Ensemble Techniques

Ethan Huynh and Ryan Gagliardi

Load packages and data. Factor the necessary columns.

```
library(e1071)
library(MASS)
library(RWeka)
df <- read.csv('adult.csv')
colnames(df) <- c("age","workclass","fnlwgt","education","education-num","marital-status","occup
ation","relationship","race","sex","capital-gain","capital-loss","hours-per-week","native-countr
y","income")
df$income <- factor(df$income)
df$education <- factor(df$education)
df$sex <- factor(df$sex)
df$race <- factor(df$race)
```

Divide into train, test, validate

Cut the data set from 32000 rows to 13000 rows randomly Then divide into train, test, and validate sets

```
library(mltools)
```

```
##
## Attaching package: 'mltools'
```

```
## The following object is masked from 'package:e1071':
##
##     skewness
```

```
set.seed(6164)
i <- sample(1:nrow(df), 0.4*nrow(df), replace=FALSE)
df2 <- df[i,]

spec <- c(train=.6, test=.2, validate=.2)
i2 <- sample(cut(1:nrow(df2),nrow(df2)*cumsum(c(0,spec)), labels=names(spec)))
train <- df2[i2=="train",]
test <- df2[i2=="test",]
vald <- df2[i2=="validate",]
```

Random Forest

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(6164)
rf <- randomForest(income~education+sex+race, data=train, importance=TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = income ~ education + sex + race, data = train,      importance = TRU
E)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 21.41%
## Confusion matrix:
##         <=50K  >50K class.error
##  <=50K   5796   140  0.02358491
##  >50K    1533   345  0.81629393
```

```
pred <- predict(rf, newdata=test, type="response")
acc_rf <- mean(pred==test$income)
mcc_rf <- mcc(factor(pred), test$income)
print(paste("accuracy=", acc_rf))
```

```
## [1] "accuracy= 0.774664107485605"
```

```
print(paste("mcc=", mcc_rf))
```

```
## [1] "mcc= 0.243573696870038"
```

boosting from adabag library

```
library(adabag)
```

```
## Loading required package: rpart
```

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Loading required package: lattice
```

```
## Loading required package: foreach
```

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
adab1 <- boosting(income~education+sex+race, data=train, boos=TRUE, mfinal=20, coeflearn='Breima
n')
summary(adab1)
```

```
##            Length Class   Mode
## formula        3  formula call
## trees         20  -none-  list
## weights       20  -none-  numeric
## votes      15628  -none-  numeric
## prob       15628  -none-  numeric
## class       7814  -none-  character
## importance     3  -none-  numeric
## terms          3  terms   call
## call           6  -none-  call
```

```
pred <- predict(adab1, newdata=test, type="response")
acc_adabag <- mean(pred$class==test$income)
mcc_adabag <- mcc(factor(pred$class), test$income)
print(paste("accuracy=", acc_adabag))
```

```
## [1] "accuracy= 0.77658349328215"
```

```
print(paste("mcc=", mcc_adabag))
```

```
## [1] "mcc= 0.25753198950328"
```

XGBoost

```
library(xgboost)
train_label <- ifelse(train$income==" >50K", 1, 0)
train_matrix <- data.matrix(train[c(4,9,10)])
temp <- train[c(4,9,10)]
model <- xgboost(data=train_matrix, label=train_label,
                 nrounds=100, objective='binary:logistic')
```

```
## [1]  train-logloss:0.591014
## [2]  train-logloss:0.536882
## [3]  train-logloss:0.505865
## [4]  train-logloss:0.487495
## [5]  train-logloss:0.476419
## [6]  train-logloss:0.469554
## [7]  train-logloss:0.465317
## [8]  train-logloss:0.462646
## [9]  train-logloss:0.460836
## [10] train-logloss:0.459703
## [11] train-logloss:0.458992
## [12] train-logloss:0.458468
## [13] train-logloss:0.457665
## [14] train-logloss:0.457379
## [15] train-logloss:0.456871
## [16] train-logloss:0.456655
## [17] train-logloss:0.456341
## [18] train-logloss:0.456205
## [19] train-logloss:0.455974
## [20] train-logloss:0.455787
## [21] train-logloss:0.455705
## [22] train-logloss:0.455666
## [23] train-logloss:0.455632
## [24] train-logloss:0.455536
## [25] train-logloss:0.455444
## [26] train-logloss:0.455362
## [27] train-logloss:0.455345
## [28] train-logloss:0.455332
## [29] train-logloss:0.455279
## [30] train-logloss:0.455233
## [31] train-logloss:0.455210
## [32] train-logloss:0.455188
## [33] train-logloss:0.455156
## [34] train-logloss:0.455147
## [35] train-logloss:0.455060
## [36] train-logloss:0.455052
## [37] train-logloss:0.454999
## [38] train-logloss:0.454992
## [39] train-logloss:0.454944
## [40] train-logloss:0.454915
## [41] train-logloss:0.454896
## [42] train-logloss:0.454883
## [43] train-logloss:0.454877
## [44] train-logloss:0.454857
## [45] train-logloss:0.454852
## [46] train-logloss:0.454830
## [47] train-logloss:0.454797
## [48] train-logloss:0.454777
## [49] train-logloss:0.454772
## [50] train-logloss:0.454760
## [51] train-logloss:0.454756
## [52] train-logloss:0.454745
## [53] train-logloss:0.454741
## [54] train-logloss:0.454731
## [55] train-logloss:0.454710
## [56] train-logloss:0.454696
## [57] train-logloss:0.454685
## [58] train-logloss:0.454681
## [59] train-logloss:0.454669
## [60] train-logloss:0.454665
## [61] train-logloss:0.454657
## [62] train-logloss:0.454650
## [63] train-logloss:0.454616
## [64] train-logloss:0.454599
## [65] train-logloss:0.454587
## [66] train-logloss:0.454583
## [67] train-logloss:0.454575
## [68] train-logloss:0.454572
## [69] train-logloss:0.454554
## [70] train-logloss:0.454517
## [71] train-logloss:0.454510
## [72] train-logloss:0.454504
## [73] train-logloss:0.454479
## [74] train-logloss:0.454474
## [75] train-logloss:0.454468
## [76] train-logloss:0.454461
## [77] train-logloss:0.454455
## [78] train-logloss:0.454452
## [79] train-logloss:0.454446
## [80] train-logloss:0.454443
## [81] train-logloss:0.454438
## [82] train-logloss:0.454416
## [83] train-logloss:0.454406
## [84] train-logloss:0.454383
## [85] train-logloss:0.454350
## [86] train-logloss:0.454345
## [87] train-logloss:0.454330
## [88] train-logloss:0.454309
## [89] train-logloss:0.454288
## [90] train-logloss:0.454285
## [91] train-logloss:0.454270
## [92] train-logloss:0.454266
## [93] train-logloss:0.454252
## [94] train-logloss:0.454245
## [95] train-logloss:0.454232
## [96] train-logloss:0.454228
## [97] train-logloss:0.454224
## [98] train-logloss:0.454205
## [99] train-logloss:0.454201
## [100]    train-logloss:0.454192
```

```
test_label <- ifelse(test$income==" >50K", 1, 0)
test_matrix <- data.matrix(test[c(4,9,10)])
probs <- predict(model, test_matrix)
pred <- ifelse(probs>0.5, 1, 0)
acc_xg <- mean(pred==test_label)
mcc_xg <- mcc(pred, test_label)
print(paste("accuracy=", acc_xg))
```

```
## [1] "accuracy= 0.773128598848368"
```

```
print(paste("mcc=", mcc_xg))
```

```
## [1] "mcc= 0.24110453384685"
```

Best method

In the case of Random Forest, XGBoost, and basic boosting, their speeds and accuracy vary. Basic boosting is much slower than the other two, followed by random forest, and XGBoost being the fastest by far. Random Forest and XGBoost are both .77 accuracy, while basic boost is slightly more accurate at .79 accuracy. The sacrifice of speed for accuracy is shown here with basic boosting which is slower than the rest but also more accurate. As you scale the amount of data used it might change which method you choose to use for analysis. With massive data sets it is most likely more efficient to use one of the faster options, while if the data set is smaller and needs to be more precise, basic boosting is far better.