## Module 5: Assignment 1 – Parameter Selection, Neural Networks, and Ensembles Assignment

# Lea Shipley

options(tidyverse.quiet = TRUE)  
library(tidyverse)  
library(caret)

## Warning: package 'caret' was built under R version 3.6.2

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(nnet)  
library(rpart)  
library(caretEnsemble)

## Warning: package 'caretEnsemble' was built under R version 3.6.2

##   
## Attaching package: 'caretEnsemble'

## The following object is masked from 'package:ggplot2':  
##   
## autoplot

library(ranger)

## Warning: package 'ranger' was built under R version 3.6.2

parole = read\_csv("parole.csv")

## Parsed with column specification:  
## cols(  
## male = col\_double(),  
## race = col\_double(),  
## age = col\_double(),  
## state = col\_double(),  
## time.served = col\_double(),  
## max.sentence = col\_double(),  
## multiple.offenses = col\_double(),  
## crime = col\_double(),  
## violator = col\_double()  
## )

parole = parole %>% mutate(male = as\_factor(as.numeric(male))) %>%  
mutate(male = fct\_recode(male,"male" = "1","female" = "0")) %>%  
 mutate(race = as\_factor(as.numeric(race))) %>%  
 mutate(race = fct\_recode(race, "white" = "1", "other" = "2")) %>%  
 mutate(state = as\_factor(as.numeric(state))) %>%  
 mutate(state = fct\_recode(state, "Kentucky" = "2", "Louisiana" = "3", "Virginia" = "4", "other" = "1")) %>%  
 mutate(crime = as\_factor(as.numeric(crime))) %>%  
 mutate(crime = fct\_recode(crime, "larceny" = "2", "drugs" = "3", "driving" = "4", "other" = "1")) %>%  
 mutate(multiple.offenses = as\_factor(as.numeric(multiple.offenses))) %>%  
 mutate(multiple.offenses = fct\_recode(multiple.offenses, "yes" = "1", "no" = "0")) %>%  
 mutate(violator = as.factor(as.numeric(violator))) %>%   
 mutate(violator = fct\_recode(violator, "yes" = "1", "no" = "0"))

**Task 1**

set.seed(12345)   
train.rows = createDataPartition(y = parole$violator, p=0.7, list = FALSE) #70% in training  
train = parole[train.rows,]   
test = parole[-train.rows,]

**Task 2**

start\_time = Sys.time() #for timing  
fitControl = trainControl(method = "cv",   
 number = 10)  
  
nnetGrid = expand.grid(size = 12, decay = 0.1)  
set.seed(1234)  
nnetBasic = train(x=as.data.frame(train[,-9]), y=train$violator,  
 method = "nnet",  
 trControl = fitControl,  
 tuneGrid = nnetGrid,  
 verbose = FALSE,  
 trace = FALSE)  
  
end\_time = Sys.time()  
end\_time-start\_time

## Time difference of 2.685116 secs

nnetBasic

## Neural Network   
##   
## 473 samples  
## 8 predictor  
## 2 classes: 'no', 'yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 425, 426, 426, 426, 425, 427, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.8711301 0.2265754  
##   
## Tuning parameter 'size' was held constant at a value of 12  
##   
## Tuning parameter 'decay' was held constant at a value of 0.1

**Task 3**

predNetBasic = predict(nnetBasic, train)

confusionMatrix(predNetBasic, train$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 415 23  
## yes 3 32  
##   
## Accuracy : 0.945   
## 95% CI : (0.9205, 0.9638)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 3.851e-06   
##   
## Kappa : 0.6824   
##   
## Mcnemar's Test P-Value : 0.0001944   
##   
## Sensitivity : 0.58182   
## Specificity : 0.99282   
## Pos Pred Value : 0.91429   
## Neg Pred Value : 0.94749   
## Prevalence : 0.11628   
## Detection Rate : 0.06765   
## Detection Prevalence : 0.07400   
## Balanced Accuracy : 0.78732   
##   
## 'Positive' Class : yes   
##

The model has an good accuracy rate of 0.945, which is higher than the No Information Rate of 0.8837.

**Task 4**

start\_time = Sys.time() #for timing  
fitControl = trainControl(method = "cv",   
 number = 10)  
  
nnetGrid = expand.grid(size = seq(from = 1, to = 12, by = 1), #rule of thumb --> between # of input and # of output layers  
 decay = seq(from = 0.1, to = 0.5, by = 0.1))  
set.seed(1234)  
nnetFit = train(x=as.data.frame(train[,-9]), y=train$violator,  
 method = "nnet",  
 trControl = fitControl,  
 tuneGrid = nnetGrid,  
 verbose = FALSE,  
 trace = FALSE)  
  
end\_time = Sys.time()  
end\_time-start\_time

## Time difference of 1.031449 mins

**Task 5**

predNet = predict(nnetFit, train)

confusionMatrix(predNet, train$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 415 48  
## yes 3 7  
##   
## Accuracy : 0.8922   
## 95% CI : (0.8607, 0.9187)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 0.3127   
##   
## Kappa : 0.1863   
##   
## Mcnemar's Test P-Value : 7.218e-10   
##   
## Sensitivity : 0.12727   
## Specificity : 0.99282   
## Pos Pred Value : 0.70000   
## Neg Pred Value : 0.89633   
## Prevalence : 0.11628   
## Detection Rate : 0.01480   
## Detection Prevalence : 0.02114   
## Balanced Accuracy : 0.56005   
##   
## 'Positive' Class : yes   
##

Accuracy on this model is 0.8922, which is less accurate than the previous model.

**Task 6**

predNetBasic\_test = predict(nnetBasic, test)

confusionMatrix(predNetBasic\_test, test$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 172 15  
## yes 7 8  
##   
## Accuracy : 0.8911   
## 95% CI : (0.8398, 0.9305)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.4672   
##   
## Kappa : 0.3639   
##   
## Mcnemar's Test P-Value : 0.1356   
##   
## Sensitivity : 0.34783   
## Specificity : 0.96089   
## Pos Pred Value : 0.53333   
## Neg Pred Value : 0.91979   
## Prevalence : 0.11386   
## Detection Rate : 0.03960   
## Detection Prevalence : 0.07426   
## Balanced Accuracy : 0.65436   
##   
## 'Positive' Class : yes   
##

Accuracy on the testing set is 0.8911, versus 0.945 on the training set.

**Task 7**

predNet\_test = predict(nnetFit, test)

confusionMatrix(predNet\_test, test$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 179 21  
## yes 0 2  
##   
## Accuracy : 0.896   
## 95% CI : (0.8455, 0.9345)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.3797   
##   
## Kappa : 0.1444   
##   
## Mcnemar's Test P-Value : 1.275e-05   
##   
## Sensitivity : 0.086957   
## Specificity : 1.000000   
## Pos Pred Value : 1.000000   
## Neg Pred Value : 0.895000   
## Prevalence : 0.113861   
## Detection Rate : 0.009901   
## Detection Prevalence : 0.009901   
## Balanced Accuracy : 0.543478   
##   
## 'Positive' Class : yes   
##

Accuracy on the testing set is 0.896, which is just slightly higher than the accuracy rate on the training set (0.8922).

**Task 8**

There is some accuracy degradation on the testing set on the nnetBasic model from Task 2, but it’s not terribly severe. There may be some slight overfitting in this model, but the accuracy is still higher than the No Information Rate. The nnetFit model from Task 4 had a slightly higher accuracy on the testing set; there does not appear to be overfitting in the model.

**Task 9**

control = trainControl(  
 method = "cv",  
 number = 5, #to save time, we'll use 5 fold cross-validation rather than 10  
 savePredictions = "final",  
 classProbs = TRUE, #instructs caret to calculate probabilities (rather than providing final classifications)  
 summaryFunction = twoClassSummary #enables calculation of AUC  
 )

set.seed(111)  
model\_list = caretList(  
 x=as.data.frame(train[,-9]), y=train$violator,  
   
 metric = "ROC", #specify that maximizing AUC is our objective  
 trControl= control, #using the previously defined trControl object  
 methodList=c("glm"), #specifying the model methods to use  
 #A note about logistic regression in caret: Caret does not do any stepwise removal or addition of variables.   
 tuneList = list(rf=caretModelSpec(method="ranger", tuneLength=6),  
rpart = caretModelSpec(method="rpart", tuneLength=6),  
nn = caretModelSpec(method="nnet", tuneLength=6, trace=FALSE)  
)  
 )

## Warning in trControlCheck(x = trControl, y = target): indexes not defined  
## in trControl. Attempting to set them ourselves, so each model in the  
## ensemble will have the same resampling indexes.

as.data.frame(predict(model\_list, newdata=head(train)))

## rf rpart nn glm  
## 1 0.9924078 0.9253012 0.9336210 0.9520893  
## 2 0.9825794 0.9253012 0.9443406 0.8526230  
## 3 0.9894714 0.9253012 0.9503245 0.8616431  
## 4 0.9947948 0.9253012 0.9310976 0.9268397  
## 5 0.8706749 0.9253012 0.9549287 0.9233879  
## 6 0.9633214 0.9253012 0.7650537 0.7418396

modelCor(resamples(model\_list))

## rf rpart nn glm  
## rf 1.0000000 0.6407088 0.3380813 0.3226732  
## rpart 0.6407088 1.0000000 -0.4184480 -0.4400857  
## nn 0.3380813 -0.4184480 1.0000000 0.8664434  
## glm 0.3226732 -0.4400857 0.8664434 1.0000000

The models are correlated with one another. The nnet and glm models are more highly correlated than the others.

ensemble = caretEnsemble(  
 model\_list,   
 metric="ROC",  
 trControl=trainControl(  
 method = "cv", #cross-validation during ensembling  
 number= 5, #number of folds  
 summaryFunction=twoClassSummary,  
 classProbs=TRUE  
 ))

summary(ensemble)

## The following models were ensembled: rf, rpart, nn, glm   
## They were weighted:   
## 2.7009 -3.0446 1.7228 -2.3832 -2.1428  
## The resulting ROC is: 0.8422  
## The fit for each individual model on the ROC is:   
## method ROC ROCSD  
## rf 0.8313566 0.04702462  
## rpart 0.6713712 0.04964340  
## nn 0.8444636 0.02425292  
## glm 0.8386507 0.03410036

The ROC for the ensemble is 0.8422, which is better than the ROC for the random forest, rpart, and glm models. However, the nnet model has a slightly higher ROC (0.8444636) than the ensemble.

#training set  
#training set  
pred\_ensemble = predict(ensemble, train, type = "raw")  
confusionMatrix(pred\_ensemble,train$violator)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 417 21  
## yes 1 34  
##   
## Accuracy : 0.9535   
## 95% CI : (0.9304, 0.9706)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 1.009e-07   
##   
## Kappa : 0.7313   
##   
## Mcnemar's Test P-Value : 5.104e-05   
##   
## Sensitivity : 0.9976   
## Specificity : 0.6182   
## Pos Pred Value : 0.9521   
## Neg Pred Value : 0.9714   
## Prevalence : 0.8837   
## Detection Rate : 0.8816   
## Detection Prevalence : 0.9260   
## Balanced Accuracy : 0.8079   
##   
## 'Positive' Class : no   
##

#testing set  
pred\_ensemble\_test = predict(ensemble, test, type = "raw")  
confusionMatrix(pred\_ensemble\_test,test$violator)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 176 17  
## yes 3 6  
##   
## Accuracy : 0.901   
## 95% CI : (0.8512, 0.9385)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.29683   
##   
## Kappa : 0.3322   
##   
## Mcnemar's Test P-Value : 0.00365   
##   
## Sensitivity : 0.9832   
## Specificity : 0.2609   
## Pos Pred Value : 0.9119   
## Neg Pred Value : 0.6667   
## Prevalence : 0.8861   
## Detection Rate : 0.8713   
## Detection Prevalence : 0.9554   
## Balanced Accuracy : 0.6221   
##   
## 'Positive' Class : no   
##

**Task 10**

stack = caretStack(  
 model\_list, #use the list of models already specified  
 method ="glm", #stack models linearly  
 metric ="ROC", #maximize AUC  
 trControl = trainControl(  
 method = "cv", #k-fold cross-validation  
 number = 5, #5 folds  
 savePredictions = "final",  
 classProbs = TRUE, #save probabilities  
 summaryFunction = twoClassSummary #calculate AUC values  
 )  
)  
  
print(stack)

## A glm ensemble of 4 base models: rf, rpart, nn, glm  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 473 samples  
## 4 predictor  
## 2 classes: 'no', 'yes'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 379, 378, 379, 378, 378   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8423956 0.9713425 0.2363636

The stacked model, like the ensemble model, has a ROC better than the random forest, rpart, and glm models. However, the nnet model has a slightly higher ROC (0.8444636) than the stacked model (0.8423956).

#training set  
pred\_stack = predict(stack, train, type = "raw")  
confusionMatrix(pred\_stack,train$violator)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 417 21  
## yes 1 34  
##   
## Accuracy : 0.9535   
## 95% CI : (0.9304, 0.9706)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 1.009e-07   
##   
## Kappa : 0.7313   
##   
## Mcnemar's Test P-Value : 5.104e-05   
##   
## Sensitivity : 0.9976   
## Specificity : 0.6182   
## Pos Pred Value : 0.9521   
## Neg Pred Value : 0.9714   
## Prevalence : 0.8837   
## Detection Rate : 0.8816   
## Detection Prevalence : 0.9260   
## Balanced Accuracy : 0.8079   
##   
## 'Positive' Class : no   
##

#testing set  
pred\_stack\_test = predict(stack, test, type = "raw")  
confusionMatrix(pred\_stack\_test,test$violator)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 176 17  
## yes 3 6  
##   
## Accuracy : 0.901   
## 95% CI : (0.8512, 0.9385)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.29683   
##   
## Kappa : 0.3322   
##   
## Mcnemar's Test P-Value : 0.00365   
##   
## Sensitivity : 0.9832   
## Specificity : 0.2609   
## Pos Pred Value : 0.9119   
## Neg Pred Value : 0.6667   
## Prevalence : 0.8861   
## Detection Rate : 0.8713   
## Detection Prevalence : 0.9554   
## Balanced Accuracy : 0.6221   
##   
## 'Positive' Class : no   
##

The stacked ensemble model has similar performance to the non-stacked model with accuracy of 0.9535 on the training set, and 0.901 on the testing set.