Lab 3

Urminder Singh

March 25, 2018

Data

Data was downloaded from https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center. Data coressponds 748 blood donors. There are four variables R (Recency - months since last donation), F (Frequency - total number of donation), M (Monetary - total blood donated in c.c.), T (Time - months since first donation), and a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

Solution 1

I have implemented RandomForest and AdaBoost.M1 using caret library in R.

Random Forest

The code I implemented for RandomForest is attached below.

```
# load libraries
library(caret)
library(readr)
library("caretEnsemble")
library("magrittr", lib.loc = "~/R/win-library/3.4")
# read test and train data lab3_train <- read_csv('lab3-train.csv',
# col_types = cols(Class = col_factor(levels = c('0','1'))))
lab3_train <- read_csv("lab3-train.csv")
lab3_test <- read_csv("lab3-test.csv", col_types = cols(Class = col_factor(levels = c("0",
    "1"))))

# Read the input files
lab3_train = read.csv("lab3-train.csv")
lab3_test = read.csv("lab3-test.csv")
# change labels
lab3_train$Class = ifelse(lab3_train$Class == 1, "Yes", "No")
lab3_test$Class = ifelse(lab3_test$Class == 1, "Yes", "No")
# x, y has training data
x <- lab3_train[, 1:4]
y = lab3_train[, 5]

### Train RF model

# define hyperparams
ntree <- c(1, 50, 100, 1000)
preproc <- c("center", "scale", "knnImpute")
tunelen <- c(1, 2, 3, 5)
trcontrol = trainControl(method = "cv", number = 5, summaryFunction = twoClassSummary,
```

```
    classProbs = TRUE)
# table to store results
header1 <- c("ntree", "preProc", "tuneLength", "mtry", "ROC", "Acc_test", "Time in min")
rf_resultsTab <- data.frame(matrix(ncol = 7, nrow = 0))
colnames(rf_resultsTab) <- header1
best_rf_model <- NULL
best_rf_acc <- 0
for (nt in ntree) {
    for (pp in preproc) {
        # print (pp)
        for (tl in tunelen) {
            s <- proc.time()  #start time
            rf_model = train(x, as.factor(y), method = "rf", trControl = trcontrol,
                metric = "Accuracy", tuneLength = tl, preProc = pp, ntree = nt)
            d <- proc.time() - s  #end time
            # calculate test accuraccy
            pred_res <- predict(rf_model, lab3_test)
            correct <- 0
            for (i in (1:length(pred_res))) {
                if (pred_res[i] == lab3_test$Class[i]) {
                  correct = correct + 1
                }
            }
            test_acc <- (correct)/length(pred_res)

            if (test_acc > best_rf_acc) {
                best_rf_acc = test_acc
                best_rf_model <- rf_model
            }

            rf_resultsTab[nrow(rf_resultsTab) + 1, ] <- c(nt, pp, tl, rf_model$finalModel$mtry,
                max(rf_model$results$ROC), test_acc, format(as.numeric(d)[3]/60,
                  digits = 2))

        }
    }
}
```

```
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
```

```
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
##
## note: only 3 unique complexity parameters in default grid. Truncating the grid to 3 .
```

```
rf_resultsTab %>% knitr::kable(caption = "Training RF with different hyperparameters")
```

Table 1: Training RF with different hyperparameters

ntree	preProc	tuneLength	mtry	ROC	Acc_test	Time in min
1	center	1	1	0.603832615578235	0.820598006644518	0.034
1	center	2	4	0.587583820030283	0.724252491694352	0.018
1	center	3	4	0.566552612431912	0.730897009966777	0.011
1	center	5	3	0.57768371580831	0.704318936877076	0.0092
1	scale	1	1	0.581171219003795	0.747508305647841	0.011
1	scale	2	4	0.564205651583977	0.681063122923588	0.0083
1	scale	3	4	0.576123335889721	0.754152823920266	0.013
1	scale	5	3	0.5773504021395	0.757475083056478	0.0092
1	knnImpute	1	1	0.581871276030913	0.757475083056478	0.012
1	knnImpute	2	2	0.599270446188032	0.740863787375415	0.0088
1	knnImpute	3	2	0.589806894381846	0.757475083056478	0.014
1	knnImpute	5	2	0.581969598647081	0.750830564784053	0.0098
50	center	1	1	0.662792755589641	0.803986710963455	0.013
50	center	2	4	0.650565355042967	0.724252491694352	0.01
50	center	3	4	0.623960238334022	0.730897009966777	0.016
50	center	5	2	0.624178514541915	0.787375415282392	0.012
50	scale	1	1	0.651271311427054	0.81063122923588	0.012
50	scale	2	4	0.628489961260889	0.73421926910299	0.01
50	scale	3	2	0.612211668928087	0.777408637873754	0.015
50	scale	5	4	0.639822626000433	0.717607973421927	0.017
50	knnImpute	1	1	0.652780563585236	0.803986710963455	0.0087
50	knnImpute	2	4	0.639427369083437	0.727574750830565	0.015
50	knnImpute	3	4	0.658153894558826	0.714285714285714	0.012
50	knnImpute	5	3	0.640214933238944	0.720930232558139	0.017
100	center	1	1	0.641625862780957	0.807308970099668	0.011
100	center	2	2	0.658700568304721	0.790697674418605	0.018
100	center	3	2	0.637770633001003	0.770764119601329	0.018
100	center	5	2	0.654164946020884	0.794019933554817	0.021
100	scale	1	1	0.655685013666844	0.803986710963455	0.012
100	scale	2	4	0.631337384225119	0.720930232558139	0.013
100	scale	3	2	0.647680569484593	0.780730897009967	0.019
100	scale	5	3	0.633212396515446	0.724252491694352	0.018
100	knnImpute	1	1	0.662410280612747	0.794019933554817	0.0095
100	knnImpute	2	2	0.623759660197039	0.774086378737541	0.016
100	knnImpute	3	4	0.655742040784221	0.730897009966777	0.019
100	knnImpute	5	3	0.654032210489057	0.724252491694352	0.018
1000	center	1	1	0.66886122745954	0.807308970099668	0.037
1000	center	2	2	0.631659882406151	0.770764119601329	0.069
1000	center	3	2	0.655897390517767	0.774086378737541	0.089
1000	center	5	2	0.637621182624427	0.777408637873754	0.085
1000	scale	1	1	0.655071480541954	0.807308970099668	0.029

ntree	preProc	tuneLength	mtry	ROC	Acc_test	Time in min
1000	scale	2	4	0.646209663146717	0.724252491694352	0.059
1000	scale	3	2	0.632035474799914	0.777408637873754	0.072
1000	scale	5	2	0.637246573456827	0.770764119601329	0.079
1000	knnImpute	1	1	0.640442058482292	0.800664451827243	0.031
1000	knnImpute	2	2	0.631335417772796	0.774086378737541	0.054
1000	knnImpute	3	3	0.641003480620612	0.720930232558139	0.076
1000	knnImpute	5	2	0.642396712091715	0.777408637873754	0.071

From Table 1 we can see that the random forest model with highest accuraccy on the test had following parameters: 1, center, 1, 1, 0.603832615578235, 0.820598006644518, 0.034. The maximum accuraccy acheived by RandomForest model on the test set is 0.820598

Best RandomForest Model performance

```
best_rf_model
```

```
## Random Forest
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 357, 358, 358, 357, 358
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6038326  0.8250565  0.3826087
##
## Tuning parameter 'mtry' was held constant at a value of 1
```

```
pred_res <- predict(best_rf_model, lab3_test)
confusionMatrix(pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  219  35
##        Yes  19  28
##
##                Accuracy : 0.8206
##                  95% CI : (0.7725, 0.8623)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.11289
##
##                   Kappa : 0.4022
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 0.9202
```

```
##             Specificity : 0.4444
##          Pos Pred Value : 0.8622
##          Neg Pred Value : 0.5957
##              Prevalence : 0.7907
##          Detection Rate : 0.7276
##    Detection Prevalence : 0.8439
##       Balanced Accuracy : 0.6823
##
##        'Positive' Class : No
##
```

AdaBoost.M1

The code I implemented for AdaBoost.M1 is attached below.

```
### Train AdaBoost.M1 model define ada hyperparams
clearn = c("Breiman", "Freund", "Zhu")
mdepth = c(2, 5)
mfinal = c(4, 10)
preproc <- c("center", "scale", "knnImpute")
# table to store results
header2 <- c("coeflearn", "preProc", "maxdepth", "mfinal", "ROC", "Acc_test",
    "Time in min")
ada_resultsTab <- data.frame(matrix(ncol = 7, nrow = 0))
colnames(ada_resultsTab) <- header2
best_ada_model <- NULL
best_ada_acc <- 0
for (cl in clearn) {
    for (md in mdepth) {
        for (mf in mfinal) {
            for (pp in preproc) {
                control = trainControl(method = "cv", number = 2, summaryFunction = twoClassSummary,
                  classProbs = TRUE)
                grid = expand.grid(mfinal = mf, maxdepth = md, coeflearn = cl)
                s <- proc.time()  #start time
                ada_model = train(x, as.factor(y), method = "AdaBoost.M1", trControl = control,
                  tuneGrid = grid, metric = "ROC", preProc = pp)
                d <- proc.time() - s  #end time
                # calculate test accuraccy
                pred_res <- predict(ada_model, lab3_test)
                correct <- 0
                for (i in (1:length(pred_res))) {
                  if (pred_res[i] == lab3_test$Class[i]) {
                    correct = correct + 1
                  }
                }
                test_acc <- (correct)/length(pred_res)
                if (test_acc > best_ada_acc) {
                  best_ada_acc = test_acc
                  best_ada_model <- ada_model
                }

                ada_resultsTab[nrow(ada_resultsTab) + 1, ] <- c(cl, pp, md,
                  mf, max(ada_model$results$ROC), test_acc, format(as.numeric(d)[3]/60,
```

```
                    digits = 2))

            }

        }
    }
}

ada_resultsTab %>% knitr::kable(caption = "Training AdaBoost.M1 with different hyperparameters")
```

Table 2: Training AdaBoost.M1 with different hyperparameters

coeflearn	preProc	maxdepth	mfinal	ROC	Acc_test	Time in min
Breiman	center	2	4	0.629254313078084	0.803986710963455	0.072
Breiman	scale	2	4	0.645648383005707	0.800664451827243	0.071
Breiman	knnImpute	2	4	0.658670526024242	0.813953488372093	0.07
Breiman	center	2	10	0.683182366489552	0.817275747508306	0.14
Breiman	scale	2	10	0.660679924780793	0.784053156146179	0.14
Breiman	knnImpute	2	10	0.700541275811048	0.823920265780731	0.14
Breiman	center	5	4	0.624213733336249	0.79734219269103	0.071
Breiman	scale	5	4	0.659476836565864	0.807308970099668	0.071
Breiman	knnImpute	5	4	0.546142829029366	0.794019933554817	0.071
Breiman	center	5	10	0.636314769058083	0.807308970099668	0.15
Breiman	scale	5	10	0.62134335891661	0.73421926910299	0.14
Breiman	knnImpute	5	10	0.632224451344398	0.813953488372093	0.14
Freund	center	2	4	0.692339229877769	0.794019933554817	0.07
Freund	scale	2	4	0.678092588138397	0.800664451827243	0.072
Freund	knnImpute	2	4	0.701687421191117	0.813953488372093	0.069
Freund	center	2	10	0.701981246218996	0.830564784053156	0.15
Freund	scale	2	10	0.688023418538036	0.827242524916944	0.16
Freund	knnImpute	2	10	0.701717031465244	0.820598006644518	0.16
Freund	center	5	4	0.60098151225592	0.803986710963455	0.077
Freund	scale	5	4	0.626183499879737	0.744186046511628	0.081
Freund	knnImpute	5	4	0.617837046917252	0.817275747508306	0.077
Freund	center	5	10	0.647033688292189	0.780730897009967	0.16
Freund	scale	5	10	0.659326051939154	0.79734219269103	0.16
Freund	knnImpute	5	10	0.658596728110263	0.790697674418605	0.16
Zhu	center	2	4	0.634652038280162	0.817275747508306	0.08
Zhu	scale	2	4	0.679138058586433	0.800664451827243	0.077
Zhu	knnImpute	2	4	0.659734673722112	0.800664451827243	0.079
Zhu	center	2	10	0.672317218055525	0.817275747508306	0.15
Zhu	scale	2	10	0.708607114483342	0.803986710963455	0.16
Zhu	knnImpute	2	10	0.677438884394201	0.807308970099668	0.16
Zhu	center	5	4	0.590732713066422	0.803986710963455	0.078
Zhu	scale	5	4	0.633297709895845	0.750830564784053	0.08
Zhu	knnImpute	5	4	0.685310206342612	0.730897009966777	0.074
Zhu	center	5	10	0.61074060306562	0.780730897009967	0.16
Zhu	scale	5	10	0.631204946829059	0.807308970099668	0.16
Zhu	knnImpute	5	10	0.66073413435958	0.727574750830565	0.16

From Table 2 we can see that the AdaBoost.M1 model with highest accuraccy on the test had following parameters: Freund, center, 2, 10, 0.701981246218996, 0.830564784053156, 0.15. The maximum accuraccy

acheived by AdaBoost.M1 model on the test set is 0.8305648

Best AdaBoost.M1 Model performance

```
best_ada_model
```

```
## AdaBoost.M1
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 224, 223
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7019812  0.8945783  0.2779794
##
## Tuning parameter 'mfinal' was held constant at a value of 10
##
## Tuning parameter 'maxdepth' was held constant at a value of 2
##
## Tuning parameter 'coeflearn' was held constant at a value of Freund
```

```
pred_res <- predict(best_ada_model, lab3_test)
confusionMatrix(pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  224  37
##        Yes  14  26
##
##                Accuracy : 0.8306
##                  95% CI : (0.7833, 0.8712)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.048979
##
##                   Kappa : 0.4087
##  Mcnemar's Test P-Value : 0.002066
##
##             Sensitivity : 0.9412
##             Specificity : 0.4127
##          Pos Pred Value : 0.8582
##          Neg Pred Value : 0.6500
##              Prevalence : 0.7907
##          Detection Rate : 0.7442
##    Detection Prevalence : 0.8671
##       Balanced Accuracy : 0.6769
##
##        'Positive' Class : No
```

##

Solution 2

I trained individual Neural Network (NN), KNN, Logistic Regression (LR), Naive Bayes (NB), and Decision Tree (DT). The code i implemented is below.

Neural Network

```
header3 <- c("Model", "Acc_test", "Time in min")
ind_resultsTab <- data.frame(matrix(ncol = 3, nrow = 0))
colnames(ind_resultsTab) <- header3
# train neural net
nn_test_acc <- 0
grid <- expand.grid(.decay = c(0.5, 0.1), .size = c(4, 4, 2))
control = trainControl(method = "cv", number = 2, summaryFunction = twoClassSummary,
    classProbs = TRUE)
s <- proc.time()  #start time
nnet_model = train(Class ~ ., data = lab3_train, method = "nnet", trControl = control,
    metric = "ROC", tuneLength = 3, preProc = c("center"))
```

```
## # weights:  7
## initial  value 154.699651
## final  value 123.506644
## converged
## # weights:  19
## initial  value 141.223477
## final  value 123.506644
## converged
## # weights:  31
## initial  value 173.031355
## final  value 123.506644
## converged
## # weights:  7
## initial  value 132.795059
## iter  10 value 121.070033
## iter  20 value 114.601351
## final  value 114.575875
## converged
## # weights:  19
## initial  value 137.834729
## iter  10 value 123.678074
## iter  20 value 113.741794
## iter  30 value 106.300065
## iter  40 value 105.169580
## iter  50 value 98.091463
## iter  60 value 96.614019
## iter  70 value 93.995162
## iter  80 value 93.263981
## final  value 93.163823
## converged
```

```
## # weights:  31
## initial  value 194.801603
## iter  10 value 118.781436
## iter  20 value 113.881205
## iter  30 value 106.656463
## iter  40 value 96.593300
## iter  50 value 96.290766
## iter  60 value 95.654492
## iter  70 value 93.246015
## iter  80 value 93.098078
## iter  90 value 93.090865
## iter 100 value 93.064912
## final  value 93.064912
## stopped after 100 iterations
## # weights:  7
## initial  value 136.529380
## final  value 123.506975
## converged
## # weights:  19
## initial  value 164.890303
## final  value 123.506882
## converged
## # weights:  31
## initial  value 153.401412
## iter  10 value 119.920677
## iter  20 value 112.479734
## iter  30 value 108.184335
## iter  40 value 108.063366
## iter  50 value 108.047728
## iter  60 value 107.922379
## iter  70 value 107.715792
## iter  80 value 107.011899
## iter  90 value 105.021215
## iter 100 value 104.064300
## final  value 104.064300
## stopped after 100 iterations
## # weights:  7
## initial  value 137.171482
## final  value 126.020522
## converged
## # weights:  19
## initial  value 151.824244
## iter  10 value 125.232824
## iter  20 value 125.147383
## iter  30 value 124.401436
## iter  40 value 122.372646
## iter  50 value 121.337913
## iter  60 value 121.223913
## iter  70 value 121.223075
## iter  70 value 121.223074
## iter  70 value 121.223074
## final  value 121.223074
## converged
## # weights:  31
```

```
## initial  value 144.034687
## final  value 126.020522
## converged
## # weights:  7
## initial  value 142.333353
## iter  10 value 125.110893
## iter  20 value 121.030178
## final  value 121.025851
## converged
## # weights:  19
## initial  value 130.060927
## iter  10 value 125.960330
## iter  20 value 121.472133
## iter  30 value 120.877611
## iter  40 value 119.886260
## iter  50 value 115.831419
## iter  60 value 115.574451
## final  value 115.574222
## converged
## # weights:  31
## initial  value 226.976882
## iter  10 value 126.102745
## iter  20 value 125.268329
## iter  30 value 119.267854
## iter  40 value 117.885938
## iter  50 value 114.914380
## iter  60 value 113.524104
## iter  70 value 112.169194
## iter  80 value 112.147911
## iter  90 value 112.089891
## iter 100 value 112.034237
## final  value 112.034237
## stopped after 100 iterations
## # weights:  7
## initial  value 203.713063
## final  value 126.020669
## converged
## # weights:  19
## initial  value 193.288249
## final  value 126.021198
## converged
## # weights:  31
## initial  value 135.090925
## final  value 126.020999
## converged
## # weights:  19
## initial  value 355.240520
## iter  10 value 249.816134
## iter  20 value 246.508946
## iter  30 value 243.422931
## iter  40 value 240.494759
## iter  50 value 222.575139
## iter  60 value 218.546311
## iter  70 value 217.199204
```

```
## final  value 217.175888
## converged
```

```
d <- proc.time() - s  #end time
print(nnet_model)
```

```
## Neural Network
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results across tuning parameters:
##
##   size  decay  ROC        Sens       Spec
##   1     0e+00  0.5854589  1.0000000  0.00000000
##   1     1e-04  0.5854589  1.0000000  0.00000000
##   1     1e-01  0.6692815  1.0000000  0.00000000
##   3     0e+00  0.6033991  1.0000000  0.01754386
##   3     1e-04  0.5861600  1.0000000  0.00000000
##   3     1e-01  0.7293343  0.9126506  0.40925590
##   5     0e+00  0.5854589  1.0000000  0.00000000
##   5     1e-04  0.6096072  0.9849398  0.03448276
##   5     1e-01  0.7037642  0.9066265  0.33030853
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

```
print(nnet_model$finalModel)
```

```
## a 4-3-1 network with 19 weights
## inputs: R F M T
## output(s): .outcome
## options were - entropy fitting  decay=0.1
```

```
nn_pred_res <- predict(nnet_model, lab3_test)
correct <- 0
for (i in (1:length(nn_pred_res))) {
    if (nn_pred_res[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
nn_test_acc <- (correct)/length(nn_pred_res)

ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Neural Network", nn_test_acc,
    format(as.numeric(d)[3]/60, digits = 2))
```

Using Neural Network I got accuraccy 0.8305648

Neural Network Model

```
print(nnet_model)
```

```
## Neural Network
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results across tuning parameters:
##
##   size  decay  ROC        Sens       Spec
##   1     0e+00  0.5854589  1.0000000  0.00000000
##   1     1e-04  0.5854589  1.0000000  0.00000000
##   1     1e-01  0.6692815  1.0000000  0.00000000
##   3     0e+00  0.6033991  1.0000000  0.01754386
##   3     1e-04  0.5861600  1.0000000  0.00000000
##   3     1e-01  0.7293343  0.9126506  0.40925590
##   5     0e+00  0.5854589  1.0000000  0.00000000
##   5     1e-04  0.6096072  0.9849398  0.03448276
##   5     1e-01  0.7037642  0.9066265  0.33030853
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

```
confusionMatrix(nn_pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  222  35
##        Yes  16  28
##
##                Accuracy : 0.8306
##                  95% CI : (0.7833, 0.8712)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.04898
##
##                   Kappa : 0.4243
##  Mcnemar's Test P-Value : 0.01172
##
##             Sensitivity : 0.9328
##             Specificity : 0.4444
##          Pos Pred Value : 0.8638
##          Neg Pred Value : 0.6364
##              Prevalence : 0.7907
##          Detection Rate : 0.7375
##    Detection Prevalence : 0.8538
##       Balanced Accuracy : 0.6886
```

```
##
##        'Positive' Class : No
##
```

KNN

```
knn_test_acc <- 0
control = trainControl(method = "cv", number = 2, summaryFunction = twoClassSummary,
    classProbs = TRUE)
s = proc.time()
knn_model = train(x, as.factor(y), method = "knn", trControl = control, metric = "ROC",
    tuneLength = 3, preProc = c("center"))
d = proc.time() - s
knn_pred_res <- predict(knn_model, lab3_test)
correct <- 0
for (i in (1:length(knn_pred_res))) {
    if (knn_pred_res[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
knn_test_acc <- (correct)/length(knn_pred_res)

ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("KNN", knn_test_acc, format(as.numeric(d)[3]/60,
    digits = 2))
```

Using KNN I got accuraccy 0.7840532

KNN Model

```
print(knn_model)
```

```
## k-Nearest Neighbors
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results across tuning parameters:
##
##   k  ROC        Sens       Spec
##   5  0.5644201  0.8644578  0.2522686
##   7  0.5616850  0.9156627  0.1479129
##   9  0.5652866  0.8945783  0.1733212
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
confusionMatrix(knn_pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  No Yes
##        No  229  56
##        Yes   9   7
##
##                Accuracy : 0.7841
##                  95% CI : (0.7332, 0.8292)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.6429
##
##                   Kappa : 0.101
##  Mcnemar's Test P-Value : 1.159e-08
##
##             Sensitivity : 0.9622
##             Specificity : 0.1111
##          Pos Pred Value : 0.8035
##          Neg Pred Value : 0.4375
##              Prevalence : 0.7907
##          Detection Rate : 0.7608
##    Detection Prevalence : 0.9468
##       Balanced Accuracy : 0.5366
##
##        'Positive' Class : No
##
```

Logistic Regression

```
lr_test_acc <- 0
s = proc.time()
lr_model = train(x, as.factor(y), method = "glm", trControl = control, metric = "ROC",
    tuneLength = 3, preProc = c("center"))
d = proc.time() - s
print(lr_model)
```

```
## Generalized Linear Model
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7124073  0.9759036  0.09558379
```

```
lr_pred_res <- predict(lr_model, lab3_test)
correct <- 0
for (i in (1:length(lr_pred_res))) {
    if (lr_pred_res[i] == lab3_test$Class[i]) {
```

```
        correct = correct + 1
    }
}
lr_test_acc <- (correct)/length(lr_pred_res)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("LogisticRegression", lr_test_acc,
    format(as.numeric(d)[3]/60, digits = 2))
```

Using Logistic Regression I got accuraccy 0.807309

Logistic Regression Model

```
print(lr_model)
```

```
## Generalized Linear Model
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results:
##
##   ROC        Sens       Spec
##   0.7124073  0.9759036  0.09558379
```

```
confusionMatrix(lr_pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  232  52
##        Yes   6  11
##
##                Accuracy : 0.8073
##                  95% CI : (0.7581, 0.8503)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.2646
##
##                   Kappa : 0.2042
##  Mcnemar's Test P-Value : 3.446e-09
##
##             Sensitivity : 0.9748
##             Specificity : 0.1746
##          Pos Pred Value : 0.8169
##          Neg Pred Value : 0.6471
##              Prevalence : 0.7907
##          Detection Rate : 0.7708
##    Detection Prevalence : 0.9435
##       Balanced Accuracy : 0.5747
##
##        'Positive' Class : No
```

##

Naive Bayes

```
nb_test_acc <- 0
s = proc.time()
nb_model = train(Class ~ ., data = lab3_train, method = "nb", trControl = control,
    metric = "ROC", tuneLength = 3, preProc = c("center"))
d = proc.time() - s
nb_pred_res <- predict(nb_model, lab3_test)
correct <- 0
for (i in (1:length(nb_pred_res))) {
    if (nb_pred_res[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
nb_test_acc <- (correct)/length(nb_pred_res)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Naive Bayes", nb_test_acc,
    format(as.numeric(d)[3]/60, digits = 2))
```

Using Naive Bayes I got accuraccy 0.8006645

Naive Bayes Model

```
print(nb_model)
```

```
## Naive Bayes
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 223, 224
## Resampling results across tuning parameters:
##
##   usekernel  ROC        Sens       Spec
##   FALSE      0.6757365  0.9307229  0.1476104
##    TRUE      0.6691056  0.8584337  0.4177253
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE
##  and adjust = 1.
```

```
confusionMatrix(nb_pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  No Yes
##        No  226  48
##        Yes  12  15
##
##                Accuracy : 0.8007
##                  95% CI : (0.751, 0.8443)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.3662
##
##                   Kappa : 0.2376
##  Mcnemar's Test P-Value : 6.228e-06
##
##             Sensitivity : 0.9496
##             Specificity : 0.2381
##          Pos Pred Value : 0.8248
##          Neg Pred Value : 0.5556
##              Prevalence : 0.7907
##          Detection Rate : 0.7508
##    Detection Prevalence : 0.9103
##       Balanced Accuracy : 0.5938
##
##        'Positive' Class : No
##
```

Decision Tree

```
dt_test_acc <- 0
s = proc.time()
dt_model = train(Class ~ ., data = lab3_train, method = "rpart", trControl = control,
    metric = "ROC", tuneLength = 3, preProc = c("center"))
d = proc.time() - s
dt_pred_res <- predict(dt_model, lab3_test)
correct <- 0
for (i in (1:length(dt_pred_res))) {
    if (dt_pred_res[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
dt_test_acc <- (correct)/length(dt_pred_res)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Decision Tree", dt_test_acc,
    format(as.numeric(d)[3]/60, digits = 2))
ind_resultsTab %>% knitr::kable(caption = "Training individual classifiers")
```

Table 3: Training individual classifiers

Model	Acc_test	Time in min
Neural Network	0.830564784053156	0.012
KNN	0.784053156146179	0.012
LogisticRegression	0.807308970099668	0.0078
Naive Bayes	0.800664451827243	0.026
Decision Tree	0.817275747508306	0.011

```
# add rf and ada
rf_preds <- predict(best_rf_model, lab3_test)
rf_test_acc <- 0
correct <- 0
for (i in (1:length(rf_preds))) {
    if (rf_preds[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
rf_test_acc <- (correct)/length(rf_preds)

ada_preds <- predict(best_ada_model, lab3_test)
ada_test_acc <- 0
correct <- 0
for (i in (1:length(ada_preds))) {
    if (ada_preds[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
ada_test_acc <- (correct)/length(ada_preds)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Best RF", rf_test_acc, rf_resultsTab[which.max(rf_resul
    ]$`Time in min`)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Best ADA", ada_test_acc, ada_resultsTab[which.max(ada_r
    ]$`Time in min`)
```

Using Decision Tree I got accuraccy 0.8172757

Decision Tree Model

```
print(dt_model)
```

```
## CART
##
## 447 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 224, 223
## Resampling results across tuning parameters:
##
##   cp          ROC        Sens       Spec
##   0.00000000  0.6977452  0.9277108  0.3297036
##   0.02173913  0.6977452  0.9277108  0.3297036
##   0.04927536  0.6035262  0.9457831  0.2068966
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02173913.
```

```
confusionMatrix(dt_pred_res, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  No Yes
##        No  222  39
##        Yes  16  24
##
##                Accuracy : 0.8173
##                  95% CI : (0.7689, 0.8593)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.143479
##
##                   Kappa : 0.3624
##  Mcnemar's Test P-Value : 0.003012
##
##             Sensitivity : 0.9328
##             Specificity : 0.3810
##          Pos Pred Value : 0.8506
##          Neg Pred Value : 0.6000
##              Prevalence : 0.7907
##          Detection Rate : 0.7375
##    Detection Prevalence : 0.8671
##       Balanced Accuracy : 0.6569
##
##        'Positive' Class : No
##
```

Best Individual Model

The model with highest accuraccy on test set is Neural Network, 0.830564784053156, 0.012

Unweighted Ensembl Classifier

I used majority voting among the five classifiers. The code is attached below.

```
# combine predictions
comm_preds <- data.frame(dt_pred_res, nb_pred_res, lr_pred_res, knn_pred_res,
    nn_pred_res)

# do majority voting
require(functional)
maj_vot_5 <- as.factor(apply(comm_preds, 1, Compose(table, function(i) i ==
    max(i), which, names, function(i) paste0(i, collapse = "/"))))
maj_vot_5_acc <- 0
correct <- 0
for (i in (1:length(maj_vot_5))) {
    if (maj_vot_5[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
maj_vot_5_acc <- (correct)/length(maj_vot_5)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Majority voting", maj_vot_5_acc,
    0)
```

Accuraccy acheived by majority voting is 0.8272425

Majority voting confusion matrix

```
confusionMatrix(maj_vot_5, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  231  45
##        Yes   7  18
##
##                Accuracy : 0.8272
##                  95% CI : (0.7797, 0.8682)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.066
##
##                   Kappa : 0.3293
##  Mcnemar's Test P-Value : 2.882e-07
##
##             Sensitivity : 0.9706
##             Specificity : 0.2857
##          Pos Pred Value : 0.8370
##          Neg Pred Value : 0.7200
##              Prevalence : 0.7907
##          Detection Rate : 0.7674
##    Detection Prevalence : 0.9169
##       Balanced Accuracy : 0.6282
##
##        'Positive' Class : No
##
```

Weighted Ensembl Classifier

Here i took the weighted mean of outputs from each classifier. The weights were propotional to each classifiers accuraccy on the test set. I used the softmax function to calculate weight of each classifier based on its accuraccy.

```
## weighted voting get prob out puts
nn_preds_prob <- predict(nnet_model, lab3_test, type = "prob")
knn_preds_prob <- predict(knn_model, lab3_test, type = "prob")
lr_preds_prob <- predict(lr_model, lab3_test, type = "prob")
nb_preds_prob <- predict(nb_model, lab3_test, type = "prob")
dt_preds_prob <- predict(dt_model, lab3_test, type = "prob")

# get weights def softmax
softmax <- function(x) exp(x)/sum(exp(x))
total_wt = dt_test_acc + nb_test_acc + knn_test_acc + lr_test_acc + nn_test_acc
wt_softmax <- softmax(c(dt_test_acc, nb_test_acc, knn_test_acc, lr_test_acc,
    nn_test_acc))
dt_wt <- wt_softmax[1]
nb_wt <- wt_softmax[2]
knn_wt <- wt_softmax[3]
lr_wt <- wt_softmax[4]
```

```
nn_wt <- wt_softmax[5]

## combine all prob of NO in a dataframe
comm_preds_prob <- data.frame(nn_preds_prob[, 1] * nn_wt, knn_preds_prob[, 1] *
    knn_wt, lr_preds_prob[, 1] * lr_wt, nb_preds_prob[, 1] * nb_wt, dt_preds_prob[,
    1] * dt_wt)
wt_comm_preds <- as.factor(ifelse(rowSums(comm_preds_prob) > 0.5, "No", "Yes"))

wt_maj_vot_5_acc <- 0
correct <- 0
for (i in (1:length(wt_comm_preds))) {
    if (wt_comm_preds[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
wt_maj_vot_5_acc <- (correct)/length(wt_comm_preds)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Weighted Majority voting",
    wt_maj_vot_5_acc, 0)
ind_resultsTab %>% knitr::kable(caption = "Accuraccy comparision of indiviual and Ensembl classifiers")
```

Table 4: Accuraccy comparision of indiviual and Ensembl classifiers

Model	Acc_test	Time in min
Neural Network	0.830564784053156	0.012
KNN	0.784053156146179	0.012
LogisticRegression	0.807308970099668	0.0078
Naive Bayes	0.800664451827243	0.026
Decision Tree	0.817275747508306	0.011
Best RF	0.820598006644518	0.034
Best ADA	0.830564784053156	0.15
Majority voting	0.827242524916944	0
Weighted Majority voting	0.817275747508306	0

Accuraccy acheived by Weighted Ensemble is 0.8172757

Weighted Ensemble confusion matrix

```
confusionMatrix(wt_comm_preds, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  230  47
##        Yes   8  16
##
##                Accuracy : 0.8173
##                  95% CI : (0.7689, 0.8593)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.1435
##
```

```
##                   Kappa : 0.2853
##  Mcnemar's Test P-Value : 2.992e-07
##
##             Sensitivity : 0.9664
##             Specificity : 0.2540
##          Pos Pred Value : 0.8303
##          Neg Pred Value : 0.6667
##              Prevalence : 0.7907
##          Detection Rate : 0.7641
##    Detection Prevalence : 0.9203
##       Balanced Accuracy : 0.6102
##
##        'Positive' Class : No
##
```

Conclusion

We expect that weighted majority voting should be the best among all the classifiers as ensembl classifiers can combine knowledge from different classifiers. From Table 3. we can see that the overall highest accuracy was acheived by Neural Network, 0.830564784053156, 0.012.

Solution 3

I added my best RF and AdaBoost models to the ensembl learners.

Unweighted Ensembl Classifier with Seven models

I used majority voting among all the seven classifiers. The code is attached below.

```
# combine predictions
comm_preds <- data.frame(dt_pred_res, nb_pred_res, lr_pred_res, knn_pred_res,
    nn_pred_res, rf_preds, ada_preds)

# do majority voting
require(functional)
maj_vot_7 <- as.factor(apply(comm_preds, 1, Compose(table, function(i) i ==
    max(i), which, names, function(i) paste0(i, collapse = "/"))))
maj_vot_7_acc <- 0
correct <- 0
for (i in (1:length(maj_vot_7))) {
    if (maj_vot_7[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
maj_vot_7_acc <- (correct)/length(maj_vot_7)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Majority voting_ALL", maj_vot_7_acc,
    0)
ind_resultsTab %>% knitr::kable(caption = "Accuraccy comparision of indiviual and Ensembl classifiers (s
```

Table 5: Accuraccy comparision of indiviual and Ensembl classifiers (seven classifiers)

Model	Acc_test	Time in min
Neural Network	0.830564784053156	0.012
KNN	0.784053156146179	0.012
LogisticRegression	0.807308970099668	0.0078
Naive Bayes	0.800664451827243	0.026
Decision Tree	0.817275747508306	0.011
Best RF	0.820598006644518	0.034
Best ADA	0.830564784053156	0.15
Majority voting	0.827242524916944	0
Weighted Majority voting	0.817275747508306	0
Majority voting_ALL	0.843853820598007	0

Accuraccy acheived by majority voting is 0.8272425

Majority voting confusion matrix

```
confusionMatrix(maj_vot_7, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  228  37
##        Yes  10  26
##
##                Accuracy : 0.8439
##                  95% CI : (0.7978, 0.883)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.0120165
##
##                   Kappa : 0.44
##  Mcnemar's Test P-Value : 0.0001491
##
##             Sensitivity : 0.9580
##             Specificity : 0.4127
##          Pos Pred Value : 0.8604
##          Neg Pred Value : 0.7222
##              Prevalence : 0.7907
##          Detection Rate : 0.7575
##    Detection Prevalence : 0.8804
##       Balanced Accuracy : 0.6853
##
##        'Positive' Class : No
##
```

Weighted Ensembl Classifier

Here i took the weighted mean of outputs from each classifier. The weights were propotional to each classifiers accuraccy on the test set. I used the softmax function to calculate weight of each classifier based on its accuraccy. Table 4 shows the weights used to weight each classifier.

```
## weighted voting get prob out puts
nn_preds_prob <- predict(nnet_model, lab3_test, type = "prob")
knn_preds_prob <- predict(knn_model, lab3_test, type = "prob")
lr_preds_prob <- predict(lr_model, lab3_test, type = "prob")
nb_preds_prob <- predict(nb_model, lab3_test, type = "prob")
dt_preds_prob <- predict(dt_model, lab3_test, type = "prob")
rf_preds_prob <- predict(rf_model, lab3_test, type = "prob")
ada_preds_prob <- predict(ada_model, lab3_test, type = "prob")

# get weights def softmax
softmax <- function(x) exp(x)/sum(exp(x))
wt_softmax <- softmax(c(dt_test_acc, nb_test_acc, knn_test_acc, lr_test_acc,
    nn_test_acc, ada_test_acc, rf_test_acc))
dt_wt <- wt_softmax[1]
nb_wt <- wt_softmax[2]
knn_wt <- wt_softmax[3]
lr_wt <- wt_softmax[4]
nn_wt <- wt_softmax[5]
ada_wt <- wt_softmax[6]
rf_wt <- wt_softmax[7]

## combine all prob of NO in a dataframe
comm_preds_prob <- data.frame(nn_preds_prob[, 1] * nn_wt, knn_preds_prob[, 1] *
    knn_wt, lr_preds_prob[, 1] * lr_wt, nb_preds_prob[, 1] * nb_wt, dt_preds_prob[,
    1] * dt_wt, ada_preds_prob[, 1] * ada_wt, rf_preds_prob[, 1] * rf_wt)
wt_comm_preds <- as.factor(ifelse(rowSums(comm_preds_prob) > 0.5, "No", "Yes"))

wt_maj_vot_7_acc <- 0
correct <- 0
for (i in (1:length(wt_comm_preds))) {
    if (wt_comm_preds[i] == lab3_test$Class[i]) {
        correct = correct + 1
    }
}
wt_maj_vot_7_acc <- (correct)/length(wt_comm_preds)
ind_resultsTab[nrow(ind_resultsTab) + 1, ] <- c("Weighted Majority voting_ALL",
    wt_maj_vot_7_acc, 0)
ind_resultsTab %>% knitr::kable(caption = "Accuraccy comparision of indiviual and Ensembl classifiers")
```

Table 6: Accuraccy comparision of indiviual and Ensembl classifiers

Model	Acc_test	Time in min
Neural Network	0.830564784053156	0.012
KNN	0.784053156146179	0.012
LogisticRegression	0.807308970099668	0.0078
Naive Bayes	0.800664451827243	0.026
Decision Tree	0.817275747508306	0.011
Best RF	0.820598006644518	0.034

Model	Acc_test	Time in min
Best ADA	0.830564784053156	0.15
Majority voting	0.827242524916944	0
Weighted Majority voting	0.817275747508306	0
Majority voting_ALL	0.843853820598007	0
Weighted Majority voting_ALL	0.820598006644518	0

Accuraccy acheived by weighted voting using all models is 0.820598

Weighted Ensemble confusion matrix using all models

```
confusionMatrix(wt_comm_preds, lab3_test[, 5])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  229  45
##        Yes   9  18
##
##                Accuracy : 0.8206
##                  95% CI : (0.7725, 0.8623)
##     No Information Rate : 0.7907
##     P-Value [Acc > NIR] : 0.1129
##
##                   Kappa : 0.3138
##  Mcnemar's Test P-Value : 1.908e-06
##
##             Sensitivity : 0.9622
##             Specificity : 0.2857
##          Pos Pred Value : 0.8358
##          Neg Pred Value : 0.6667
##              Prevalence : 0.7907
##          Detection Rate : 0.7608
##    Detection Prevalence : 0.9103
##       Balanced Accuracy : 0.6239
##
##        'Positive' Class : No
##
```

```
header4 <- c("Model", "Weight")
wt_resultsTab <- data.frame(matrix(ncol = 2, nrow = 0))
colnames(wt_resultsTab) <- header4
wt_resultsTab[1, ] <- c("Neural Net", nn_wt)
wt_resultsTab[2, ] <- c("KNN", knn_wt)
wt_resultsTab[3, ] <- c("Naive Bayes", nb_wt)
wt_resultsTab[4, ] <- c("Logistic Regression", lr_wt)
wt_resultsTab[5, ] <- c("Decision Tree", dt_wt)
wt_resultsTab[6, ] <- c("ADA", ada_wt)
wt_resultsTab[7, ] <- c("RF", rf_wt)
wt_resultsTab %>% knitr::kable(caption = "Weights assigned to different classifiers in weighted Ensembl
```

Table 7: Weights assigned to different classifiers in weighted Ensemble method

Model	Weight
Neural Net	0.145370181289588
KNN	0.138763609682946
Naive Bayes	0.141087904360259
Logistic Regression	0.142028486917264
Decision Tree	0.14345113104402
ADA	0.145370181289588
RF	0.143928505416335

Overall the maximum accuraccy was acheived by Majority voting_ALL, 0.843853820598007, 0.

Conclusion

We expect the performance of weighted ensemble to be the best. When running the code I found that most of the time majority voting over all classifiers and weighted ensemble of all classifier performed best. Thus, we can say an ensemble of classifiers are better than using individual classifiers. I did not see much difference between weighted and un-weighted ensemble classifiers. This may be because of the weigths i am assiging to the classifiers. My weights are directly propotional to the classification accuraccy of individual classifiers. Since, all classifiers perform with almost same accuraccy, my weights are almost same for each classifiers which is almost like having a majority voting. If I decide to implement a more sophisticated weight scheme I would see some more difference in classification.