Ensemble Methods

Code

Importing Data

Hide

```
library(randomForest)
library(adabag)
library(xgboost)
library(performanceEstimation)
library(mltools)

train <- read.csv("C:/Users/jah200003/Downloads/training.csv", header=TRUE, stringsAsFactors=TRUE)
test <- read.csv("C:/Users/jah200003/Downloads/testing.csv", header=TRUE, stringsAsFactors=TRUE)
```

Data Cleaning

Data is already prepared fairly well, but the classes are very imbalanced Applied synthetic oversampling and undersampling to improve balance before SVM (Copied from classification notebook)

Hide

```
table(train$class)
```

```
     farm    forest     grass impervious    orchard     water 
     1441      7431       446        969         53       205 
```

Hide

```
train <- smote(class~ ., train, perc.over = 50, perc.under = 5)
table(train$class)
```

```
     farm    forest     grass impervious    orchard     water 
     1777      9470       555       1173       2703       275 
```

Random Forest

Hide

```
rf <- randomForest(class~ ., data = train, importance = TRUE)

predrf <- predict(rf, newdata = test, type = "response")
accrf <- mean(predrf == test$class)
mccrf <- mcc(factor(predrf), test$class)

print(paste("Accuracy: ", accrf))
```

```
[1] "Accuracy:  0.663333333333333"
```

Hide

```
print(paste("MCC: ", mccrf))
```

```
[1] "MCC:  0.604621407160539"
```

Random forest performed the best of all three ensemble techniques, likely due to the large amount of predictors and the relative lack of importance for most of them. For some types of land, like intact forests and open water, the vegetation index will not change much over time, causing the chronological indexes to potentially do more harm than good. None of the algorithms had an exceptionally high accuracy, but given the fact that there is a 1-in-6 chance of correctly guessing the type of land, hovering around 50% would be great in an even distribution, though the dominance of forest land in the training data threatens that idea. The runtime for random forest was about 45 seconds, being neither the slowest nor the fastest algorithm.

Ada Boost

Hide

```
adab <- boosting(class~ ., data = train, boos = TRUE, coeflearn = "Breiman")
summary(adab)
```

```
           Length Class   Mode
formula         3  formula call
trees         100  -none-  list
weights       100  -none-  numeric
votes       95718  -none-  numeric
prob        95718  -none-  numeric
class       15953  -none-  character
importance     28  -none-  numeric
terms           3  terms   call
call            5  -none-  call
```

Hide

```
predada <- predict(adab, newdata = test, type = "response")
accada <- mean(predada$class == test$class)
mccada <- mcc(factor(predada$class), test$class)

print(paste("Accuracy: ", accada))
```

```
[1] "Accuracy:  0.6"
```

Hide

```
print(paste("MCC: ", mccada))
```

```
[1] "MCC:  0.520903676908143"
```

On paper, this algorithm should have done very well because it works well with weak classifiers and nonbinary identification, but the algorithm was outdone by random forests. On top of that, this was the slowest algorithm, taking roughly a minute to go through the same number of iterations. The lower accuracy may be due to outliers or imbalance in the data set.

XGBoost

Hide

```
train_labels <- sapply(as.numeric(train$class), function(x) x - 1)
train_matrix <- data.matrix(train[ , -29])
xgmodel <- xgboost(data = train_matrix, label = train_labels, verbose = 0, nrounds = 100, num_class = 6, ob
jective = "multi:softmax")

test_labels <- sapply(as.numeric(test$class), function(x) x - 1)
test_matrix <- data.matrix(test[ , -29])

probs <- predict(xgmodel, test_matrix)
predxg <- ifelse(probs > 0.5, 1, 0)

accxg <- mean(predxg == test_labels)
mccxg <- mcc(predxg, test_labels)

print(paste("Accuracy: ", accxg))
```

```
[1] "Accuracy:  0.436666666666667"
```

Hide

```
print(paste("MCC: ", mccxg))
```

```
[1] "MCC:  0.391630713194483"
```

This algorithm performed the worst out of the three algorithms, which may be caused by trees that are too deep and are overfitting the data set. It is also certainly possible that I am doing something wrong, as the given documentation pertained to binary classification, but this land data set has six different classes to select from. On the bright side, this algorithm is blazingly fast and took fewer than 5 seconds.