Kaggle : Titanic Machine Learning Project with Ensemble Learning

Jordan Rinder

August 14, 2015

About This Project

This project contains several *Machine Learning* algorithms for the Titanic Kaggle project. These algorithms are by no means the most efficient way to solve this problem, but provide a great base for future Machine Learning Projects.

Data

These links contain the training data and testing data sets. I will combine the data and create new columns in this combined data set so it will make it easier when I split the data back up into the respective training and testing sets.

Basic Exploratory Data Analysis

Basic Exploratory Data Analysis must be done in order to find any lurking variables that might be highly correlated with one another. The key is to have all of the predictors as uncorrelated to one another as possible. The exploratory data analysis phase of any successful predictive algorithm is always the most crucial. This is because exploratory data analysis allows us to form relations in order to simplify the data. One goal is to simplify the data so that it will be interpretable, but still have the same accuracy.

During this phase I add the Title, FamilySize, Surname, and FamilyID columns to hopefully add a little more accuracy. This is not necessarily simplifying the data, but it is creating relationships that were not previously present in a column. These columns will hopefully be useful.

```
# Fill in Age NAs
summary(combo$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.17   21.00   28.00   29.88   39.00   80.00     263
```

```
Agefit <- rpart(Age ~ Pclass + Sex + SibSp + Parch + Fare + Embarked + Title + FamilySize,
                data = combo[!is.na(combo$Age), ], method = "anova")
combo$Age[is.na(combo$Age)] <- predict(Agefit, combo[is.na(combo$Age),])
# Check what else might be missing
summary(combo)
```

```
##   PassengerId      Survived          Pclass          Name
##  Min.   :   1   Min.   :0.0000   Min.   :1.000   Length:1309
##  1st Qu.: 328   1st Qu.:0.0000   1st Qu.:2.000   Class :character
##  Median : 655   Median :0.0000   Median :3.000   Mode  :character
##  Mean   : 655   Mean   :0.3838   Mean   :2.295
##  3rd Qu.: 982   3rd Qu.:1.0000   3rd Qu.:3.000
##  Max.   :1309   Max.   :1.0000   Max.   :3.000
```

```
##                 NA's   :418
##      Sex           Age            SibSp            Parch
##  female:466   Min.   : 0.17   Min.   :0.0000   Min.   :0.000
##  male  :843   1st Qu.:22.00   1st Qu.:0.0000   1st Qu.:0.000
##               Median :28.86   Median :0.0000   Median :0.000
##               Mean   :29.70   Mean   :0.4989   Mean   :0.385
##               3rd Qu.:36.50   3rd Qu.:1.0000   3rd Qu.:0.000
##               Max.   :80.00   Max.   :8.0000   Max.   :9.000
##
##       Ticket          Fare                      Cabin      Embarked
##  CA. 2343:  11   Min.   :  0.000                  :1014    :  2
##  1601    :   8   1st Qu.:  7.896   C23 C25 C27    :   6   C:270
##  CA 2144 :   8   Median : 14.454   B57 B59 B63 B66:   5   Q:123
##  3101295 :   7   Mean   : 33.295   G6             :   5   S:914
##  347077  :   7   3rd Qu.: 31.275   B96 B98        :   4
##  347082  :   7   Max.   :512.329   C22 C26        :   4
##  (Other) :1261   NA's   :1         (Other)        : 271
##      Title       FamilySize       Surname            FamilyID
##  Mr     :757   Min.   : 1.000   Length:1309        Large :  60
##  Miss   :260   1st Qu.: 1.000   Class :character   Medium: 224
##  Mrs    :197   Median : 1.000   Mode  :character   Small :1025
##  Master : 61   Mean   : 1.884
##  Dr     :  8   3rd Qu.: 2.000
##  Rev    :  8   Max.   :11.000
##  (Other): 18
```

```
# Fill in Embarked blanks
summary(combo$Embarked)
```

```
##       C   Q   S
##   2 270 123 914
```

```
which(combo$Embarked == '')
```

```
## [1]  62 830
```

```
combo$Embarked[c(62,830)] = "S"
combo$Embarked <- factor(combo$Embarked)
# Fill in Fare NAs
summary(combo$Fare)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   7.896  14.450  33.300  31.280 512.300       1
```

```
which(is.na(combo$Fare))
```

```
## [1] 1044
```

```
combo$Fare[1044] <- median(combo$Fare, na.rm = T)
```

- It is very important to set the seed for reproducible results. The seed that is set is `set.seed(4872)`

Random Forest I will use the Random Forest Model as an example of how to predict properly. The as.factor(Survived) ensures that our prediction using the **predict** function is a bernoulli outcome *(0 or 1)*. The accur.rf is the in-training accuracy of this particular model.

```
fit.rf <- randomForest(as.factor(Survived) ~., data = subTraining,
                       importance = T, ntree = 200)
pred.rf <- predict(fit.rf, validation)
confusionMatrix(pred.rf, validation$Survived)
```

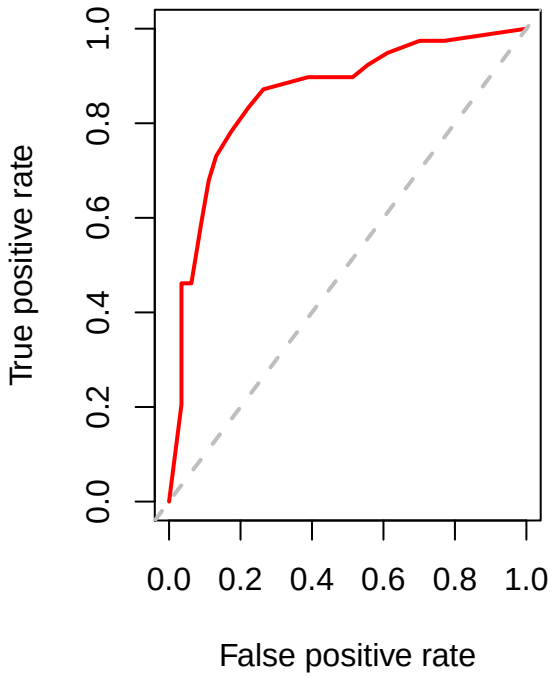
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 128  22
##          1  16  56
##
##                Accuracy : 0.8288
##                  95% CI : (0.7727, 0.8759)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 2.153e-09
##
##                   Kappa : 0.6177
##  Mcnemar's Test P-Value : 0.4173
##
##             Sensitivity : 0.8889
##             Specificity : 0.7179
##          Pos Pred Value : 0.8533
##          Neg Pred Value : 0.7778
##              Prevalence : 0.6486
##          Detection Rate : 0.5766
##    Detection Prevalence : 0.6757
##       Balanced Accuracy : 0.8034
##
##        'Positive' Class : 0
##
```

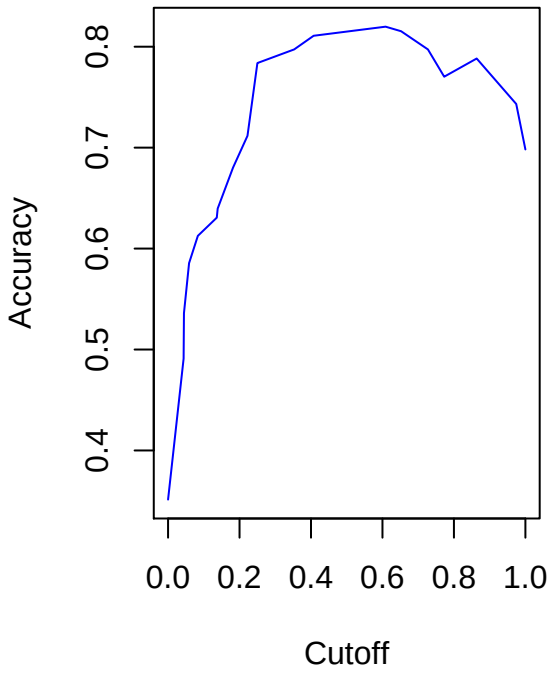
```
accur.rf <- confusionMatrix(pred.rf, validation$Survived)$overall[[1]]
```

```
fit.rpart <- rpart(Survived ~., data = subTraining, control = rpart.control(minsplit = 50, cp=0))
```

Classification and Regression Trees

```
##   accuracy cutoff.890
##  0.8198198  0.6086957
```






```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 125  21
##          1  19  57
##
##                Accuracy : 0.8198
##                  95% CI : (0.7628, 0.868)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 1.417e-08
##
##                   Kappa : 0.6024
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.8681
##             Specificity : 0.7308
##          Pos Pred Value : 0.8562
##          Neg Pred Value : 0.7500
##              Prevalence : 0.6486
##          Detection Rate : 0.5631
##    Detection Prevalence : 0.6577
##       Balanced Accuracy : 0.7994
##
##        'Positive' Class : 0
##
```

```
fit.cforest <- cforest(as.factor(Survived) ~., data = subTraining,
                       controls = cforest_unbiased(ntree=2000, mtry=3))
```

cForest

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 126  24
##          1  18  54
##
##                Accuracy : 0.8108
##                  95% CI : (0.753, 0.8601)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 8.279e-08
##
##                   Kappa : 0.5775
##  Mcnemar's Test P-Value : 0.4404
##
##             Sensitivity : 0.8750
##             Specificity : 0.6923
##          Pos Pred Value : 0.8400
##          Neg Pred Value : 0.7500
##              Prevalence : 0.6486
##          Detection Rate : 0.5676
##    Detection Prevalence : 0.6757
##       Balanced Accuracy : 0.7837
##
##        'Positive' Class : 0
##
```

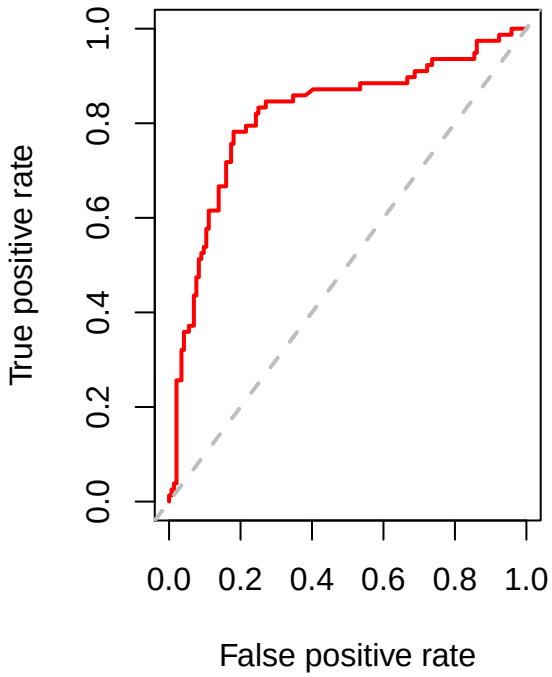
```
fit.gbm <- gbm(Survived ~., data = subTraining, distribution = "bernoulli", shrinkage = 0.01,
               interaction.depth = 2, n.trees = 2000)
pred.gbm <- predict(fit.gbm, validation, n.trees = 2000, "response")
```

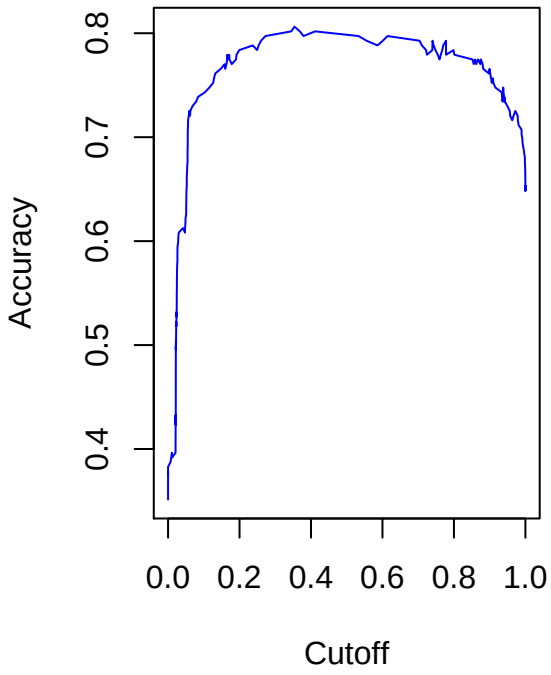
Gradient Boosted Model

```
fit.nb <- naiveBayes(as.factor(Survived) ~., data = subTraining, laplace = 3)
```

Naive Bayes

```
##  accuracy    cutoff
## 0.8063063 0.3537814
```






```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 118  17
##          1  26  61
##
##                Accuracy : 0.8063
##                  95% CI : (0.7481, 0.8561)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 1.917e-07
##
##                   Kappa : 0.586
##  Mcnemar's Test P-Value : 0.2225
##
##             Sensitivity : 0.8194
##             Specificity : 0.7821
##          Pos Pred Value : 0.8741
##          Neg Pred Value : 0.7011
##              Prevalence : 0.6486
##          Detection Rate : 0.5315
##    Detection Prevalence : 0.6081
##       Balanced Accuracy : 0.8007
##
##        'Positive' Class : 0
##
```

```
fit.svm <- svm(as.factor(Survived) ~., data = subTraining, kernel = "radial",
               degree = 3, cross = 4)
```

Support Vector Machines

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 126  21
##          1  18  57
##
##                Accuracy : 0.8243
##                  95% CI : (0.7678, 0.872)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 5.607e-09
##
##                   Kappa : 0.6112
##  Mcnemar's Test P-Value : 0.7488
##
##             Sensitivity : 0.8750
##             Specificity : 0.7308
##          Pos Pred Value : 0.8571
##          Neg Pred Value : 0.7600
##              Prevalence : 0.6486
##          Detection Rate : 0.5676
##    Detection Prevalence : 0.6622
##       Balanced Accuracy : 0.8029
##
##        'Positive' Class : 0
##
```

```
fit.nnet <- train(as.factor(Survived) ~., data = subTraining, method = "nnet", trace = F)
```

```
pred.nnet <- predict(fit.nnet, validation)
confusionMatrix(pred.nnet, validation$Survived)
```

Artificial Neural Networks

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 130  28
##          1  14  50
##
##                Accuracy : 0.8108
##                  95% CI : (0.753, 0.8601)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 8.279e-08
##
##                   Kappa : 0.5671
```

```
##  Mcnemar's Test P-Value : 0.04486
##
##             Sensitivity : 0.9028
##             Specificity : 0.6410
##          Pos Pred Value : 0.8228
##          Neg Pred Value : 0.7812
##              Prevalence : 0.6486
##          Detection Rate : 0.5856
##    Detection Prevalence : 0.7117
##       Balanced Accuracy : 0.7719
##
##        'Positive' Class : 0
##
```

```
accur.nnet <- confusionMatrix(pred.nnet, validation$Survived)$overall[[1]]
```

```
grid <- expand.grid(size=c(5,10,20,50), k=c(1,2,3,4,5))
fit.lvq <- train(as.factor(Survived) ~., data = subTraining, method = "lvq", tuneGrid = grid)
```

Learning Vector Quantization

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  29
##          1  40  49
##
##                Accuracy : 0.6892
##                  95% CI : (0.6238, 0.7494)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 0.1154
##
##                   Kappa : 0.3395
##  Mcnemar's Test P-Value : 0.2286
##
##             Sensitivity : 0.7222
##             Specificity : 0.6282
##          Pos Pred Value : 0.7820
##          Neg Pred Value : 0.5506
##              Prevalence : 0.6486
##          Detection Rate : 0.4685
##    Detection Prevalence : 0.5991
##       Balanced Accuracy : 0.6752
##
##        'Positive' Class : 0
##
```

```
fit.fda <- train(as.factor(Survived) ~., data = subTraining, method = "fda")
```

Flexible Discriminant Analysis

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 125  21
##          1  19  57
##
##                Accuracy : 0.8198
##                  95% CI : (0.7628, 0.868)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 1.417e-08
##
##                   Kappa : 0.6024
##  Mcnemar's Test P-Value : 0.8744
##
##             Sensitivity : 0.8681
##             Specificity : 0.7308
##          Pos Pred Value : 0.8562
##          Neg Pred Value : 0.7500
##              Prevalence : 0.6486
##          Detection Rate : 0.5631
##    Detection Prevalence : 0.6577
##       Balanced Accuracy : 0.7994
##
##        'Positive' Class : 0
##
```

```
fit.ml <- multinom(as.factor(Survived) ~., data = subTraining)
```

Multinomial Logit Model

```
## # weights:  23 (22 variable)
## initial  value 463.715464
## iter  10 value 271.917465
## iter  20 value 262.131908
## iter  30 value 261.569695
## iter  40 value 261.560257
## final  value 261.560219
## converged
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 123  17
##          1  21  61
##
##                Accuracy : 0.8288
##                  95% CI : (0.7727, 0.8759)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 2.153e-09
##
##                   Kappa : 0.6288
##  Mcnemar's Test P-Value : 0.6265
##
##             Sensitivity : 0.8542
##             Specificity : 0.7821
##          Pos Pred Value : 0.8786
##          Neg Pred Value : 0.7439
##              Prevalence : 0.6486
##          Detection Rate : 0.5541
##    Detection Prevalence : 0.6306
##       Balanced Accuracy : 0.8181
##
##        'Positive' Class : 0
##
```

Model Fitting

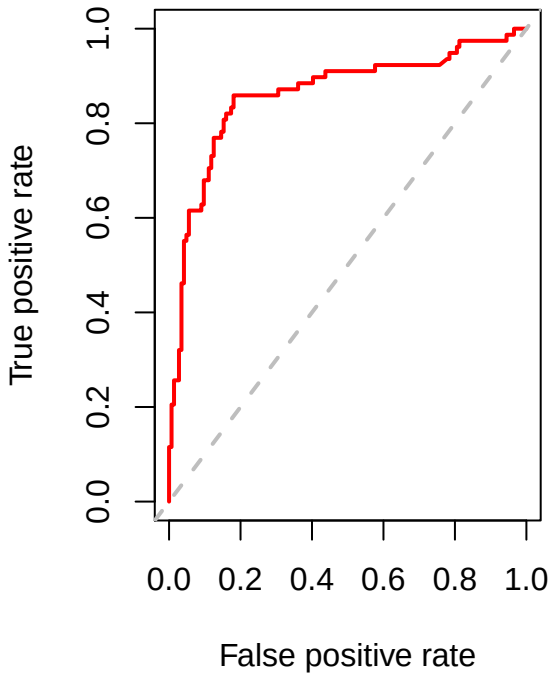
Throughout the model fitting process **gbm**, **naive bayes**, and the **cart** models all cannot be coerced into a classification model, and are regressions. This means that when we predict, we must choose an optimal cutoff for the prediction to be categorized as survived or dead. The selection of the proper cutoff is done based off of the ROC (Receiver Operating Characteristic) curve.

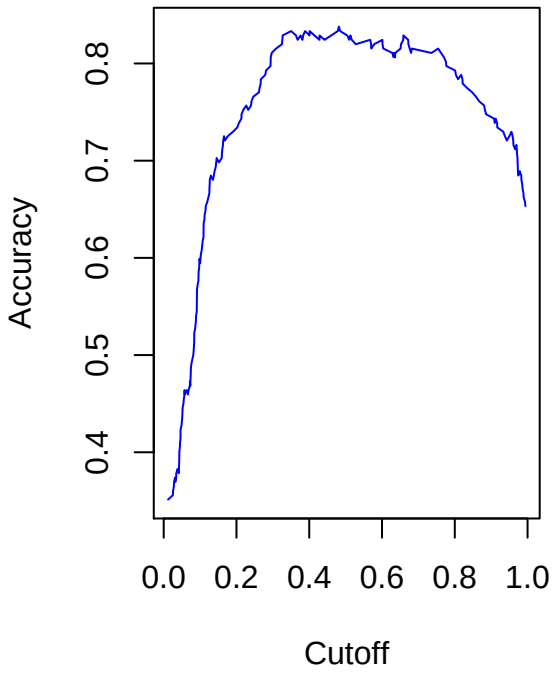
For example, the way to pick and visualize the cutoff for the gbm model is done like this :

```
predGBM <- prediction(pred.gbm, validation$Survived)
roc.perfGBM = performance(predGBM, measure = "tpr", x.measure = "fpr")

# Function for finding the optimal cutoff
acc.perfGBM = performance(predGBM, measure = "acc")

indGBM = which.max(slot(acc.perfGBM, "y.values")[[1]])
accGBM = slot(acc.perfGBM, "y.values")[[1]][indGBM]
cutoffGBM = slot(acc.perfGBM, "x.values")[[1]][indGBM]
print(c(accuracy= accGBM, cutoff = cutoffGBM))
```

```
##  accuracy    cutoff
## 0.8378378 0.4814433
```

```
#auc.perfGBM = performance(predGBM, measure = "auc")
#print(auc.perfGBM)

par(mfrow = c(1, 2))
plot(roc.perfGBM, col = "red", lwd = 2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
plot(acc.perfGBM, col = "blue")
```






```
pred.gbm <- ifelse(pred.gbm >= cutoffGBM, 1, 0)
confusionMatrix(pred.gbm, validation$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 126  18
##          1  18  60
##
##                Accuracy : 0.8378
##                  95% CI : (0.7826, 0.8838)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 2.893e-10
##
##                   Kappa : 0.6442
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8750
##             Specificity : 0.7692
##          Pos Pred Value : 0.8750
##          Neg Pred Value : 0.7692
##              Prevalence : 0.6486
##          Detection Rate : 0.5676
##    Detection Prevalence : 0.6486
##       Balanced Accuracy : 0.8221
##
##        'Positive' Class : 0
##
```

```
accur.gbm <- confusionMatrix(pred.gbm, validation$Survived)$overall[[1]]
```

At the bottom we can see that we commented out the AUC characteristic, which is a more advanced way to look at the predictive capabilities of a model.

Picking a model

I will look at all 10 of our predictions and pick the top 8 that have the highest in-training accuracy. An alternative approach could be to pick the best prediction algorithms based on the AUC. Although AUC is not a perfect representation of how each algorithm will perform, it is still a very good measure of the predictive capabilities of a model.

I will then use these 8 models and concatinate each prediction with the validation Survived column to get a data frame with 9 columns of Survived data. Furthermore, I will then train this new data with the ***Random Forest Model*** because it has a very high accuracy level. Some advantages of the Random Forest model is that it is a robust model as well as taking advantage of bagging to increase the predictive power and reduce variance. This result will be flawed because the prediction data frame contains the Survived column from the validation data set even though I tested the model precisely with the same validation set.

Combining classifiers is not always the best method. With this model, I got an accuracy of ***0.79426*** based off of the public leader boards. This is clearly not the best model as it would put us around ***900/2941*** which is not the best. I am much higher up on the leader boards (mid 200s) with an accuracy of 0.81340, and I advise you to figure out how to build successful predictive models on your own. You can rather easily get the same accuracy of 0.79426 with a careful randomForest model as I did on my first try.

Final Model

Warning : Some code here differs from the R script

I first make a list of all of the accuracies of the 10 models to determine which models perform the best. I will differ from the R script here as I will use a *Random Forest* model for the final model as opposed to GBM. This is because we do not have to predict a cutoff for the data, which will hopefully improve the accuracy.

```
Algorithm <- c("CART", "Random Forest", "cForest", "GBM", "Naive Bayes", "SVM",
               "ANN", "LVQ", "FDA", "Multinomial Logit Model")
Accuracy <- c(accur.rpart, accur.rf, accur.cforest, accur.gbm, accur.nb, accur.svm,
              accur.nnet, accur.lvq, accur.fda, accur.ml)
TopPerformance <- data.frame(Accuracy)
colnames(TopPerformance) <- "Accuracy"
rownames(TopPerformance) <- Algorithm
print(TopPerformance)
```

```
##                            Accuracy
## CART                      0.8198198
## Random Forest             0.8288288
## cForest                   0.8108108
## GBM                       0.8378378
## Naive Bayes               0.8063063
## SVM                       0.8243243
## ANN                       0.8108108
## LVQ                       0.6891892
## FDA                       0.8198198
## Multinomial Logit Model 0.8288288
```

```
predDF <- data.frame(CART = pred.rpart, RF = pred.rf, GBM = pred.gbm, NNET = pred.nnet,
                     SVM = pred.svm, FDA = pred.fda, ML = pred.ml, cForest = pred.cforest,
                     Survived = validation$Survived)

fit.final <- randomForest(as.factor(Survived) ~., data = predDF, mtry = 3, ntree = 1600,
                          importance=TRUE)
pred.final <- predict(fit.final, predDF)
confusionMatrix(pred.final, validation$Survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 133  16
##          1  11  62
##
##                Accuracy : 0.8784
##                  95% CI : (0.828, 0.9183)
##     No Information Rate : 0.6486
##     P-Value [Acc > NIR] : 6.326e-15
##
##                   Kappa : 0.7292
##  Mcnemar's Test P-Value : 0.4414
##
##             Sensitivity : 0.9236
##             Specificity : 0.7949
##          Pos Pred Value : 0.8926
##          Neg Pred Value : 0.8493
##              Prevalence : 0.6486
##          Detection Rate : 0.5991
##    Detection Prevalence : 0.6712
##       Balanced Accuracy : 0.8592
##
##        'Positive' Class : 0
##
```

The final model selection will be an ensemble of the top 8 best performing classifiers. I must first train all 8 models on the testing data before combining the predictors. This is a step in which it confuses many people.

```
pred.rpartF <- predict(fit.rpart, testing)
pred.rpartF <- ifelse(pred.rpartF >= cutoffCART, 1, 0)
pred.rfF <- predict(fit.rf, testing)
pred.gbmF <- predict(fit.gbm, testing, n.trees = 2000, "response")
pred.gbmF <- ifelse(pred.gbmF >= cutoffGBM, 1, 0)
pred.nnetF <- predict(fit.nnet, testing)
pred.svmF <- predict(fit.svm, testing)
pred.mlF <- predict(fit.ml, testing)
pred.cforestF <- predict(fit.cforest, testing, OOB = T, type = "response")
pred.fdaF <- predict(fit.fda, testing)

finalDF <- data.frame(CART = pred.rpartF, RF = pred.rfF, GBM = pred.gbmF, NNET = pred.nnetF,
                      SVM = pred.svmF, ML = pred.mlF, cForest = pred.cforestF, FDA = pred.fdaF)
```

```r
pred.submission <- predict(fit.final, finalDF)
PassengerId <- 892:1309
submit <- data.frame(PassengerId = PassengerId, Survived = pred.submission)
write.csv(submit, file = "Ensemble10.csv", row.names = FALSE)
```

Conclusion

Throughout this analysis, it is evident that combining predictors is not always the best method to use. Ensembling methods can be very advantagious, but only if the predictors only vary slightly. If the predictors have a large variance, although some may see this as a positive, it is difficult for any model to try to predict on such a large amount of conflicting information. This is an especially prevalent issue when dealing with large data sets as there will potentially be thousands of conflicting estimators between each model. This goal is to reduce this variance between predictors as little as possible.

In summary, the key is to extrapolate as much information as you can throughout the exploratory data analysis phase so that each algorithm will not be overly susceptible to overfitting / underfitting. It is also important the you do not include too many models if you choose to use ensembling, as it might lead to overfitting.

Some popular and powerful algorithms to try include, but are not limited to :

1. Random Forest
2. GBM
3. KNN
4. ANN
5. cForest (although it is still in development)
6. CART
7. SVM