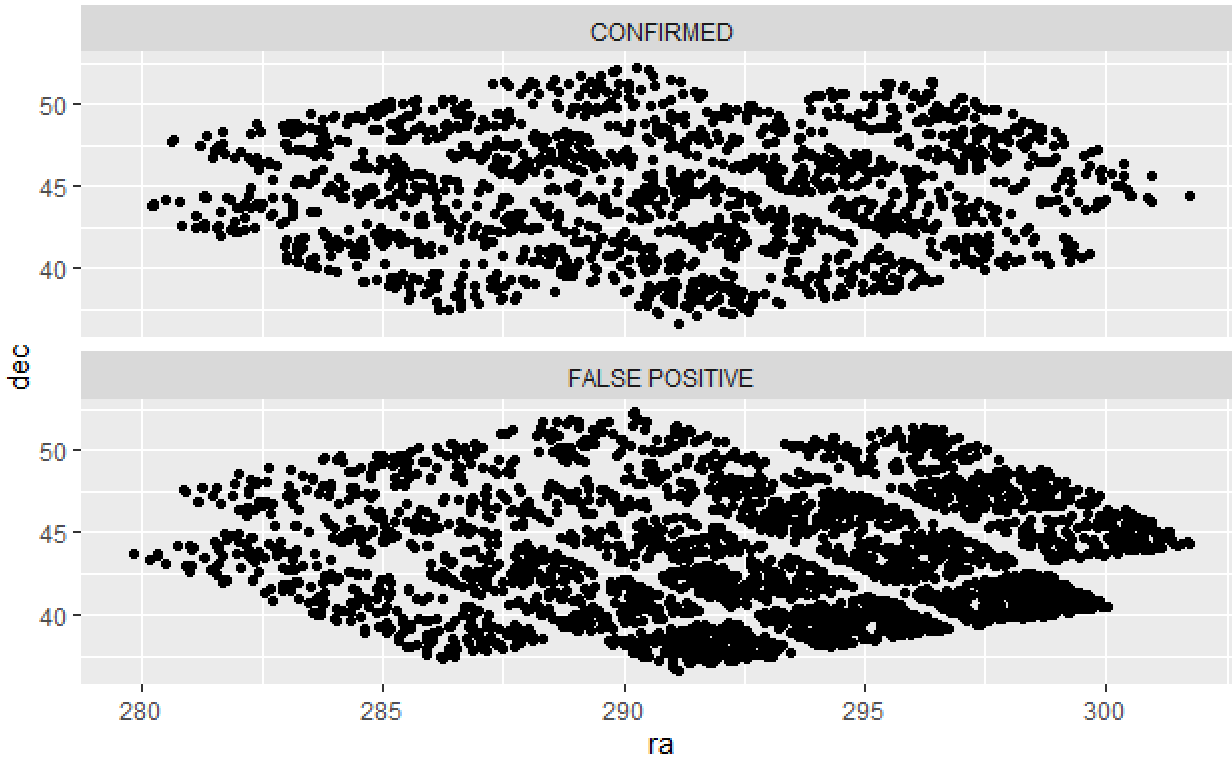
```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = koi_slogg, y = koi_srad)) +
  facet_wrap(~ koi_disposition, nrow = 2)
```



FALSE POSITIVEs can have significantly lower koi_sloggs and higher koi_srads than CONFIRMEDs

Hide

```
ggplot(data = labeled_final) +
  geom_point(mapping = aes(x = ra, y = dec)) +
  facet_wrap( ~ koi_disposition, nrow = 2)
```



Lastly, for ra and dec, there is no clear different in distribution for CONFIRMED or FALSE POSITIVE.

Overall, an easy way to determine FALSE POSITIVEs tends to be to look for data points outside a given range. Inside that range, the determination of whether an observation is a exoplanet or not becomes more difficult.

3. Creation and testing of candidate models

Tto classify the data and determine which observation are, in fact, planets, our group first had to determine which model most accurately predicted our labeled data set. We first examined models that did not require scaling, these included: Decision Tree, Random Forest, XGBoost, Logistic Regression, and Support Vector Machines (SVM). Next, we examined models that required scaling, these included: KNN, Neural Network (with and without Principal Component Analysis), and K-means clustering. To choose the best model for our data set, our group decided to look at which model produced the lowest misclassification rate. However, it should be noted that in real-life application a number of other metrics should be considered as well. These include, but are not limited to, precision analysis (PR curves), recall, ROC-AUC, and F1 Scores.

When running the various models, our group incorporated K-fold cross-validation. The benefit of this approach is that it allows the model to become more generalized, helping with over-fitting concerns. In addition, we take the average result of five misclassification rates for each model, significantly lowering the chance that a given misclassification rate is produced only by chance due to a specific testing/training set. This allows us to be more confident that the model with the lowest misclassification rate truly is the best. Due to computational limits, we decided to use a K value of 5 as we believed that would be adequate for our purposes. Finally, when deciding the proportion of training and testing set, we decided to follow class standards with 80% training set and 20% testing set. Our specific data set is fairly large with ~6,031 rows; we believe having ~4800 rows to train and ~1,200 rows to test is large enough for each purpose.

3.1) Models using non-scaled data:

We first wanted to create a “baseline” model - that is, a model that predicts the average proportions of our training set every time. This will give a baseline misclassification rate that we are trying to beat.

Hide

```
set.seed(123)
num_samples = dim(labeled_final)[1]
sampling.rate = 0.8
training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
trainingSet = subset(labeled_final[training,])
testing = setdiff(1:num_samples, training)
testingSet = subset(labeled_final[testing, ])
sizeTrainingSet = dim(trainingSet)[1]
sizeTestingSet = dim(testingSet)[1]
propC = sum(trainingSet[, 1] == "CONFIRMED") / sizeTrainingSet #the proportion of the tr
aining set that is confirmed
naivePredictions = sample(
  c("CONFIRMED", "FALSE POSITIVE"),
  sizeTestingSet,
  replace = TRUE,
  prob = c(propC, 1 - propC)
) #predict taking a random sample, with the same proportions as the training set


Errors = sum(testingSet$koi_disposition != naivePredictions)
ErrorRateNaive = Errors / sizeTestSet
paste("Naive Error Rate: ", round(ErrorRateNaive * 100, 2), "%", sep = "")
```

```
[1] "Naive Error Rate: 46.73%"
```

This “naive” method, which makes predictions based purely off proportions and chance, gives an error rate of 46.73%. This is the “number to beat” for the rest of our models.

3.1.1) Decision Tree, Random Forest, and SVM

The goal of a decision tree is to make splitting decisions on the data, in an effort to minimize the least squares, thus creating a tree-like structure. These models are useful as a starting point because they are easy to interpret as the plot can display which variables have the highest importance in the tree. Normalization and other data cleaning is also not required for this model. Adding more depth to the tree can reduce the fitting error to the data, but it can lead to overfitting the model. As a result, the complexity parameter, cp, must be tuned to ensure that the optimal depth-to-fit of the model is used. As the cp value decreases, so does the relative error in the model. We automatically prune our decision tree to select the cp where the change in error is less than 0.05; this is our first example of parameter tuning.

Random forest models are more accurate and robust but harder to interpret than a single tree. The model creates many decision trees with different randomized learning and testing sets, then the trees “vote” or “average” their results to determine the resultant random forest model. Though the model is not as interpretable as a single tree and it is more difficult to understand the significance of a single variable, it will result in lower misclassification rate. The number of trees in the forest is a parameter that needs to be tuned in this model. As the number of trees increases, the error decreases exponentially, reaching an asymptote of error.

SVM models are supervised learning models that use the data points to create a line to separate the data. This separation then decides the binary classification for each data point. While this model can be incredibly versatile and robust against outliers and inaccurate data, it may not be as accurate if there is much overlap between the data.

Helper Function:

Hide

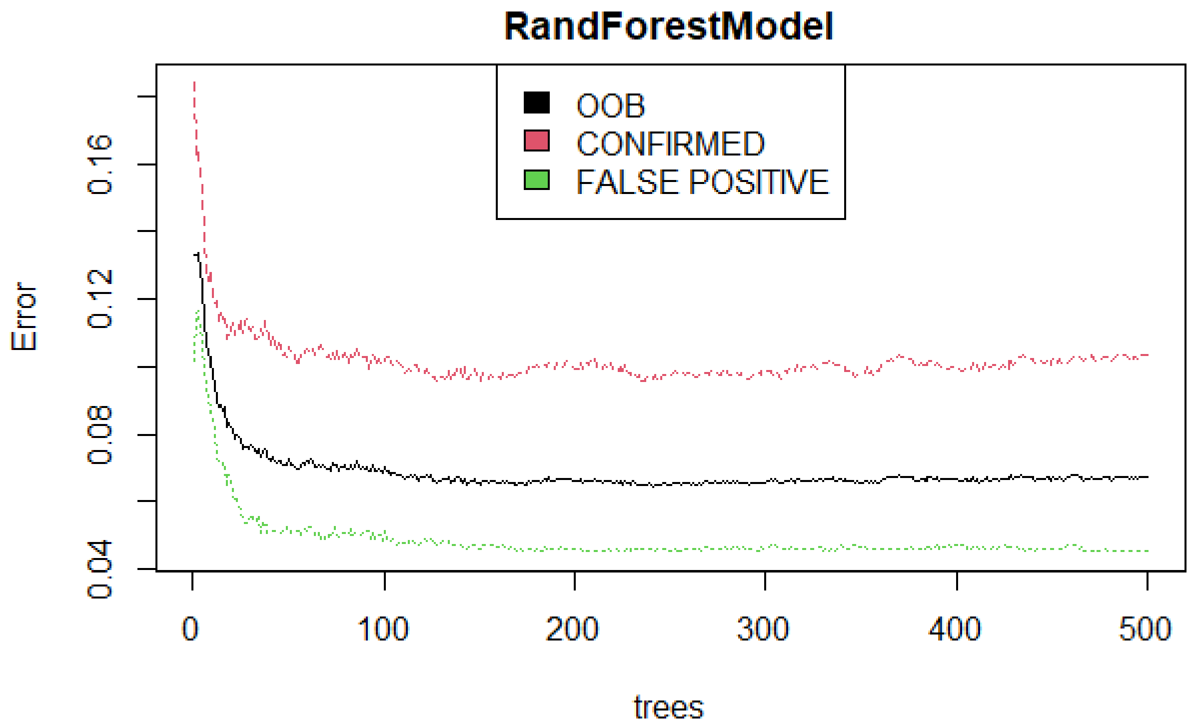
```
AllErrors = function(correctResults,
                     predictedResults,
                     sizeTestSet, numerical = 0) {
  isWrong = (correctResults != predictedResults)
  isRight = (correctResults == predictedResults)
  Errors = sum(correctResults != predictedResults)
  ErrorRate = Errors / sizeTestSet


  if (numerical == 0) {
    IsC = (predictedResults == 'CONFIRMED')
    IsF = (predictedResults == 'FALSE POSITIVE')
  }else{
    IsC = (predictedResults == 1)
    IsF = (predictedResults == 0)
  }

  FalsePositives = sum(isWrong & IsC)
  FalseNegatives = sum(isWrong & IsF)
  TruePositives = sum(isRight & IsC)
  TrueNegatives = sum(isRight & IsF)
  FP_Rate = (FalsePositives / (FalsePositives + TrueNegatives))#define FP rate as FP/(FP
+TN)
  FN_Rate = (FalseNegatives / (FalseNegatives + TruePositives))#define FN rate as FN/(FN
+TP)
  return(list(ErrorRate, FP_Rate, FN_Rate))
}
```

Code for decision tree, random forest, and SVM:

Hide

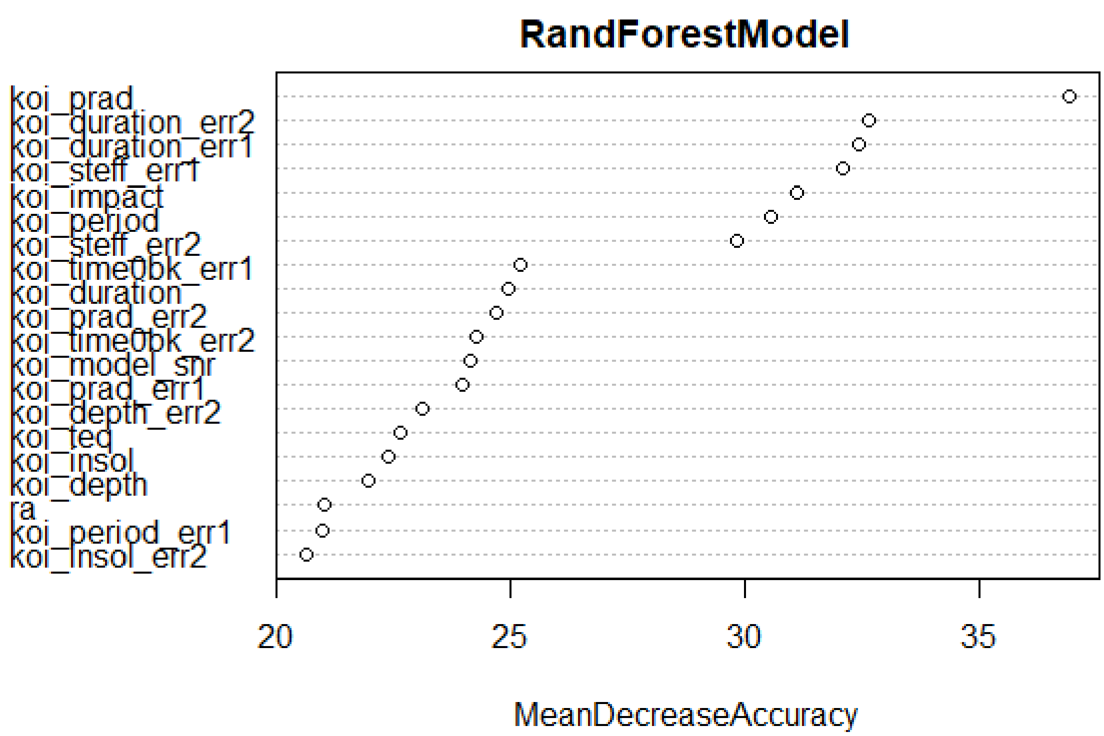
```
NumFolds = 5
CreateErrorMatrix = function(numFolds) {
  return(rep(0, numFolds))
}


decTree_Error = CreateErrorMatrix(NumFolds)
decTree_FP = CreateErrorMatrix(NumFolds)
decTree_FN = CreateErrorMatrix(NumFolds)

RF_error = CreateErrorMatrix(NumFolds)
RF_error_FP = CreateErrorMatrix(NumFolds)
RF_error_FN = CreateErrorMatrix(NumFolds)
SVM_error = CreateErrorMatrix(NumFolds)
SVM_error_FP = CreateErrorMatrix(NumFolds)
SVM_error_FN = CreateErrorMatrix(NumFolds)
for (fold in 1:NumFolds) {
  #set up k-crossfold validation
  set.seed(fold)


  #split data into testing and training sets
  num_samples = dim(labeled_final)[1]
  sampling.rate = 0.8
  training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
  trainingSet = subset(labeled_final[training, ])
  testing = setdiff(1:num_samples, training)
  testingSet = subset(labeled_final[testing,])


  #Decision tree
  decTreeModel = rpart(koi_disposition ~ ., data = trainingSet)

  #Automatically select the stopping point where cp no longer improves error by 0.05
  errors = decTreeModel$cptable[, 3]
  decTreeChangeError = c(0, 0, 0, 0, 0, 0, 0, 0, 0)
  for (i in 1:8) {
    decTreeChangeError[i] = errors[i + 1] - errors[i]
  }
  decTreeChangeError
  for (i in 1:9) {
    if (abs(decTreeChangeError[i]) < 0.05) {
      stopIndex = i
      break
    }
  }
  cps = decTreeModel$cptable[, 1]
  cpStop = cps[stopIndex]
```

```
  prunedDecTreeModel = rpart::prune(decTreeModel, cp = cpStop) #Prune decision tree
  decTreePredictions = predict(prunedDecTreeModel, testingSet, type = "class") #make pre
dictions
  #Determine decision tree error
  sizeTestSet = dim(testingSet)[1]
  decTreeModel_errors = AllErrors(testingSet$koi_disposition,
                                  decTreePredictions,
                                  sizeTestSet)
  decTree_Error[fold] = decTreeModel_errors[[1]]
  decTree_FP[fold] = decTreeModel_errors[[2]]
  decTree_FN[fold] = decTreeModel_errors[[3]]
  #Random Forest
  RandForestModel = randomForest(koi_disposition ~ ., data = trainingSet, ntrees = 200,
 importance = TRUE)#visual inspection gives this as a good number for trees



  predictedLabels = predict(RandForestModel, testingSet)

  #Determine Random Forest Error
  sizeTestSet = dim(testingSet)[1]
  RF_Errors = AllErrors(testingSet$koi_disposition, predictedLabels, sizeTestSet)


  RF_error[fold] = RF_Errors[[1]]
  RF_error_FP[fold] = RF_Errors[[2]]
  RF_error_FN[fold] = RF_Errors[[3]]

  #SVM Model
  svmModel = svm(koi_disposition ~ ., data = trainingSet, kernel = "linear")
  predictedlabelsSVM = predict(svmModel, testingSet)
  #Determine SVM error
  SVM_Errors = AllErrors(testingSet$koi_disposition,
                         predictedlabelsSVM,
                         sizeTestSet)

  errorSVM = sum(predictedlabelsSVM != testingSet$koi_disposition)
  misclassification_rateSVM = errorSVM / sizeTestSet
  SVM_error[fold] = SVM_Errors[[1]]
  SVM_error_FP[fold] = SVM_Errors[[2]]
  SVM_error_FN[fold] = SVM_Errors[[3]]
}
#Show varimp plot and error plot of 1, example random forest

plot(RandForestModel) #the error stabilizes around 200 trees; this is why this is the nu
mber we chose to use
legend("top", colnames(RandForestModel$err.rate), fill = 1:3)
```



Hide

```
varImpPlot(RandForestModel, type = 1, n.var = 20) #This shows that, for random forest, t
he most important variables are prad and the errors on duration. This shows that the imp
ortance of errors is actually quite high, which is not what would be intuitively expecte
d. To better triage variable importance in models that cannot use all of the variables f
or computational reasons, (Neural Networks), we will therefore use PCA to reduce dimensi
onality
```



Hide

```
#Take average of errors from each fold to determine average error for each model

AvgErrorDT = mean(decTree_Error)
AvgFP_DT = mean(decTree_FP)
AvgFN_DT = mean(decTree_FN)
AvgErrorRF = mean(RF_error)
AvgFP_RF = mean(RF_error_FP)
AvgFN_RF = mean(RF_error_FN)
AvgErrorSVM = mean(SVM_error)
AvgFP_SVM = mean(SVM_error_FP)
AvgFN_SVM = mean(SVM_error_FN)

paste("DT Error: ", round(100 * AvgErrorDT, 2), "%", sep = "")
```

```
[1] "DT Error: 11.58%"
```

Hide

```
paste("RF Error: ", round(100 * AvgErrorRF, 2), "%", sep = "")
```

```
[1] "RF Error: 6.66%"
```

Hide

```
paste("SVM Error: ", round(100 * AvgErrorSVM, 2), "%", sep = "")
```

```
[1] "SVM Error: 8.7%"
```

All three models give low misclassification rates <12%, but Random Forest gives the best misclassification rate at only 6.74%.

3.1.2) XGBoost

XGBoost stands for “extreme gradient boosting” and is an open-source tree learning algorithm similar to random forests that is also widely used in industry. This model seeks to minimize an objective function representing model complexity and loss (error), using a gradient descent algorithm to minimize loss when adding new models. This is known as tree boosting; random forest models differ because they use a tree bagging algorithm, possibly leading to different model accuracies.

XGBoost outputs a probability between 0 and 1, rather than a binary classification. For this reason, it is necessary to determine the “threshold” where a value stops being a FALSE POSITIVE, and start being a CONFIRMED. Questioning the classification threshold which is built into our models can help us develop more accurate models. For example, arbitrarily assuming that the threshold for classifying based on our XGBoost model was exactly 0.5 could have resulted in a higher misclassification rate if we were to consider all possible thresholds. As a result, we conducted threshold analysis on all our models that output probabilities by looping through all classification thresholds at 0.1 increments, plotted them against their respective misclassification rates, and took the minimum as the optimal. The first model that we have done this for is XGBoost.

Hide

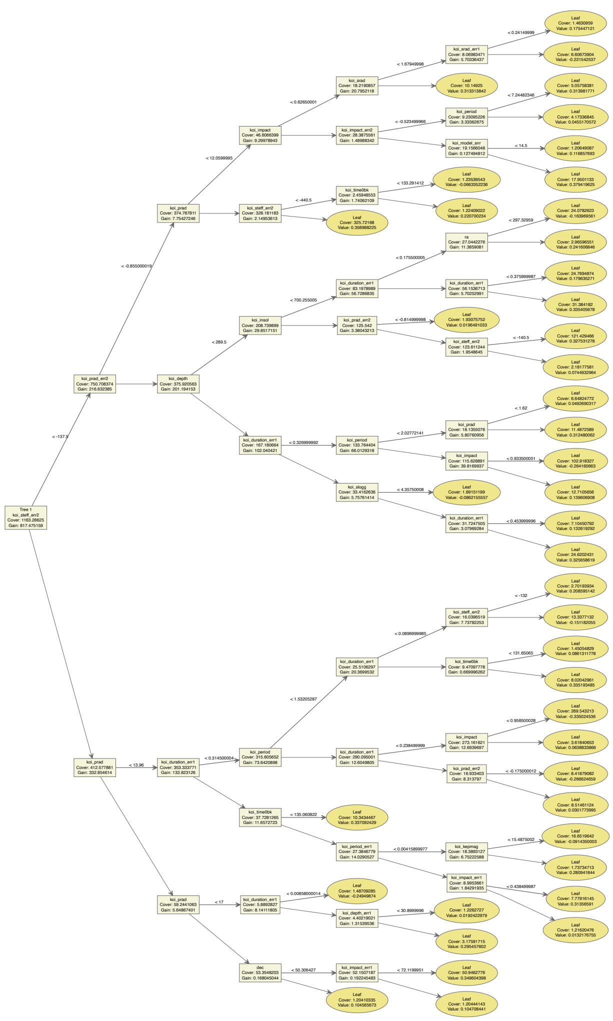
```
xgb.set.config(verbosity = 0)
```

```
[1] TRUE
```

Hide

```
threshold = 0.1
XGBoost_error_thresholds = rep(0, 10)
XGBoost_FP_thresholds = rep(0,10)
XGBoost_FN_thresholds = rep(0,10)
index = 1

while (threshold < 1) {
  #loop that reruns the algorithm with increments of 0.1 in the threshold
  XGB_error = CreateErrorMatrix(NumFolds)
  XGB_FNs = CreateErrorMatrix(NumFolds)
  XGB_FPs = CreateErrorMatrix(NumFolds)
  for (fold in 1:NumFolds) {
    #k cross-fold validation
    set.seed(fold)

    num_samples = dim(labeled_final)[1]

    #create testing and training set
    sampling.rate = 0.8
    training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
    trainingSet = subset(labeled_final[training,])
    testing = setdiff(1:num_samples, training)
    testingSet = subset(labeled_final[testing, ])

    #XGBoost model
    xgTrain = data.matrix(trainingSet)
    xgTrain[, 1] = ifelse(xgTrain[, 1] == 2, 1, 0)

    xgBoostModel = xgboost(
      data = xgTrain[, 2:36],
      label = xgTrain[, 1],
      max.depth = 6,
      eta = .22,
      nrounds = 100,
      verbose = 0,
      objective = "binary:logistic",
      eval_metric="error"
    )
    xgTest = data.matrix(testingSet)
    xgTest[, 1] = ifelse(xgTest[, 1] == 2, 1, 0)



    #make predictions
    BoostPredictions = predict(xgBoostModel, data.matrix(testingSet)[, 2:36])




    BoostPredictionsRounded = ifelse(BoostPredictions > threshold, 1, 0) #convert probab
ilities to ouputs of 1 or 0 based on whether they are greater than the threshold - this
 is where parameter tuning occurs.
```

```
    BoostErrors = AllErrors(xgTest[, 1],BoostPredictionsRounded,dim(xgTest)[1],1)
    XGB_error[fold] = BoostErrors[[1]]
    XGB_FPs = BoostErrors[[2]]
    XGB_FNs = BoostErrors[[3]]

  }
  XGBoost_error_thresholds[index] = mean(XGB_error)#average the error from all folds
  XGBoost_FP_thresholds[index] = mean(XGB_FPs)
  XGBoost_FN_thresholds[index] = mean(XGB_FNs)

  index = index + 1
  threshold = threshold + .1
}
xgbPlot = xgb.plot.tree(model = xgBoostModel,
                        trees = 1,
                        render = TRUE)
xgbPlot #plot an example tree
```



Hide

```
#Determine correct threshold value
XGBoost_error_thresholds#shows the average error at each threshold
```

```
 [1] 0.07854184 0.06611433 0.06197183 0.06081193 0.06081193 0.06362883
 [7] 0.06545153 0.06992543 0.07887324 0.62220381
```

Hide

```
order(XGBoost_error_thresholds) #Lowest error is threshold = .4
```

```
 [1]  4  5  3  6  7  2  8  1  9 10
```

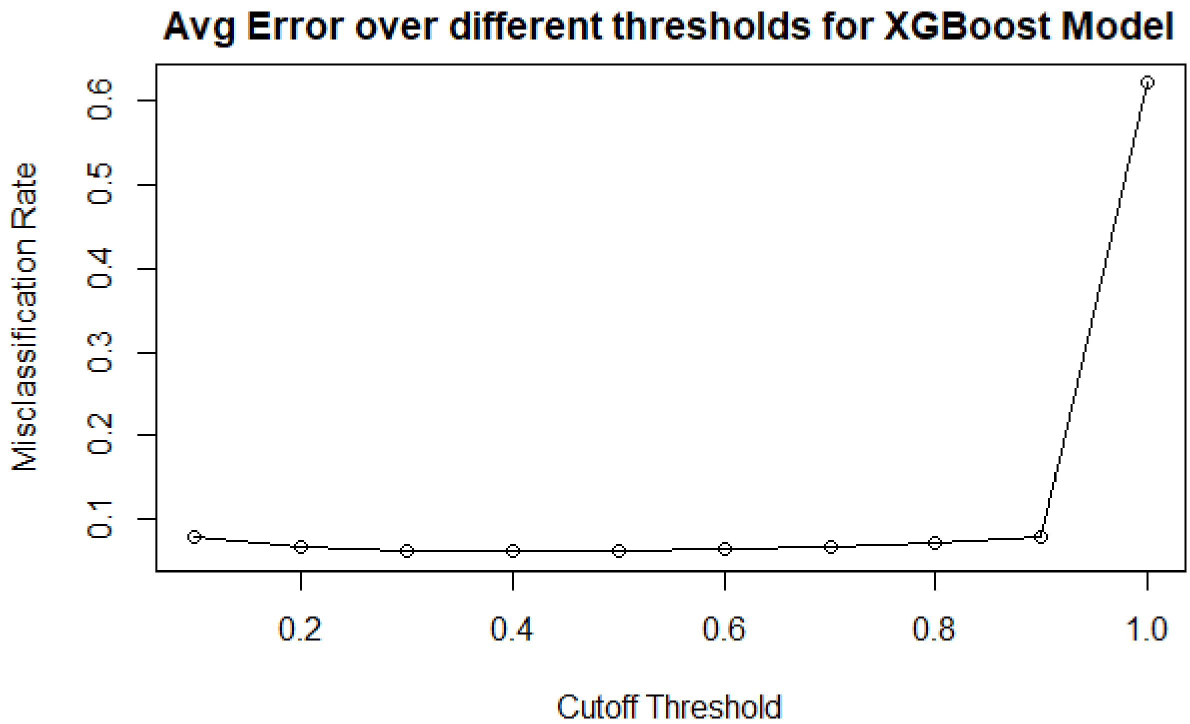
Hide

```
AvgErrorXGB = XGBoost_error_thresholds[(order(XGBoost_error_thresholds)[1])] #chose the
 threshold with the lowest error as the one we use
AvgFP_XGB = XGBoost_FP_thresholds[(order(XGBoost_error_thresholds)[1])]
AvgFN_XGB = XGBoost_FN_thresholds[(order(XGBoost_error_thresholds)[1])]
paste("XGBoost Error: ", round(AvgErrorXGB*100,2), "%", sep = "")
```

```
[1] "XGBoost Error: 6.08%"
```

Hide

```
#plot the error thresholds
plot(
  x = 1:10 / 10,
  y = XGBoost_error_thresholds,
  main = "Avg Error over different thresholds for XGBoost Model",
  xlab = "Cutoff Threshold",
  ylab = "Misclassification Rate"
)
lines(x = 1:10 / 10, y = XGBoost_error_thresholds)
```



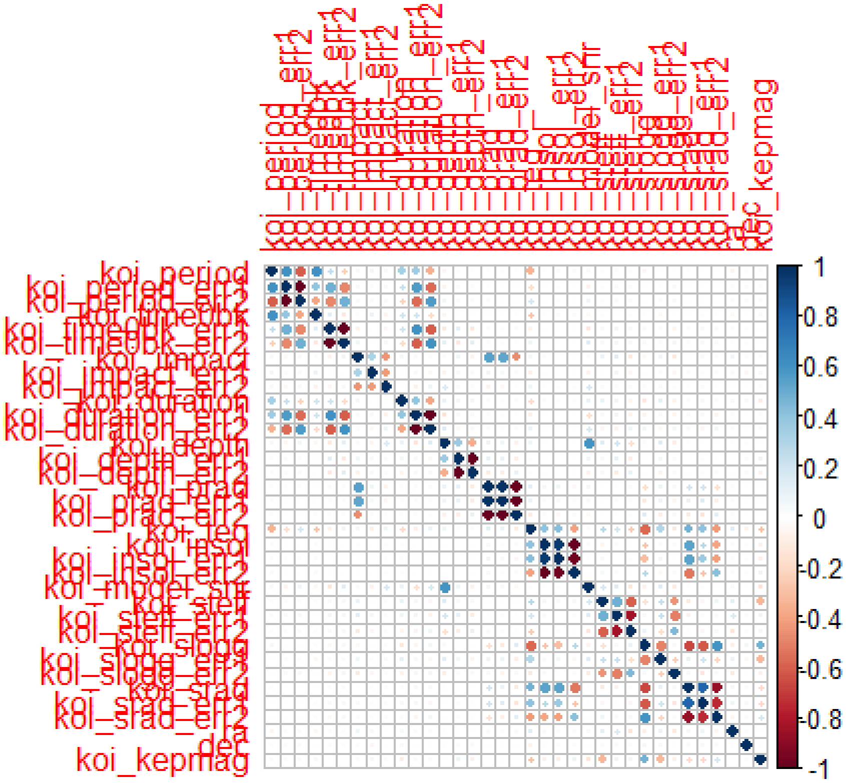
XGBoost gives an error rate of 6.08%. This is significantly better than any model run so far. The threshold analysis graph above shows that the best misclassification rate for XGBoost is at the threshold = .4.

3.1.5) Logistic Regression

Logistic regression models is a supervised classification algorithm that builds a regression model to predict the classification by assigning data entries to binary values, based on the Sigmoid function. When performing logistic regression, it is important to consider the problems that arise from multicollinearity which can cause unstable estimates and inaccuracy. For this reason, we decided to first remove all major multicollinearity from the model.

Hide

```
#Check for multicollinearity
corrFrame = data.frame(cor(labeled_final[, 2:36]))
corrplot(cor(labeled_final[, 2:36])) #many multicollinear variables. Definine collineari
ty as correlation >.4
```



Hide

```
GLM_Data = data.frame(labeled_final$koi_disposition)


#all err2s are collinear with err1s. Removing err1s
variable_counter = 2
variable.names = colnames(labeled_final)
for (i in 2:length(labeled_final)) {
  variable.names[i]
  error1 = grepl("_err1", variable.names[i], fixed = TRUE)
  if (error1 == FALSE) {
    GLM_Data[, variable_counter] = labeled_final[, i]
    variable_counter = variable_counter + 1
  } else{
    variable.names[i] = NA
  }
}
variable.names = na.omit(variable.names)
colnames(GLM_Data) = variable.names
GLM_Data
```

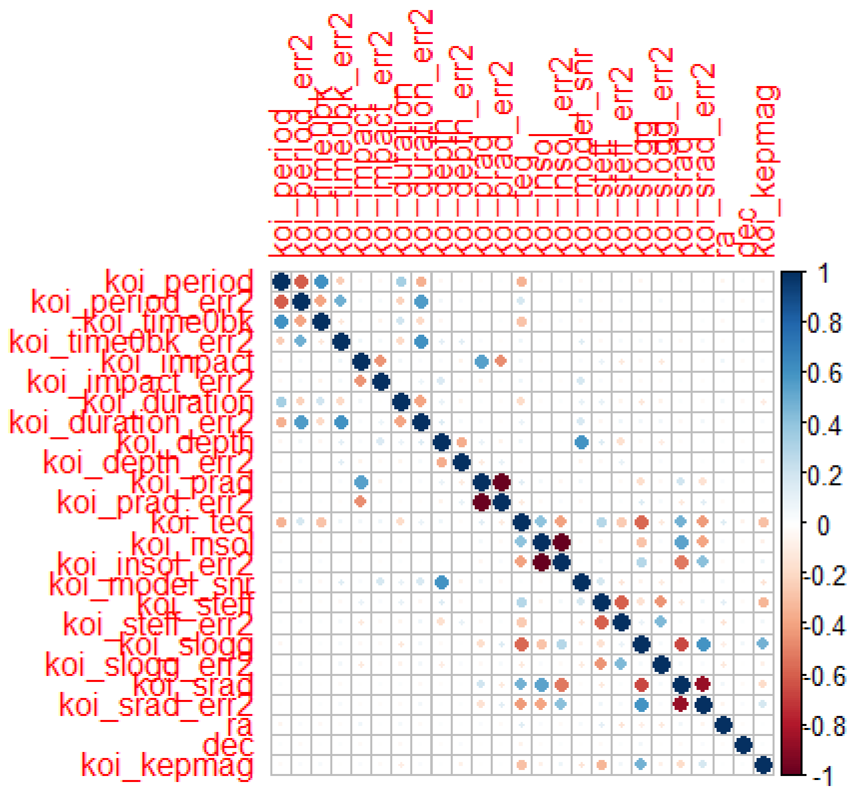
koi_disposition <fctr>	koi_period <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0bk_err2 <dbl>	koi_impact <dbl>
CONFIRMED	9.4880356	-2.775e-05	170.5387	-2.16e-03	0.146
CONFIRMED	54.4183827	-2.479e-04	162.5138	-3.52e-03	0.586

koi_disposition <fctr>	koi_period <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0bk_err2 <dbl>	koi_impact <dbl>
FALSE POSITIVE	19.8991399	-1.494e-05	175.8503	-5.81e-04	0.969
FALSE POSITIVE	1.7369525	-2.630e-07	170.3076	-1.15e-04	1.276
CONFIRMED	2.5255918	-3.761e-06	171.5956	-1.13e-03	0.701
CONFIRMED	11.0943205	-2.036e-05	171.2012	-1.41e-03	0.538
CONFIRMED	4.1344351	-1.046e-05	172.9794	-1.90e-03	0.762
CONFIRMED	2.5665890	-1.781e-05	179.5544	-4.61e-03	0.755
FALSE POSITIVE	7.3617896	-2.128e-05	132.2505	-2.53e-03	1.169
CONFIRMED	16.0686467	-1.088e-05	173.6219	-5.17e-04	0.052

1-10 of 6,031 rows | 1-6 of 26 columns Previous **1** 2 3 4 5 6 ... 100 Next

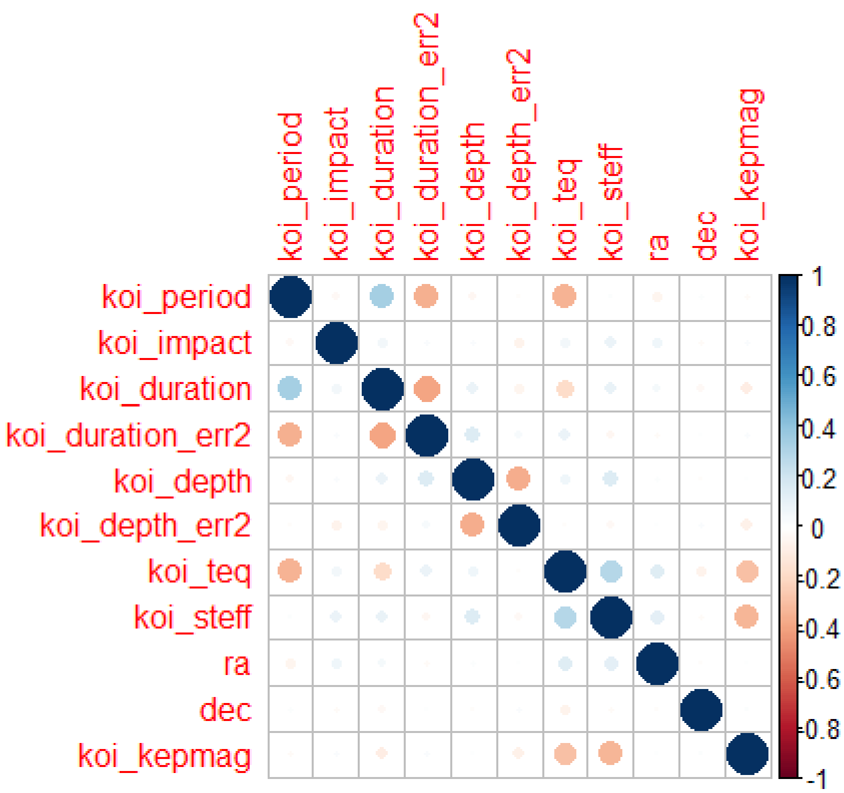
Hide

```
corrFrame2 = data.frame(cor(GLM_Data[, 2:dim(GLM_Data)[2]]))
corrplot(cor(GLM_Data[, 2:dim(GLM_Data)[2]])) #Many correlated variables remain
```



Hide

```
#KOI_period is collinear with koi_time0b
GLM_Data = subset(GLM_Data, select = -c(koi_time0bk))
#koi_period_err is collinear with koi_time0bk error
GLM_Data = subset(GLM_Data, select = -c(koi_time0bk_err2))
#Koi_period is collinear with koi_period_err2
GLM_Data = subset(GLM_Data, select = -c(koi_period_err2))
#koi_impact is collinear with koi_impact_err2
GLM_Data = subset(GLM_Data, select = -c(koi_impact_err2))
#koi_depth is collinear with koi_model_snr
GLM_Data = subset(GLM_Data, select = -c(koi_model_snr))
#koi impact is collinear with koi_prad and koi_prad_err
GLM_Data = subset(GLM_Data, select = -c(koi_prad, koi_prad_err2))
#koi_teq is collinear with KOI_insol, Koi_insol_err, koi_slogg, koi_srad, and koi_srad_e
rr
GLM_Data = subset(GLM_Data,
                  select = -c(koi_insol, koi_insol_err2, koi_slogg, koi_srad, koi_srad_e
rr2))
#koi_steff is collinear with koi_steff_err2 and koi_slogg_err2
GLM_Data = subset(GLM_Data, select = -c(koi_slogg_err2, koi_steff_err2))
corrplot(cor(GLM_Data[, 2:dim(GLM_Data)[2]]))
```



Hide

```
#All major multicollinearity has now been removed
```

Similarly to XGBoost, the cutoff threshold for prediction must be tuned. Our group decided to run multiple tests ranging from 0.1 to 1 to determine that the ideal threshold value of 0.5 should be used as that corresponded with the lowest average error.

Hide

```
threshold = 0.1
GLM_error = CreateErrorMatrix(NumFolds)
GLM_error_thresholds = rep(0, 10)
GLM_FP_thresholds = rep(0, 10)
GLM_FN_thresholds = rep(0, 10)
index = 1
while (threshold < 1) {
  #loop for threshold analysis
  GLM_error = CreateErrorMatrix(NumFolds)
  GLM_FP = CreateErrorMatrix(NumFolds)
  GLM_FN = CreateErrorMatrix(NumFolds)
  for (fold in 1:5) {
    #K-cross fold validation

    #training/testing set
    set.seed(fold)
    num_samples = dim(GLM_Data)[1]
    sampling.rate = 0.8
    training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
    trainingSet = subset(GLM_Data[training,])
    testing = setdiff(1:num_samples, training)
    testingSet = subset(GLM_Data[testing, ])
    defaultW = getOption("warn")
    options(warn = -1)
    #set up model
    LogisticReg = glm(koi_disposition ~ .,
                      data = trainingSet,
                      family = binomial(logit))


    options(warn = defaultW)
    predictions = predict(LogisticReg, testingSet, type = "response")
    predictedLabels = rep(0, sizeTestSet)
    predictedLabels = ifelse(predictions > threshold, 'FALSE POSITIVE', 'CONFIRMED') #th
is parameter is tuned

    GLMErrors = AllErrors(testingSet$koi_disposition, predictedLabels, sizeTestSet, 0)
    #determine error
    # error = sum(predictedLabels != testingSet$koi_disposition)
    #misclassificationRateLR = error / sizeTestSet
    #GLM_error[fold] = misclassificationRateLR
    GLM_error[fold] = GLMErrors[[1]]
    GLM_FP[fold] = GLMErrors[[2]]
    GLM_FN[fold] = GLMErrors[[3]]

  }

  GLM_error_thresholds[index] = mean(GLM_error)
  GLM_FP_thresholds[index] = mean(GLM_FP)
  GLM_FN_thresholds[index] = mean(GLM_FN)
  index = index + 1
  threshold = threshold + .1
```

```
}
order(GLM_error_thresholds) #Lowest error is threshold = .5
```

```
 [1]  5  4  6  3  7  2  8  9  1 10
```

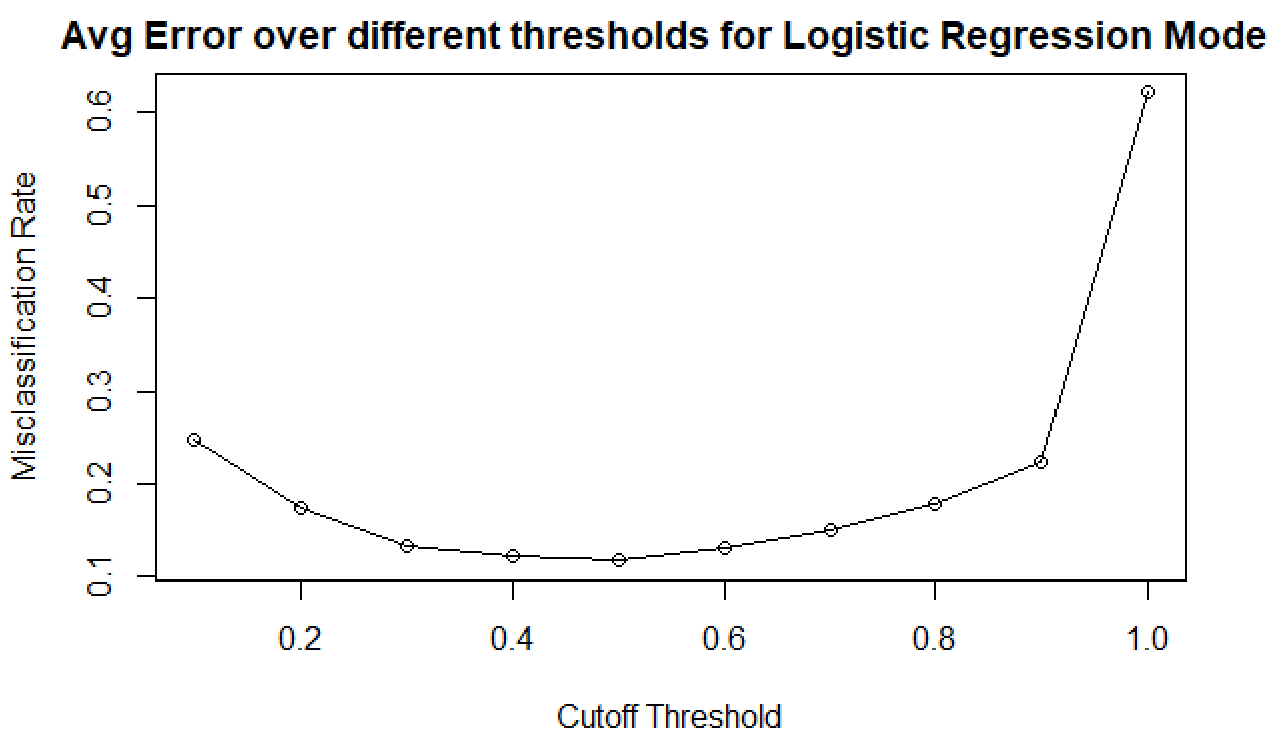
Hide

```
AvgErrorGLM = GLM_error_thresholds[order(GLM_error_thresholds)[1]]
AvgFP_GLM = GLM_FP_thresholds[order(GLM_error_thresholds)[1]]
AvgFN_GLM = GLM_FN_thresholds[order(GLM_error_thresholds)[1]]

paste("Logistic Regression Error: ", round(AvgErrorGLM*100,2),"%",sep="")
```

```
[1] "Logistic Regression Error: 11.73%"
```

Hide

```
plot(
  x = 1:10 / 10,
  y = GLM_error_thresholds,
  main = "Avg Error over different thresholds for Logistic Regression Model",
  xlab = "Cutoff Threshold",
  ylab = "Misclassification Rate"
)
lines(x = 1:10 / 10, y = GLM_error_thresholds)
```



Logistic Regression has an error rate of 11.7%, making it the worst model yet. Its error is best when threshold = .5.

3.2) Models using scaled data:

The models below all require normalization of the data to be effective. This is an important step as all features need to be in the same scale. If not, the features with larger scales would dominate the model causing it to be inaccurate. To do this, we used the “scale” function to normalize all the dependent variables. We also changed the independent variable, koi_disposition, to be binary

3.2.2) Neural Network: Original Variables

The Neural Network model is built through functions, “neurons” that are then organized into layers. It is an advanced model that is ideally suited for complex problems as it requires significant computational resources. In addition, it is quite difficult to understand afterwards given the complexity of the math within the model. Our group was able to see the significant use of computational resources as our computer was unable to run the model. For this reason, the group did not use K-fold cross-validation in order to lower the computational power required to run the model, but in real-life application K-fold cross-validation should still be done.

When choosing the number of neurons and hidden layers it is important to find the right balance between accuracy and over-fitting. Our group manually adjusted the number of neurons and hidden layers, testing variations such as 4&2, 5&1, 6, 3&1, etc, until we found that two neurons and one hidden layer allowed the model to converge. The next step in the model was to choose the threshold value for classification. This was similar to choosing the threshold as we did in Logistic Regression. The ideal threshold of 0.5 was found by plotting the average error for each threshold ranging from 0.1 to 1 as that corresponded with the lowest misclassification rate.

Hide

```
#Create testing and training set
set.seed(123)
scaled_data = data.frame(scale(labeled_final[, 2:36])) #normalize data
scaled_data$koi_disposition = ifelse(labeled_final$koi_disposition == "CONFIRMED", 1, 0)
num_samples = dim(scaled_data)[1]
sampling.rate = 0.8
training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
trainingSet.norm = subset(scaled_data[training,])
testing = setdiff(1:num_samples, training)
testingSet.norm = subset(scaled_data[testing, ])
sizeTestSet = dim(testingSet.norm)[1]

#set up variables for neural network. Since 36 variables put too much computational stra
in, dimensionality reduction needed to take place. We chose to take only the features, n
ot their errors, to reduce dimensionality.

koiDP.name = "koi_disposition"
numCols = dim(testingSet.norm)[2]
variable.names = colnames(testingSet.norm)[1:numCols - 1]
variable.names
```

```
 [1] "koi_period"        "koi_period_err1"   "koi_period_err2"
 [4] "koi_time0bk"       "koi_time0bk_err1"  "koi_time0bk_err2"
 [7] "koi_impact"        "koi_impact_err1"   "koi_impact_err2"
[10] "koi_duration"      "koi_duration_err1" "koi_duration_err2"
[13] "koi_depth"         "koi_depth_err1"    "koi_depth_err2"
[16] "koi_prad"          "koi_prad_err1"     "koi_prad_err2"
[19] "koi_teq"           "koi_insol"         "koi_insol_err1"
[22] "koi_insol_err2"    "koi_model_snr"     "koi_steff"
[25] "koi_steff_err1"    "koi_steff_err2"    "koi_slogg"
[28] "koi_slogg_err1"    "koi_slogg_err2"    "koi_srad"
[31] "koi_srad_err1"     "koi_srad_err2"     "ra"
[34] "dec"               "koi_kepmag"
```

Hide

```
for (i in 1:length(variable.names)) {
  error = grepl("_err", variable.names[i], fixed = TRUE)
  if (error) {
    variable.names[i] = NA
  }

}
variable.names = na.omit(variable.names)
variable.names = variable.names[1:15]
#use formulaic library to create formula
nn.form <-
  create.formula(outcome.name = koiDP.name,
                 input.names = variable.names)
nn.form #15 variables
```

```
$formula
koi_disposition ~ koi_period + koi_time0bk + koi_impact + koi_duration +
    koi_depth + koi_prad + koi_teq + koi_insol + koi_model_snr +
    koi_steff + koi_slogg + koi_srad + ra + dec + koi_kepmag
<environment: 0x00000155aac47e00>

$inclusion.table
```

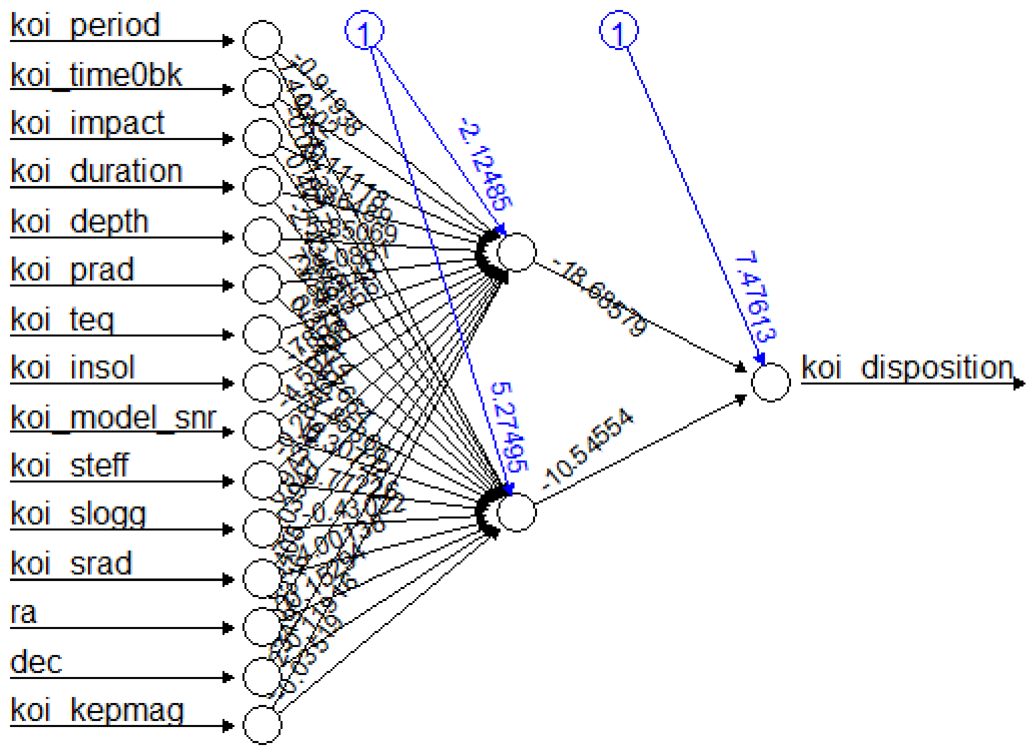
0 rows

```
$interactions.table
```

0 rows

Hide

```
#Fit neural network
nnModel1 = neuralnet(
  nn.form,
  data = trainingSet.norm,
  hidden = 2,
  linear.output = FALSE,
  act.fct = "logistic",
)
plot(nnModel1)
```



Hide

```
#Predict
predictedLabels = compute(nnModel1, testingSet.norm[, variable.names])

#tune threshold parameter
threshold = .1
NNErrors = CreateErrorMatrix(10)
NN_FPs = CreateErrorMatrix(10)
NN_FNs = CreateErrorMatrix(10)
index = 1
while (threshold <= 1) {
  results = data.frame(actual = testingSet.norm$koi_disposition,
                       prediction = predictedLabels$net.result)
  results$roundedPrediction = ifelse(results$prediction > threshold, 1, 0)
  #error = sum(results$actual != results$roundedPrediction)
  Errors = AllErrors(results$actual,results$roundedPrediction,sizeTestSet,1)
  NNErrors[index] = Errors[[1]]
  NN_FPs[index] = Errors[[2]]
  NN_FNs[index] = Errors[[3]]
  threshold = threshold + .1
  index = index + 1
}

order(NNErrors) #lowest misclass rate is at threshold = .5
```

```
 [1]  5  4  6  3  7  2  8  1  9 10
```

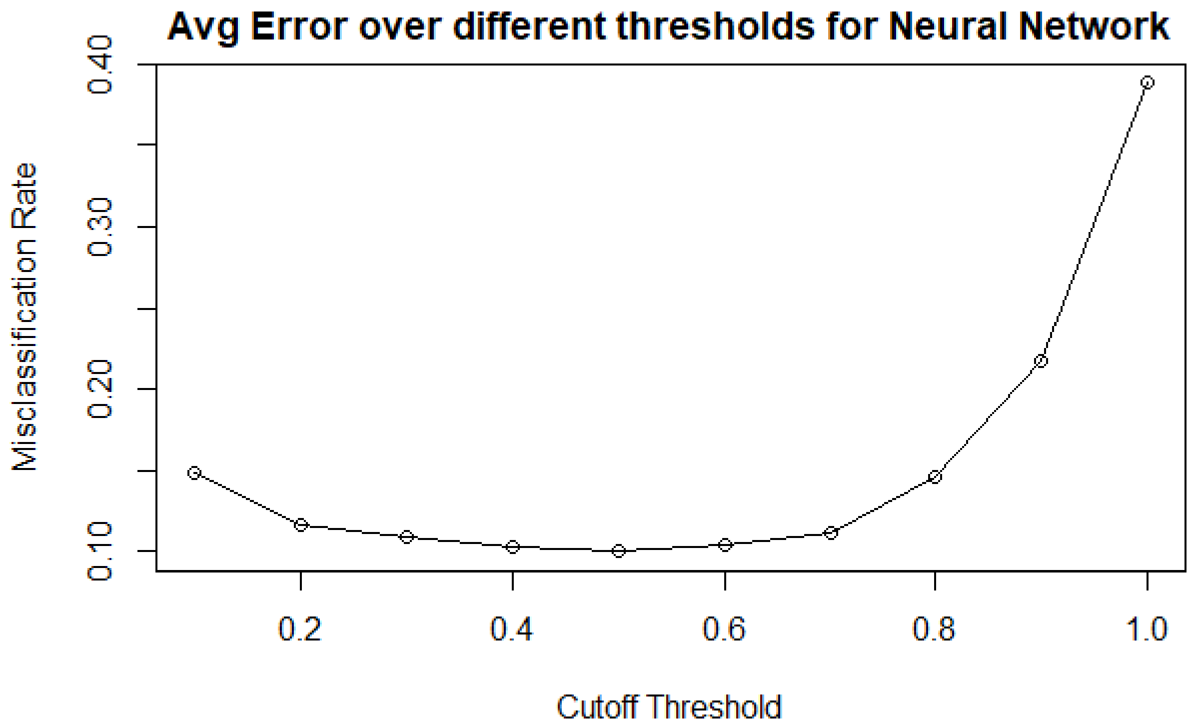
Hide

```
NeuralNetMisClassRate = NNErrors[order(NNErrors)[1]]
AvgFP_NN = NN_FPs[order(NNErrors)[1]]
AvgFN_NN = NN_FNs[order(NNErrors)[1]]
paste("Neural Net Misclassification Rate: ",round(100*NeuralNetMisClassRate,2),"%",sep=
"")
```

```
[1] "Neural Net Misclassification Rate: 10.02%"
```

Hide

```
plot(
  x = 1:10 / 10,
  y = NNErrors,
  main = "Avg Error over different thresholds for Neural Network",
  xlab = "Cutoff Threshold",
  ylab = "Misclassification Rate"
)
lines(x = 1:10 / 10, y = NNErrors)
```



In this run, NN had an error rate of 0.1002486, with an optimal cutoff threshold of .5. We will revisit the NN below, to look at ways to use PCA to capture variation while reducing necessary computational resources.

Neural Network: PCA w/15 Variables

Principal Component Analysis allows us to reduce dimensionality while capturing as much of the underlying variation in the data as possible. The first application of PCA we found was for Neural Networks, which are very computationally difficult. We decided to first use a PCA with 15 variables, the same number of variables as our original neural network. This would mean the same computational strain, but with features that are guaranteed to capture as much variation as possible with that number of variables.

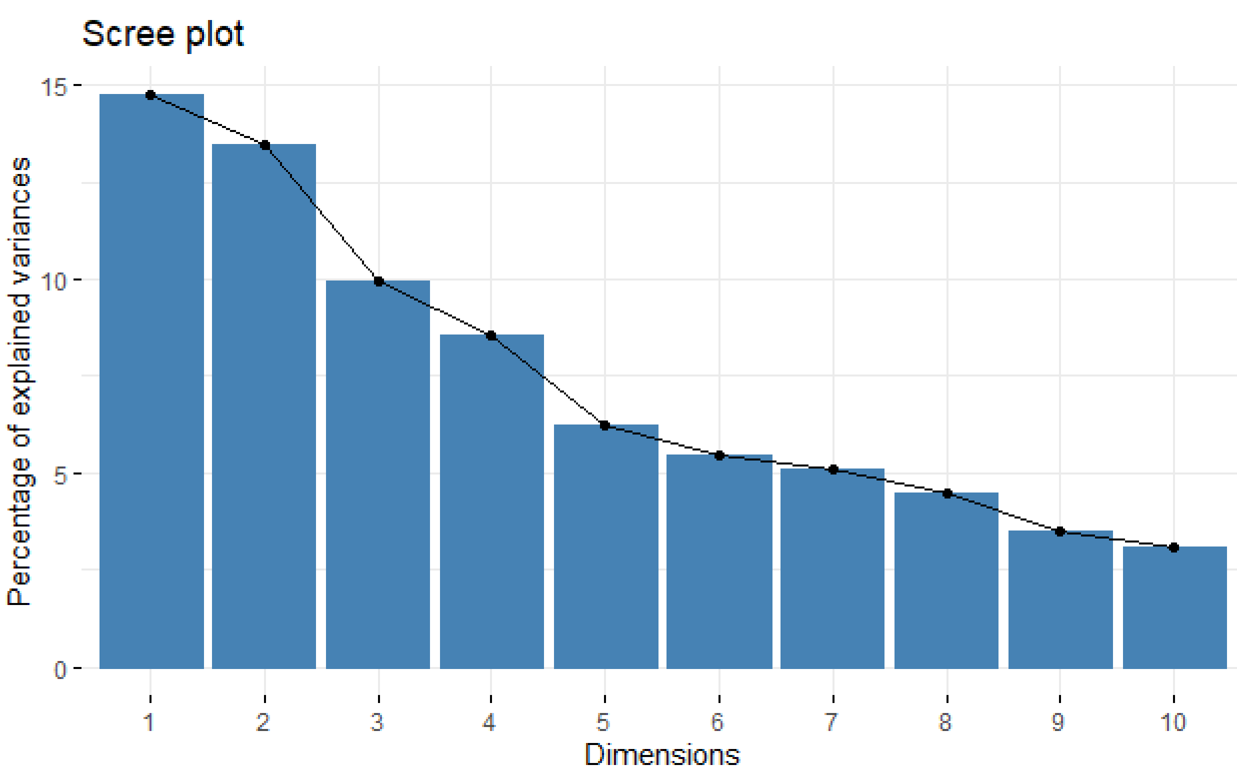
Hide

```
res.pca.exoplanets = prcomp(identifiers_removed[2:36], center = TRUE, scale = TRUE) #per
form PCA
summary(res.pca.exoplanets)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6
Standard deviation     2.2712 2.1730 1.86778 1.72894 1.47639 1.37931
Proportion of Variance 0.1474 0.1349 0.09967 0.08541 0.06228 0.05436
Cumulative Proportion  0.1474 0.2823 0.38196 0.46737 0.52965 0.58400
                           PC7     PC8     PC9    PC10    PC11    PC12
Standard deviation     1.33622 1.25447 1.10767 1.04188 1.01171 0.98148
Proportion of Variance 0.05101 0.04496 0.03506 0.03101 0.02924 0.02752
Cumulative Proportion  0.63502 0.67998 0.71503 0.74605 0.77529 0.80282
                          PC13    PC14    PC15    PC16    PC17    PC18
Standard deviation     0.95840 0.93448 0.88263 0.78194 0.77439 0.75012
Proportion of Variance 0.02624 0.02495 0.02226 0.01747 0.01713 0.01608
Cumulative Proportion  0.82906 0.85401 0.87627 0.89374 0.91087 0.92695
                          PC19    PC20    PC21    PC22    PC23    PC24
Standard deviation     0.67294 0.63205 0.58189 0.55938 0.54423 0.48547
Proportion of Variance 0.01294 0.01141 0.00967 0.00894 0.00846 0.00673
Cumulative Proportion  0.93989 0.95130 0.96097 0.96991 0.97838 0.98511
                          PC25    PC26    PC27    PC28    PC29    PC30
Standard deviation     0.44729 0.37145 0.31152 0.23244 0.12420 0.10799
Proportion of Variance 0.00572 0.00394 0.00277 0.00154 0.00044 0.00033
Cumulative Proportion  0.99083 0.99477 0.99754 0.99909 0.99953 0.99986
                          PC31      PC32      PC33      PC34      PC35
Standard deviation     0.07019 2.036e-15 5.773e-16 5.354e-16 3.988e-16
Proportion of Variance 0.00014 0.000e+00 0.000e+00 0.000e+00 0.000e+00
Cumulative Proportion  1.00000 1.000e+00 1.000e+00 1.000e+00 1.000e+00
```

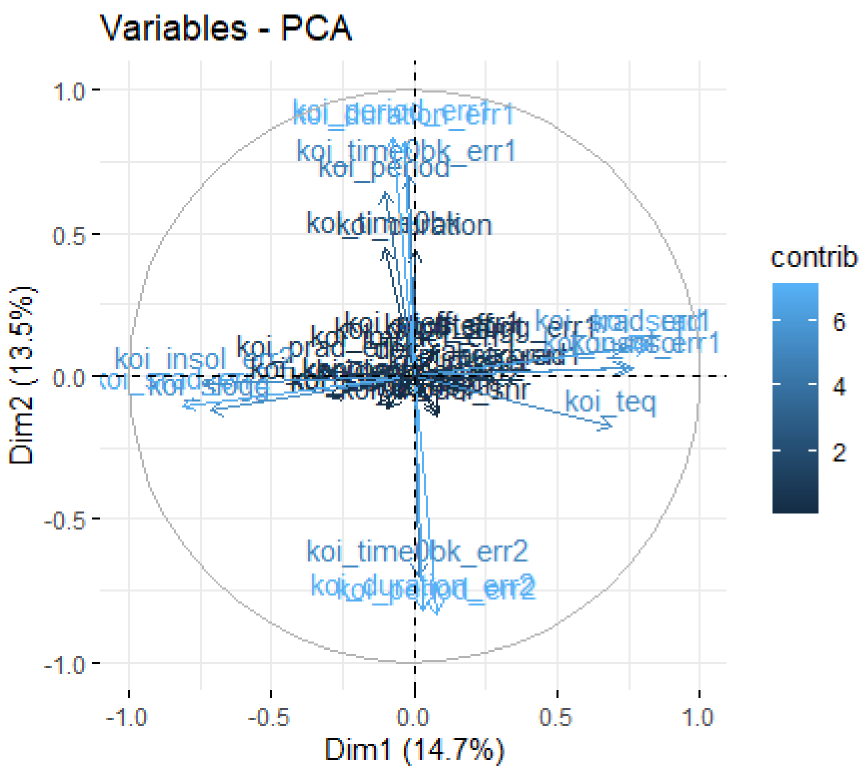
Hide

```
fviz_eig(res.pca.exoplanets)
```



Hide

```
fviz_pca_var(res.pca.exoplanets, col.var = "contrib")
```



Hide

```
PoV <-
  res.pca.exoplanets$sdev ^ 2 / sum(res.pca.exoplanets$sdev ^ 2) #get proportions of var
iance
numPcas = 15
sum(PoV[1:numPcas]) #This gives us 88% explanation of variance. This is the number of va
riable in the original neural network, so we are using this to get a better NN with the
 same number of variables
```

```
[1] 0.876268
```

Hide

```
newDataSet = data.frame(res.pca.exoplanets$x[, 1:numPcas])
newDataSet$label = identifiers_removed$koi_disposition


candidates_PCA = newDataSet[identifiers_removed$koi_disposition ==
                              "CANDIDATE", ] #separate out just the candidates
labeled_PCA = newDataSet[identifiers_removed$koi_disposition !=
                           "CANDIDATE", ]
labeled_PCA = droplevels(labeled_PCA)
candidates_PCA = droplevels(candidates_PCA)
```

While our PCA does not show any one dimension to account for an overwhelming amount of variance, it does clearly indicate that a significant portion of variance can be explained using significantly fewer that 36 principal components.

Rerunning Neural Network Model

Hide

```
set.seed(123)
NN_labeled_PCA = labeled_PCA[, 1:numPcas]
NN_labeled_PCA$koi_disposition = ifelse(labeled_PCA$label == "CONFIRMED", 1, 0)
summary(scaled_data)
```

```
  koi_period      koi_period_err1   koi_period_err2     koi_time0bk
 Min.   :-0.4049   Min.   :-0.1768   Min.   :-26.7574   Min.   :-0.64352
 1st Qu.:-0.3844   1st Qu.:-0.1762   1st Qu.:  0.1590   1st Qu.:-0.43439
 Median :-0.3277   Median :-0.1736   Median :  0.1736   Median :-0.37877
 Mean   : 0.0000   Mean   : 0.0000   Mean   :  0.0000   Mean   : 0.00000
 3rd Qu.:-0.1552   3rd Qu.:-0.1590   3rd Qu.:  0.1762   3rd Qu.: 0.07978
 Max.   :11.9778   Max.   :26.7574   Max.   :  0.1768   Max.   :23.11291
 koi_time0bk_err1   koi_time0bk_err2      koi_impact
 Min.   :-0.36586   Min.   :-31.75989   Min.   :-0.84878
 1st Qu.:-0.32745   1st Qu.:  0.00669   1st Qu.:-0.52855
 Median :-0.22068   Median :  0.22068   Median :-0.03305
 Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000
 3rd Qu.:-0.00669   3rd Qu.:  0.32745   3rd Qu.: 0.37679
 Max.   :31.75989   Max.   :  0.36586   Max.   :32.39182
 koi_impact_err1   koi_impact_err2     koi_duration      koi_duration_err1
 Min.   :-0.2207   Min.   :-37.0338   Min.   :-0.79008   Min.   :-0.46838
 1st Qu.:-0.2173   1st Qu.: -0.3289   1st Qu.:-0.45677   1st Qu.:-0.40374
 Median :-0.2034   Median :  0.2248   Median :-0.26098   Median :-0.28282
 Mean   : 0.0000   Mean   :  0.0000   Mean   : 0.00000   Mean   : 0.00000
 3rd Qu.:-0.1824   3rd Qu.:  0.4915   3rd Qu.: 0.07348   3rd Qu.:-0.02716
 Max.   : 8.2794   Max.   :  0.5591   Max.   :19.80016   Max.   :16.23186
 koi_duration_err2     koi_depth        koi_depth_err1
 Min.   :-16.23186   Min.   :-0.3491   Min.   :-0.25725
 1st Qu.:  0.02716   1st Qu.:-0.3471   1st Qu.:-0.22666
 Median :  0.28282   Median :-0.3432   Median :-0.19211
 Mean   :  0.00000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.:  0.40374   3rd Qu.:-0.3030   3rd Qu.:-0.09117
 Max.   :  0.46838   Max.   : 9.5013   Max.   :49.18710
 koi_depth_err2         koi_prad         koi_prad_err1
 Min.   :-49.18710   Min.   :-0.09248   Min.   :-0.06598
 1st Qu.:  0.09117   1st Qu.:-0.08857   1st Qu.:-0.06405
 Median :  0.19211   Median :-0.08497   Median :-0.06156
 Mean   :  0.00000   Mean   : 0.00000   Mean   : 0.00000
 3rd Qu.:  0.22666   3rd Qu.:-0.01473   3rd Qu.:-0.02127
 Max.   :  0.25725   Max.   :73.22898   Max.   :75.98152
 koi_prad_err2          koi_teq           koi_insol
 Min.   :-76.00748   Min.   :-1.2681   Min.   :-0.04894
 1st Qu.:  0.03814   1st Qu.:-0.6527   1st Qu.:-0.04872
 Median :  0.05676   Median :-0.2437   Median :-0.04762
 Mean   :  0.00000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.:  0.05834   3rd Qu.: 0.4011   3rd Qu.:-0.04094
 Max.   :  0.05933   Max.   :15.5272   Max.   :65.69789
 koi_insol_err1     koi_insol_err2      koi_model_snr
 Min.   :-0.06739   Min.   :-68.06196   Min.   :-0.3877
 1st Qu.:-0.06706   1st Qu.:  0.04559   1st Qu.:-0.3714
 Median :-0.06516   Median :  0.05053   Median :-0.3514
 Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.0000
 3rd Qu.:-0.05080   3rd Qu.:  0.05123   3rd Qu.:-0.2215
 Max.   :69.45869   Max.   :  0.05136   Max.   : 9.1486
  koi_steff         koi_steff_err1    koi_steff_err2       koi_slogg
 Min.   :-3.71885   Min.   :-3.0066   Min.   :-20.71031   Min.   :-9.8852
 1st Qu.:-0.48298   1st Qu.:-0.8168   1st Qu.: -0.45354   1st Qu.:-0.2227
```

```
 Median : 0.06731   Median : 0.2676   Median :  0.03832   Median : 0.2918
 Mean   : 0.00000   Mean   : 0.0000   Mean   :  0.00000   Mean   : 0.0000
 3rd Qu.: 0.49193   3rd Qu.: 0.6430   3rd Qu.:  0.65961   3rd Qu.: 0.5374
 Max.   :12.43001   Max.   :11.0913   Max.   :  2.09635   Max.   : 2.2474
 koi_slogg_err1    koi_slogg_err2       koi_srad         koi_srad_err1
 Min.   :-0.9095   Min.   :-9.1764   Min.   :-0.28040   Min.   :-0.35977
 1st Qu.:-0.5830   1st Qu.:-0.8374   1st Qu.:-0.15546   1st Qu.:-0.22791
 Median :-0.3753   Median : 0.1305   Median :-0.12554   Median :-0.10044
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000
 3rd Qu.: 0.2109   3rd Qu.: 0.6666   3rd Qu.:-0.06386   3rd Qu.: 0.00834
 Max.   :10.0123   Max.   : 1.9621   Max.   :31.37715   Max.   :36.00187
 koi_srad_err2             ra                 dec             koi_kepmag
 Min.   :-56.21894   Min.   :-2.58287   Min.   :-2.0123   Min.   :-5.4016
 1st Qu.:  0.07577   1st Qu.:-0.69660   1st Qu.:-0.8514   1st Qu.:-0.5886
 Median :  0.14915   Median : 0.04099   Median :-0.0363   Median : 0.1868
 Mean   :  0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.:  0.17252   3rd Qu.: 0.80352   3rd Qu.: 0.8106   3rd Qu.: 0.7616
 Max.   :  0.21002   Max.   : 2.00621   Max.   : 2.3631   Max.   : 3.5325
 koi_disposition
 Min.   :0.0000
 1st Qu.:0.0000
 Median :0.0000
 Mean   :0.3761
 3rd Qu.:1.0000
 Max.   :1.0000
```

Hide

```
head(scaled_data)
```

	koi_period <dbl>	koi_period_err1 <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0bk_err1 <dbl>	koi_time0
1	-0.2987063	-0.1712251	0.1712251	0.23459560	-0.2443950	0.2
2	0.2208034	-0.1269747	0.1269747	0.09358276	-0.1676082	0.1
3	-0.1783273	-0.1738000	0.1738000	0.32792873	-0.3335467	0.3
4	-0.3883287	-0.1767501	0.1767501	0.23053324	-0.3598575	0.3
5	-0.3792100	-0.1760470	0.1760470	0.25316557	-0.3025497	0.3
6	-0.2801336	-0.1727105	0.1727105	0.24623540	-0.2867407	0.2

6 rows | 1-7 of 36 columns

Hide

```
num_samples = dim(scaled_data)[1]
sampling.rate = 0.8

#create training/testing sets
training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
trainingSet.norm.PCA = subset(NN_labeled_PCA[training,])
testing = setdiff(1:num_samples, training)
testingSet.norm.PCA = subset(NN_labeled_PCA[testing, ])
sizeTestSet = dim(testingSet.norm)[1]
label.name = "label"
variable.names = rep(0, numPcas)
numCols = dim(testingSet.norm.PCA)[2]
variable.names = colnames(testingSet.norm.PCA)[1:numCols - 1]
variable.names
```

```
 [1] "PC1"  "PC2"  "PC3"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "PC9"  "PC10"
[11] "PC11" "PC12" "PC13" "PC14" "PC15"
```

Hide

```
nn.form <-
  create.formula(outcome.name = koiDP.name,
                 input.names = variable.names)
nn.form
```

```
$formula
koi_disposition ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 + PC7 + PC8 +
    PC9 + PC10 + PC11 + PC12 + PC13 + PC14 + PC15
<environment: 0x00000155ad9863e8>

$inclusion.table
```

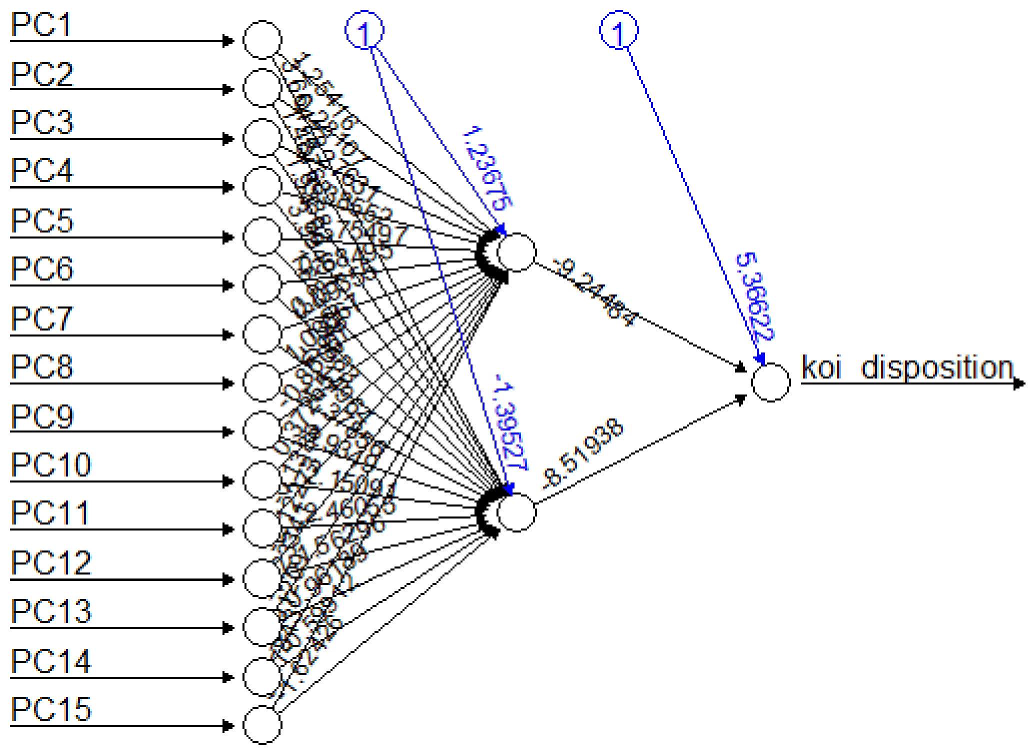
0 rows

```
$interactions.table
```

0 rows

Hide

```
#rerun neural network
nnModel2 = neuralnet(
  nn.form,
  data = trainingSet.norm.PCA,
  hidden = 2,
  linear.output = FALSE,
  act.fct = "logistic"
)
plot(nnModel2)
```



Hide

```
#Find results
predictedLabels = compute(nnModel2, testingSet.norm.PCA[, variable.names])

#Tune threshold
threshold = .1
NNErrors = rep(0, 10)
NN_FNs = rep(0,10)
NN_FPs = rep(0,10)
index = 1
while (threshold <= 1) {
  results = data.frame(actual = testingSet.norm.PCA$koi_disposition,
                       prediction = predictedLabels$net.result)
  results$roundedPrediction = ifelse(results$prediction > threshold, 1, 0)
  error = sum(results$actual != results$roundedPrediction)
  Errors = AllErrors(results$actual,results$roundedPrediction,sizeTestSet,1)
  NNErrors[index] = Errors[[1]]
  NN_FPs[index] = Errors[[2]]
  NN_FNs[index] = Errors[[3]]

  threshold = threshold + .1
  index = index + 1
}
order(NNErrors) #lowest misclassification rate is at threshold = .4
```

```
 [1]  4  6  5  3  7  2  8  1  9 10
```

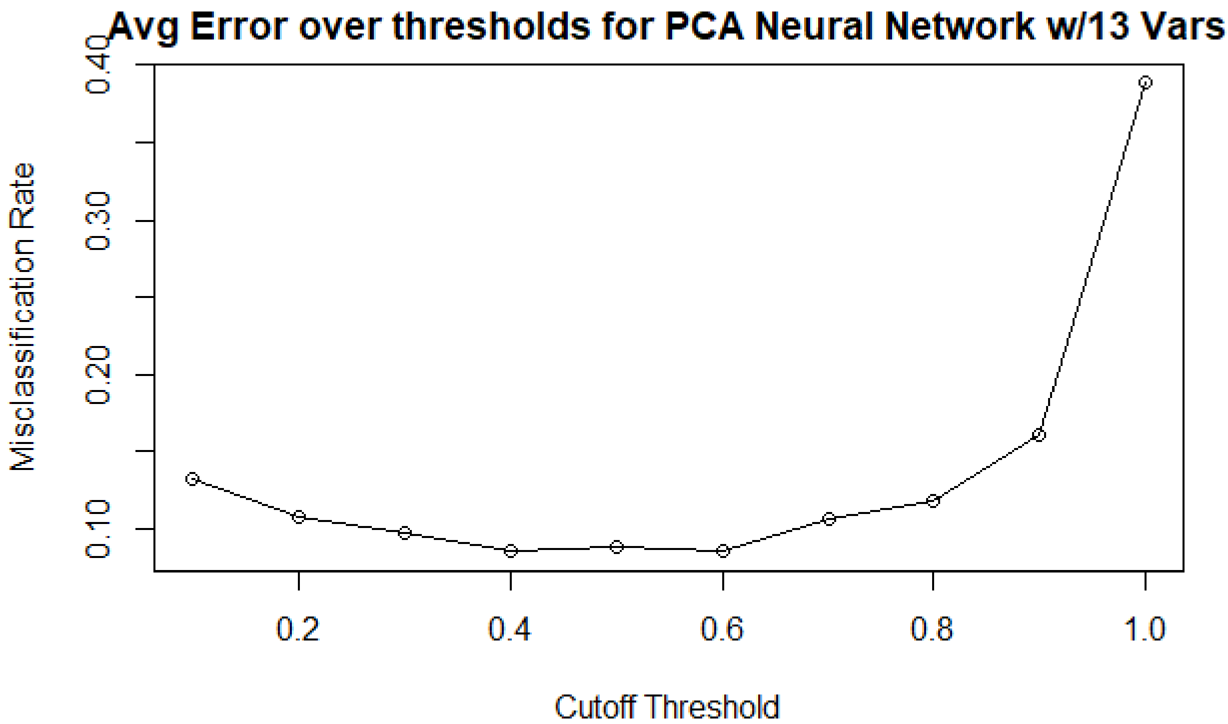
Hide

```
PCA_15_v_NeuralNetMisClassRate = NNErrors[order(NNErrors)[1]]
Avg_FP_PCA15NN = NN_FPs[order(NNErrors)[1]]
Avg_FN_PCA15NN = NN_FNs[order(NNErrors)[1]]
paste("Neural Net (PCA, 15 var) Misclassification Rate: ",round(100*PCA_15_v_NeuralNetMi
sClassRate,2),"%",sep="")
```

```
[1] "Neural Net (PCA, 15 var) Misclassification Rate: 8.53%"
```

Hide

```
plot(
  x = 1:10 / 10,
  y = NNErrors,
  main = "Avg Error over thresholds for PCA Neural Network w/13 Vars",
  xlab = "Cutoff Threshold",
  ylab = "Misclassification Rate"
)
lines(x = 1:10 / 10, y = NNErrors)
```



We get abetter misclassification rate from the PCA version of NN using 13 variables of 8.53%, which is 1.49% less than our last Neural Network, and an optimal cutoff threshold of .5.

3.2.3) Neural Network: PCA w/20 Variables

Lastly (for NNs), we decided to see how many variables could capture 95% of variation. We found that 20 variables was sufficient; this means that 16 of our 36 variables, or 44.44% represent only 5% of variation.

Hide

```
#Last Neural Network, PCA, with 95% of variation explained
numVars = (dim((labeled_final))[2])
for (i in 1:(numVars)) {
  if (sum(PoV[1:i]) >= .95) {
    numPcas = i
    break

  }
}

sum(PoV[1:numPcas])
```

```
[1] 0.9513002
```

Hide

```
newDataSet = data.frame(res.pca.exoplanets$x[, 1:numPcas])

newDataSet$label = identifiers_removed$koi_disposition

candidates_PCA = newDataSet[identifiers_removed$koi_disposition ==
                              "CANDIDATE",] #separate out just the candidates
labeled_PCA = newDataSet[identifiers_removed$koi_disposition !=
                           "CANDIDATE",]
labeled_PCA = droplevels(labeled_PCA)
candidates_PCA = droplevels(candidates_PCA)
```

Hide

```
set.seed(123)
NN_labeled_PCA = labeled_PCA[, 1:numPcas]
NN_labeled_PCA$koi_disposition = ifelse(labeled_PCA$label == "CONFIRMED", 1, 0)
head(scaled_data)
```

	koi_period <dbl>	koi_period_err1 <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0bk_err1 <dbl>	koi_time0
1	-0.2987063	-0.1712251	0.1712251	0.23459560	-0.2443950	0.2
2	0.2208034	-0.1269747	0.1269747	0.09358276	-0.1676082	0.1
3	-0.1783273	-0.1738000	0.1738000	0.32792873	-0.3335467	0.3
4	-0.3883287	-0.1767501	0.1767501	0.23053324	-0.3598575	0.3
5	-0.3792100	-0.1760470	0.1760470	0.25316557	-0.3025497	0.3
6	-0.2801336	-0.1727105	0.1727105	0.24623540	-0.2867407	0.2

6 rows | 1-7 of 36 columns

Hide

```
#testing and training sets
num_samples = dim(scaled_data)[1]
sampling.rate = 0.8
training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
trainingSet.norm.PCA = subset(NN_labeled_PCA[training,])
testing = setdiff(1:num_samples, training)
testingSet.norm.PCA = subset(NN_labeled_PCA[testing, ])
sizeTestSet = dim(testingSet.norm)[1]
label.name = "label"
variable.names = rep(0, numPcas)

numCols = dim(testingSet.norm.PCA)[2]
variable.names = colnames(testingSet.norm.PCA)[1:numCols - 1]

nn.form <-
  create.formula(outcome.name = koiDP.name,
                 input.names = variable.names)
nn.form
```

```
$formula
koi_disposition ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 + PC7 + PC8 +
    PC9 + PC10 + PC11 + PC12 + PC13 + PC14 + PC15 + PC16 + PC17 +
    PC18 + PC19 + PC20
<environment: 0x00000155bfa33960>

$inclusion.table
```

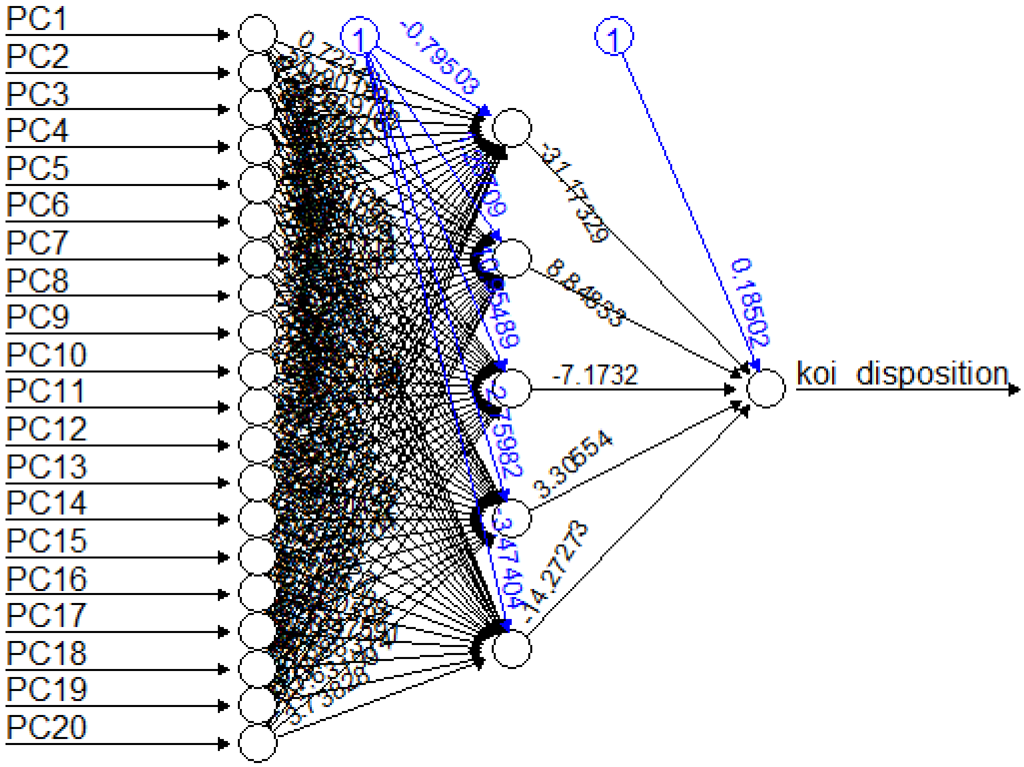
0 rows

```
$interactions.table
```

0 rows

Hide

```
library(neuralnet)
nnModel3 = neuralnet( #parameters tuned manually to ensure this stays computationally tr
actable
  nn.form,
  data = trainingSet.norm.PCA,
  hidden = c(5),
  linear.output = FALSE,
  act.fct = "logistic",
  stepmax=17000,
  threshold = 0.1
)

plot(nnModel3)
```



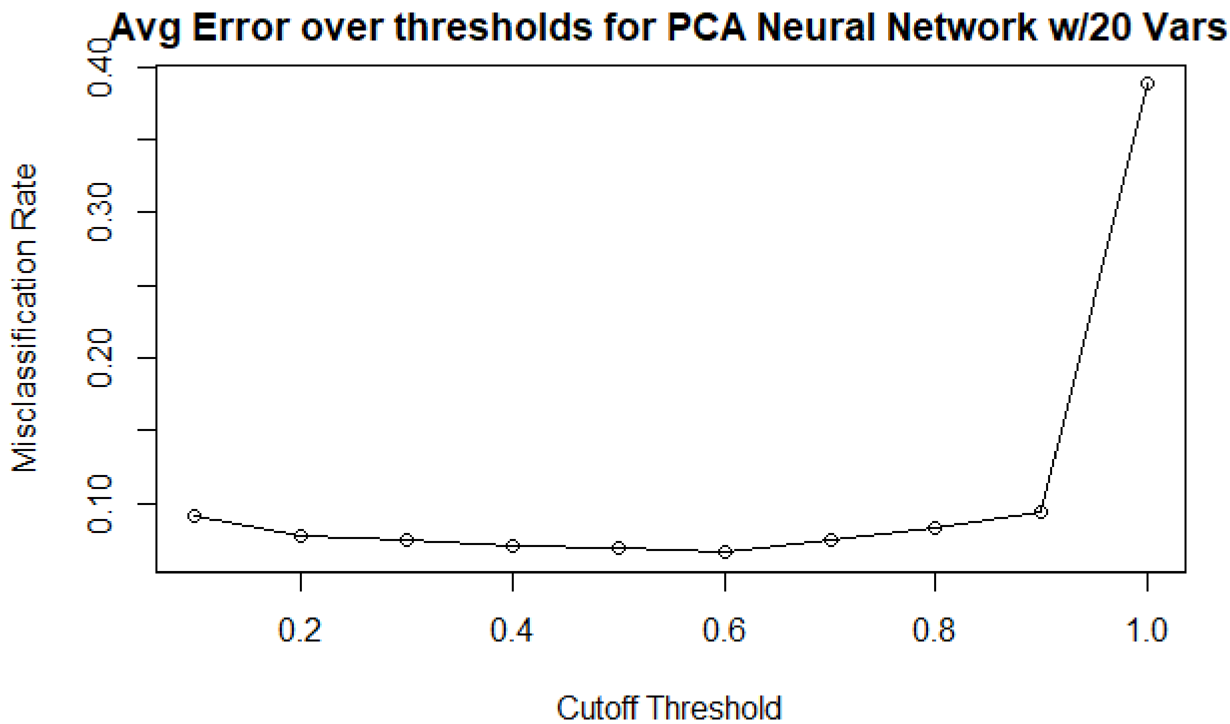
Hide

```
#Make prediction
predictedLabels = compute(nnModel3, testingSet.norm.PCA[, variable.names])

index = 1
threshold = .1
NNErrors = rep(0, 10)
NN_FPs = rep(0,10)
NN_FNs = rep(0,10)
while (threshold <= 1) { #tune threshold parameter
  results = data.frame(actual = testingSet.norm.PCA$koi_disposition,
                       prediction = predictedLabels$net.result)
  results$roundedPrediction = ifelse(results$prediction > threshold, 1, 0)
  error = sum(results$actual != results$roundedPrediction)
   Errors = AllErrors(results$actual,results$roundedPrediction,sizeTestSet,1)
  NNErrors[index] = Errors[[1]]
  NN_FPs[index] = Errors[[2]]
  NN_FNs[index] = Errors[[3]]
  threshold = threshold + .1
  index = index + 1
}
NNErrors #lowest error is with threshold = .6
```

```
 [1] 0.09196355 0.07705054 0.07456504 0.07125104 0.06876553 0.06628003
 [7] 0.07539354 0.08367854 0.09444905 0.38856669
```

Hide

```
PCA_20_v_NeuralNetMisClassRate = NNErrors[order(NNErrors)[1]]
Avg_FP_PCA20NN = NN_FPs[order(NNErrors)[1]]
Avg_FN_PCA20NN = NN_FNs[order(NNErrors)[1]]
plot(
  x = 1:10 / 10,
  y = NNErrors,
  main = "Avg Error over thresholds for PCA Neural Network w/20 Vars",
  xlab = "Cutoff Threshold",
  ylab = "Misclassification Rate"
)
lines(x = 1:10 / 10, y = NNErrors)
```



As expected, this version has the lowest misclassification rate of 6.63%, which is 1.91% lower than the 13 variable PCA NN, and 3.4% lower than the original neural network. As you can see, however, we are reaching a point of diminishing returns; adding ~15% more variation only decreased the error rate by 1.91%

3.2.4) K-Nearest Neighbours

kNN works by computing the euclidean distances of the test features to training data points, known as “neighbours”. This model requires pre-processing because the distances from each data point must be in the same scale, therefore we first normalized the training and testing features. This model has an input parameter, k, which represents the number of neighbours considered. Both too-small and too-large values of k can be detrimental. To tune this parameter, we tested several values of k in a loop, selecting the k which resulted in the lowest misclassification rate.

Hide

```
numKs = 10
k_errors = rep(0, numKs)
k_FPs = rep(0,numKs)
k_FNs = rep(0,numKs)
for (ki in 1:numKs) {
  #tune k parameter
  avgErrors_fold = CreateErrorMatrix(NumFolds)
  avgFP_fold = CreateErrorMatrix(NumFolds)
  avgFN_fold = CreateErrorMatrix(NumFolds)
  for (fold in 1:NumFolds) {
    #k-fold cross validation
    set.seed(fold)
    #make normalized training and testing sets
    scaled_data = data.frame(scale(labeled_final[, 2:36]))
    scaled_data$koi_disposition = ifelse(labeled_final$koi_disposition == "CONFIRMED", 1
, 0)
    summary(scaled_data)
    head(scaled_data)
    num_samples = dim(scaled_data)[1]
    sampling.rate = 0.8
    training = sample(1:num_samples, sampling.rate * num_samples, replace = FALSE)
    trainingSet.norm = subset(scaled_data[training,])
    testing = setdiff(1:num_samples, training)
    testingSet.norm = subset(scaled_data[testing, ])
    sizeTestSet = dim(testingSet.norm)[1]

    trainingfeatures = subset(trainingSet.norm, select = c(-koi_disposition))
    traininglabels = trainingSet.norm$koi_disposition
    testingfeatures = subset(testingSet.norm, select = c(-koi_disposition))
    testinglabels = testingSet.norm$koi_disposition


    #fit model and predict
    predictedLabels = knn(trainingfeatures, testingfeatures, traininglabels, k =
                            ki)
    #determine error
    error = sum(predictedLabels != testingSet.norm$koi_disposition)
    Errors = AllErrors(testingSet.norm$koi_disposition,
                       predictedLabels,
                       sizeTestSet,
                       1)
    misclassification_rate = error / sizeTestSet
    avgErrors_fold[fold] = Errors[[1]]
    avgFP_fold[fold] = Errors[[2]]
    avgFN_fold[fold] = Errors[[3]]

  }

  k_errors[ki] = mean(avgErrors_fold)
  k_FPs[ki] = mean(avgFP_fold)
  k_FNs[ki] = mean(avgFN_fold)
```

```
}
print(order(k_errors))
```

```
 [1]  4  6  8  7 10  5  9  3  2  1
```

Hide

```
#The lowest average error (this run) is from the model with k = 4.
AvgError_best_knn = k_errors[order(k_errors)[1]]
AvgFP_knn = k_FPs[order(k_errors)[1]]
AvgFN_knn = k_FNs[order(k_errors)[1]]
```

KNN produces an error of 9.08%, with an optimal k-value of 4

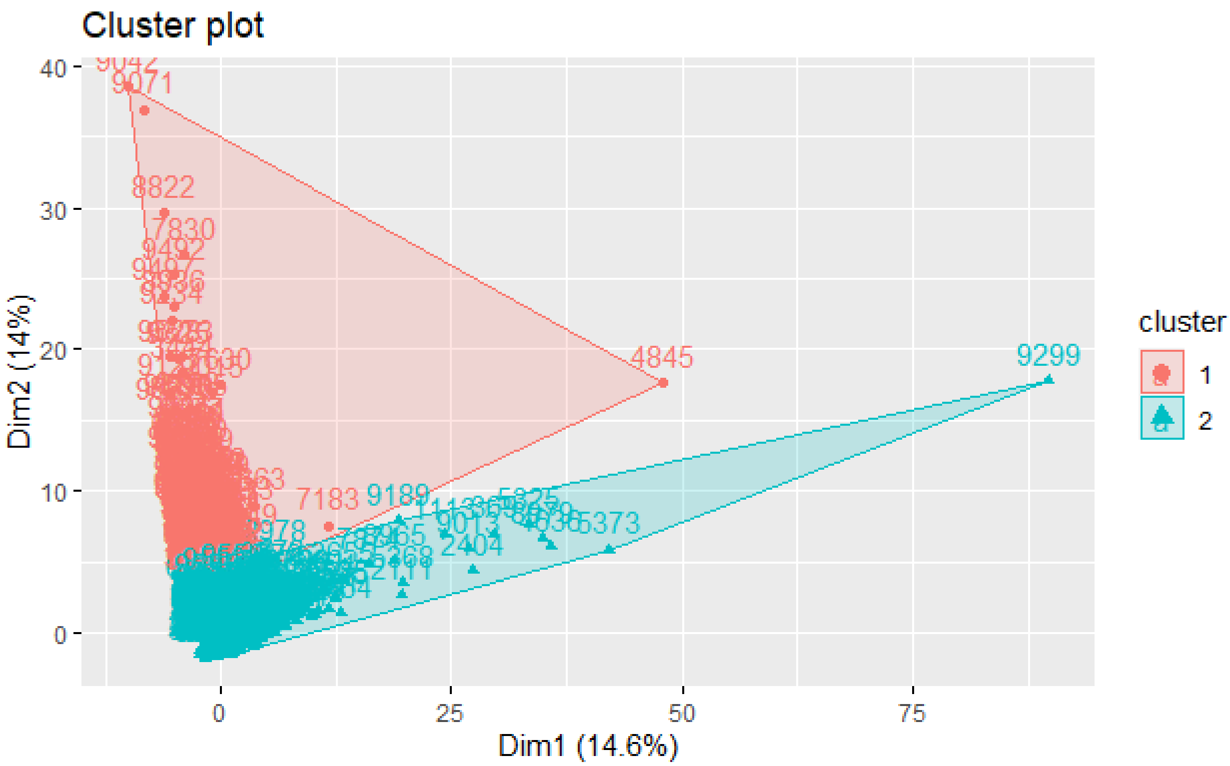
3.2.5) K-Means Clustering

Clustering is an unsupervised model which uses randomly generated centroids and assigns every point to a centroid based on euclidean distance. The model then iterates to find the centroid locations which minimize the distances to the data points in the clusters. Since this model also requires calculation of euclidean distance, the normalized data set was also used here.

Since our data set was labelled, the supervised models will likely yield a better misclassification rate than k-Means clustering. We include clustering in case the algorithm found unforeseen relationships in the unlabelled data features.

Hide

```
#scale full data set.
set.seed(123)
scaled_data = data.frame(scale(labeled_final[, 2:36]))
scaled_data$koi_disposition = ifelse(labeled_final$koi_disposition == "CONFIRMED", 2, 1)
num_samples = dim(scaled_data)[1]
features = subset(scaled_data, select = c(-koi_disposition))
#fit the model
kclustering = kmeans(features, centers = 2, nstart = 25)#we used two clusters, one for C
ONFIRMED, and one for FALSE POSITIVE. We will then attempt to match clusters to classifi
cations; there are two potential configurations (1 = confirmed and 2 = confirmed); which
ever one results in a misclassification rate of less than 50% will be the accepted confi
guration.
#visualize the clusters
fviz_cluster(kclustering, data = features)
```



Hide

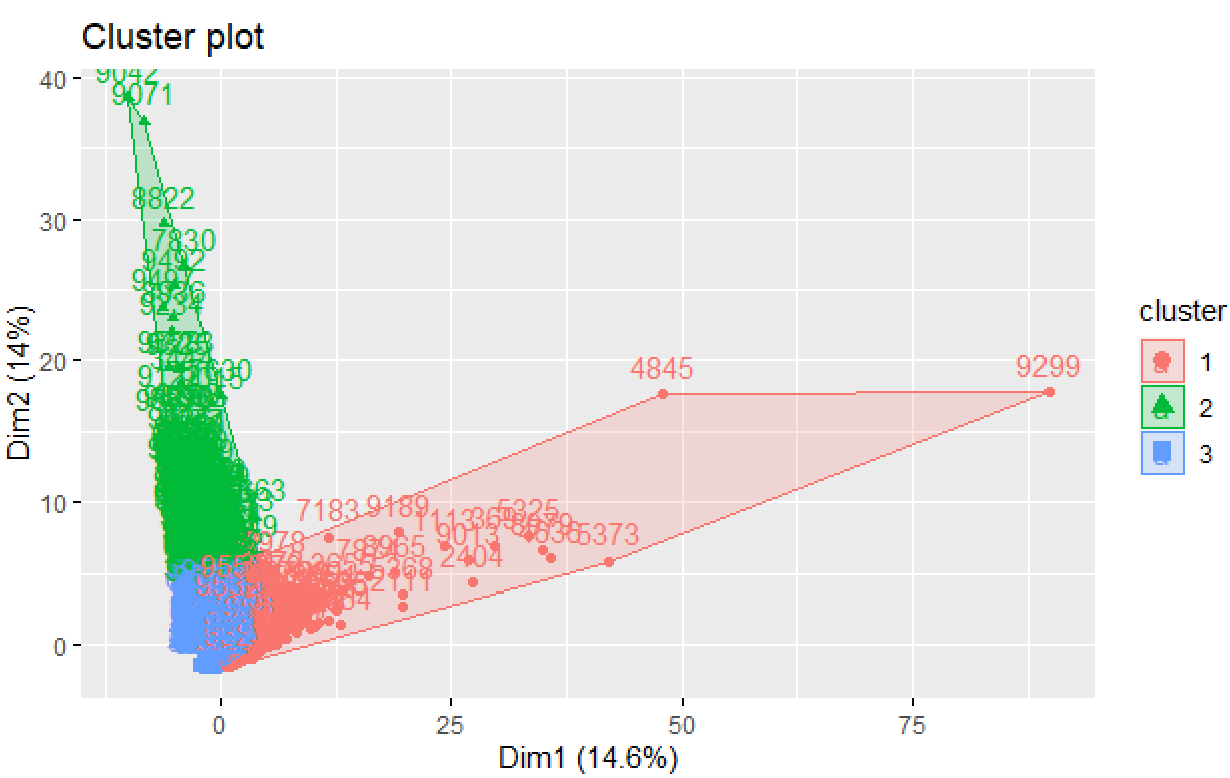
```
error = sum(scaled_data$koi_disposition != kclustering$cluster)
misclassification_rate = error / dim(scaled_data)[1]


isWrong = (scaled_data$koi_disposition != kclustering$cluster)
isRight = (scaled_data$koi_disposition == kclustering$cluster)
IsC = (kclustering$cluster == 2)
IsF = (kclustering$cluster == 1)
FalsePositives = sum(isWrong & IsC)
FalseNegatives = sum(isWrong & IsF)
TruePositives = sum(isRight & IsC)
TrueNegatives = sum(isRight & IsF)
Clustering_FP_Rate = (FalsePositives / (FalsePositives + TrueNegatives))#define FP rate
 as FP/(FP+TN)
Clustering_FN_Rate = (FalseNegatives / (FalseNegatives + TruePositives))#define FN rate
 as FN/(FN+TP)


AvgErrorClustering = misclassification_rate
if (AvgErrorClustering > .5) {
  AvgErrorClustering = 1 - AvgErrorClustering#since clustering does not know which clust
er is CONFIRMED and which is FALSE POSITIVE, they can be flipped
  Clustering_FP_Rate = 1 - Clustering_FP_Rate
  Clustering_FN_Rate = 1 - Clustering_FN_Rate

}
summary(factor(kclustering$cluster))
```

```
  1    2
275 5756
```

Unsurprisingly, clustering has the largest error, of 41.83%; given this is an unsupervised method and our data has labels. Clustering seems to have put nearly all values in category “2,” which has resulted in a misclassification rate that is roughly the same as the proportions of CONFIRMED and FALSE POSITIVES in the original data set. The fact that clustering is almost only predicting one value can be seen in its terrible false negative rate of 99.56%. One hypothesis for why this might be is because there are certain values that are different enough from the rest of the data set that they are not clearly negative or positive results, and these are being clustered together. To test this, I will try again with 3 centers.

Hide

```
#scale full data set.
set.seed(123)
scaled_data = data.frame(scale(labeled_final[, 2:36]))
scaled_data$koi_disposition = ifelse(labeled_final$koi_disposition == "CONFIRMED", 2, 1)
num_samples = dim(scaled_data)[1]
features = subset(scaled_data, select = c(-koi_disposition))
#fit the model
kclustering = kmeans(features, centers = 3, nstart = 25)#we used two clusters, one for C
ONFIRMED, and one for FALSE POSITIVE. We will then attempt to match clusters to classifi
cations; there are two potential configurations (1 = confirmed and 2 = confirmed); which
ever one results in a misclassification rate of less than 50% will be the accepted confi
guration.
#visualize the clusters
fviz_cluster(kclustering, data = features)
```



Hide

```
summary(factor(kclustering$cluster))
```

```
   1    2    3 
1126  245 4660 
```

With three clusters, we are still getting category 2 with very few values, but have more robust categories 1 and 3. I will therefore labeled category 2 “inconclusive” and get a misclassification rate using categories 1 and 3.

Hide

```
scaled_data$koi_disposition = ifelse(labeled_final$koi_disposition == "CONFIRMED", 3, 1)
error = sum(scaled_data$koi_disposition != kclustering$cluster)
misclassification_rate = error / dim(scaled_data)[1]
isWrong = (scaled_data$koi_disposition != kclustering$cluster)
isRight = (scaled_data$koi_disposition == kclustering$cluster)
IsC = (kclustering$cluster == 3)
IsF = (kclustering$cluster == 1)
FalsePositives = sum(isWrong & IsC)
FalseNegatives = sum(isWrong & IsF)
TruePositives = sum(isRight & IsC)
TrueNegatives = sum(isRight & IsF)
Clustering_FP_2_Rate = (FalsePositives / (FalsePositives + TrueNegatives))#define FP rat
e as FP/(FP+TN)
Clustering_FN_2_Rate = (FalseNegatives / (FalseNegatives + TruePositives))#define FN rat
e as FN/(FN+TP)

AvgErrorClustering_2 = misclassification_rate
if (AvgErrorClustering_2 > .5) {
  AvgErrorClustering_2 = 1 - AvgErrorClustering_2#since clustering does not know which c
luster is CONFIRMED and which is FALSE POSITIVE, they can be flipped
  Clustering_FP_2_Rate = 1 - Clustering_FP_2_Rate
  Clustering_FN_2_Rate = 1 - Clustering_FN_2_Rate

}

paste("Clustering 2 Error Rate: ", round(AvgErrorClustering_2*100,2),"%",sep = "")
```

```
[1] "Clustering 2 Error Rate: 45.18%"
```

This produces an error rate barely better than that of the naive model, but with more sensible false positive/false negative rates of 69.2% and 1.77%. Although this is not better than any of our other models, it illustrates an interesting application of unsupervised learning to find categories that might not be immediately apparent in the labeled data.

4) Select Best Model

Hide

```
error_output = data.frame(
  "Model" = c(
    "Decision Tree",
    "GLM",
    "Random Forest",
    "SVM",
    "Neural Net",
    "KNN",
    "Clustering",
    "PCA 15 Var NN",
    "PCA 20 Var NN",
    "XGBoost"
  ),
  "Misclassification Rate" = c(
    AvgErrorDT,
    AvgErrorGLM,
    AvgErrorRF,
    AvgErrorSVM,
    NeuralNetMisClassRate,
    AvgError_best_knn,
    AvgErrorClustering_2,
    PCA_15_v_NeuralNetMisClassRate,
    PCA_20_v_NeuralNetMisClassRate,
    AvgErrorXGB
  ), "False Positive Rate" = c(
    AvgFP_DT,
    AvgFP_GLM,
    AvgFP_RF,
    AvgFP_SVM,
    AvgFP_NN,
    AvgFP_knn,
    Clustering_FP_2_Rate,
    Avg_FP_PCA15NN,
    Avg_FP_PCA20NN,
    AvgFP_XGB
  ), "False Negative Rate" = c(
    AvgFN_DT,
    AvgFN_GLM,
    AvgFN_RF,
    AvgFN_SVM,
    AvgFN_NN,
    AvgFN_knn,
    Clustering_FN_2_Rate,
    Avg_FN_PCA15NN,
    Avg_FN_PCA20NN,
    AvgFN_XGB
  )
)
print(error_output)
```

Model <chr>	Misclassification.Rate <dbl>	False.Positive.Rate <dbl>	False.Negative.Rate <dbl>
Decision Tree	0.11582436	0.12863903	0.09471457
GLM	0.11731566	0.10041469	0.14515929
Random Forest	0.06661143	0.04633321	0.10000657
SVM	0.08699254	0.08095670	0.09689713
Neural Net	0.10024855	0.10840108	0.08742004
KNN	0.09080365	0.07241510	0.12097613
Clustering	0.45183220	0.69200227	0.01769912
PCA 15 Var NN	0.08533554	0.08807588	0.08102345
PCA 20 Var NN	0.06628003	0.05149051	0.08955224
XGBoost	0.06081193	0.10307018	0.04660453

1-10 of 10 rows

All models beat the naive misclassification rate of 46.73%, with all except clustering beating it by a significant margin. XGBoost has the lowest misclassification rate; although it has a slightly higher False Positive rate than some other model, it still has the best overall accuracy. We will remake this model using the full dataset.

#5 Make predictions

Remake the XGBoost model using the full dataset:

Hide

```
xgData = data.matrix(labeled_final)
 xgData[, 1] = ifelse(xgData[, 1] == 2, 1, 0)

 xgBoostModelFinal = xgboost(
   data = xgData[, 2:36],
   label = xgData[, 1],
   max.depth = 6,
   eta = .22,
   nrounds = 100,
   verbose = 0,
   objective = "binary:logistic",
   eval_metric="error"
 )
 xgPredict = data.matrix(candidates_final)
 xgPredict[, 1] = ifelse(xgPredict[, 1] == 2, 1, 0)
```

Predict labels of candidates dataset, and write it to file

Hide

```
 #make predictions
BoostPredictions = predict(xgBoostModelFinal, data.matrix(candidates_final)[, 2:36])
BoostPredictionsRounded = ifelse(BoostPredictions > .4, 1, 0)
predictedLabels = ifelse(BoostPredictionsRounded == 1, "FALSE POSITIVE", "CONFIRMED")
candidates_final$koi_disposition = predictedLabels
head(candidates_final)
```

	koi_disposition <chr>	koi_period <dbl>	koi_period_err1 <dbl>	koi_period_err2 <dbl>	koi_time0bk <dbl>	koi_time0l
38	CONFIRMED	4.959319	5.150e-07	-5.150e-07	172.2585	8
59	FALSE POSITIVE	40.419504	1.139e-04	-1.139e-04	173.5647	2
63	FALSE POSITIVE	7.240661	1.617e-05	-1.617e-05	137.7554	2
64	FALSE POSITIVE	3.435916	4.729e-05	-4.729e-05	132.6624	1
73	FALSE POSITIVE	1.626630	1.015e-06	-1.015e-06	169.8202	4
85	FALSE POSITIVE	10.181584	6.188e-06	-6.188e-06	177.1419	4

6 rows | 1-7 of 36 columns

Hide

```
write.csv(candidates_final, "labeledCandidates.csv")

numConfirmed = sum(candidates_final[,1]=="CONFIRMED")
numFalse = sum(candidates_final[,1]=="FALSE POSITIVE")
Proportions = data.frame(label = c("CONFIRMED","FALSE POSITIVE"), number = c(numConfirme
d,numFalse))
bp = ggplot(Proportions,aes(x = "",y=number,fill=label))+geom_bar(width = 1,stat = "iden
tity")
pie = bp+coord_polar("y", start = 0) + ggtitle("Proportions of final predictions that ar
e CONFIRMED or FALSE POSITIVE")
```

As shown in the pie above, our final model is predicting more non-planets than confirmed planets. Given that our XGBoost model has a higher False Positive than False Negative rate, it is likely that some of these CONFIRMEDs are in fact incorrect, meaning that the there are likely slightly more FALSEs and slightly fewer CONFIRMED than predicted.

In conclusion, this model can be used the identify new planets in the future. Given more computational resources, the candidate models for neural networks would likely be able to be parameter tuned more, and run with more input variables. Given our limited computational power, however, we believe this to be a strong model for predicting whether a given stellar observation is an exoplanet.