Data Analytics and Predictions 2019

Final Project: Prediction of Mortality in SAH patients

Seokhun Kim, Babak Soltanalizadeh, Jaehwan Han

Data Pre-processing

1. Merging Encounter and Medication data frames.

```
setwd('D:/UTHealth/2019 Spring/Data Analytics Predictions/Final Project/Data')
encounter=read.csv('encounters.csv')
medication=read.csv('medications.csv')
medication=medication[,-4] #remove the final column (time)
```

(1) For each encounter id, we Counted the number of each medication (frequency) with groupby method.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
medication_count=medication %>%
  group_by(ENCOUNTER_ID, GENERIC_NAME) %>%
  summarise(n = n())
rm(medication)
```

(2) The medication (categorical) variable is one-hot-coded, and each dummy variable was multiplied with the count column.

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
dmy=dummyVars(" ~ .", data=medication_count)
medication_hot=data.frame(predict(dmy, newdata=medication_count))
for(i in names(medication_hot)[2:815]){
  medication_hot[[paste(i)]]=(medication_hot[[i]]*medication_hot$n)
}
medication_hot=medication_hot[,-816]
rm(medication_count, dmy)
```

(3) The one-hot-coded variables were collapsed so that the data frame has only 1 observation for each encounter id.

```
medication_hot=medication_hot %>%
  group_by(ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
```

(4) Finally, the Encounter and Medication data frames were merged.

```
Total_1=inner_join(x=encounter, y=medication_hot, by='ENCOUNTER_ID')
rm(medication_hot)
```

2. Merging the Procedure data frame were accomplished in the same method as above.

```
procedure=read.csv('procedures.csv')
procedure=procedure[,-3] #remove the final column (time)
procedure_count=procedure %>%
  group_by(ENCOUNTER_ID, PROCEDURE_ID) %>%
  summarise(n=n())
rm(procedure)
##one-hot-coding and multiplying with frequency
procedure_count$PROCEDURE_ID=as.factor(procedure_count$PROCEDURE_ID)
dmy=dummyVars(" ~ .", data=procedure_count)
procedure_hot=data.frame(predict(dmy, newdata=procedure_count))
for(i in names(procedure_hot)[2:675]){
  procedure_hot[[paste(i)]]=(procedure_hot[[i]]*procedure_hot$n)
}
procedure_hot=procedure_hot[,-676]
rm(procedure_count, dmy)
##collapsing for each encounter_id
procedure_hot=procedure_hot %>%
  group_by(ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
Total_2=inner_join(x=Total_1, y=procedure_hot, by='ENCOUNTER_ID')
rm(Total_1, procedure_hot)
```

3. Merging the Lab data frame in the same way, but we obtained and used [mean, max, min] instead of frequency.

The final merged data frame had 1370 observations.

```
lab=read.csv('labs.csv')
lab=lab[,-4] #remove the final column (time)
lab_aggregate=lab %>%
  group_by(ENCOUNTER_ID, DETAIL_LAB_PROCEDURE_ID) %>%
  summarise(mean=mean(NUMERIC_RESULT), max=max(NUMERIC_RESULT), min=min(NUMERIC_RESULT))
rm(lab, encounter)
##one-hot-coding and multiplying with mean, max, and min
lab_aggregate$DETAIL_LAB_PROCEDURE_ID=as.factor(lab_aggregate$DETAIL_LAB_PROCEDURE_ID)
dmy=dummyVars(" ~ .", data=lab_aggregate)
lab_hot=data.frame(predict(dmy, newdata=lab_aggregate))
for(i in names(lab_hot)[2:789]){
  lab_hot[[paste0("mean_", i)]]=(lab_hot[[i]]*lab_hot$mean)
}
for(i in names(lab_hot)[2:789]){
  lab_hot[[paste0("max_", i)]]=(lab_hot[[i]]*lab_hot$max)
}
for(i in names(lab_hot)[2:789]){
  lab_hot[[paste0("min_", i)]]=(lab_hot[[i]]*lab_hot$min)
}
lab_hot=lab_hot[,-c(2:792)]
rm(lab_aggregate, dmy)
##collapsing for each encounter_id
lab_hot=lab_hot %>%
  group_by(ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
Total=inner_join(x=Total_2, y=lab_hot, by='ENCOUNTER_ID')
rm(Total_2, lab_hot)
```

4. Due to high multicollnearity between mean, max, and min values of lab data, we dropped max and min.

Also, the outcome variable 'Mortality' was converted into a factor variable.

```
Total=na.omit(Total)
Total=Total[,-1] #removing patient_id
Total=Total[,-c(2284:3860)] #removing mins maxs due to collinearity with means
Total$Mortality=as.factor(Total$Mortality)
```

5. Repeat the all 1-4 steps to merge the test datasets (encounters500, medications500, procedures500, labs500).

```
##### Testing data setup #####
encounter=read.csv('encounters500.csv')
medication=read.csv('medications500.csv')
medication=medication[,-4]

medication_count=medication %>%
  group_by(NEW_ENCOUNTER_ID, GENERIC_NAME) %>%
  summarise(n = n())
rm(medication)

dmy=dummyVars(" ~ .", data=medication_count)
```

```
medication_hot=data.frame(predict(dmy, newdata=medication_count))
for(i in names(medication_hot)[2:624]){
  medication_hot[[paste(i)]]=(medication_hot[[i]]*medication_hot$n)
}
medication_hot=medication_hot[,-625]
rm(medication_count, dmy)

medication_hot=medication_hot %>%
  group_by(NEW_ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
Total_1=inner_join(x=encounter, y=medication_hot, by='NEW_ENCOUNTER_ID')
rm(medication_hot)

procedure=read.csv('procedures500.csv')
procedure=procedure[,-3]
procedure_count=procedure %>%
  group_by(NEW_ENCOUNTER_ID, PROCEDURE_ID) %>%
  summarise(n=n())
rm(procedure)

procedure_count$PROCEDURE_ID=as.factor(procedure_count$PROCEDURE_ID)
dmy=dummyVars(" ~ .", data=procedure_count)
procedure_hot=data.frame(predict(dmy, newdata=procedure_count))
for(i in names(procedure_hot)[2:366]){
  procedure_hot[[paste(i)]]=(procedure_hot[[i]]*procedure_hot$n)
}
procedure_hot=procedure_hot[,-367]
rm(procedure_count, dmy)

procedure_hot=procedure_hot %>%
  group_by(NEW_ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
Total_2=inner_join(x=Total_1, y=procedure_hot, by='NEW_ENCOUNTER_ID')
rm(Total_1, procedure_hot)

lab=read.csv('labs500.csv')
lab=lab[,-4]
lab_aggregate=lab %>%
  group_by(NEW_ENCOUNTER_ID, DETAIL_LAB_PROCEDURE_ID) %>%
  summarise(mean=mean(NUMERIC_RESULT), max=max(NUMERIC_RESULT), min=min(NUMERIC_RESULT))
rm(lab, encounter)

lab_aggregate$DETAIL_LAB_PROCEDURE_ID=as.factor(lab_aggregate$DETAIL_LAB_PROCEDURE_ID)
dmy=dummyVars(" ~ .", data=lab_aggregate)
lab_hot=data.frame(predict(dmy, newdata=lab_aggregate))
for(i in names(lab_hot)[2:644]){
  lab_hot[[paste0("mean_", i)]]=(lab_hot[[i]]*lab_hot$mean)
}
for(i in names(lab_hot)[2:644]){
  lab_hot[[paste0("max_", i)]]=(lab_hot[[i]]*lab_hot$max)
}
for(i in names(lab_hot)[2:644]){
  lab_hot[[paste0("min_", i)]]=(lab_hot[[i]]*lab_hot$min)
```

```
}
lab_hot=lab_hot[,-c(2:647)]
rm(lab_aggregate, dmy)

lab_hot=lab_hot %>%
  group_by(NEW_ENCOUNTER_ID) %>%
  summarise_all(funs(sum))
Total_test=inner_join(x=Total_2, y=lab_hot, by='NEW_ENCOUNTER_ID')
rm(Total_2, lab_hot)

Total_test=na.omit(Total_test)
names(Total_test)[1:10]
```

```
##  [1] "NEW_ENCOUNTER_ID"
##  [2] "AGE_IN_YEARS"
##  [3] "First_VASPRESOR"
##  [4] "RACE"
##  [5] "GENDER"
##  [6] "MARITAL_STATUS"
##  [7] "GENERIC_NAME.abciximab"
##  [8] "GENERIC_NAME.acetaminophen"
##  [9] "GENERIC_NAME.acetaminophen.codeine"
## [10] "GENERIC_NAME.acetaminophen.hydrocodone"
```

```
Total_test=Total_test[,-c(1638:2923)] #removing mins maxs due to collinearity with means
## Add variables (all values=0) that exists in Total but not in Total_test so that the ML models can pr
for (i in names(Total)[!(names(Total) %in% names(Total_test))]){
  Total_test[[i]]=0
}
```

Analyses using Machine Learning Models

0. Importing relevant libraries

```
attach(Total)
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
library(glmnet)
```

```
## Loading required package: Matrix

## Loading required package: foreach
```

```
## Loaded glmnet 2.0-16
```

```
library(tree)
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

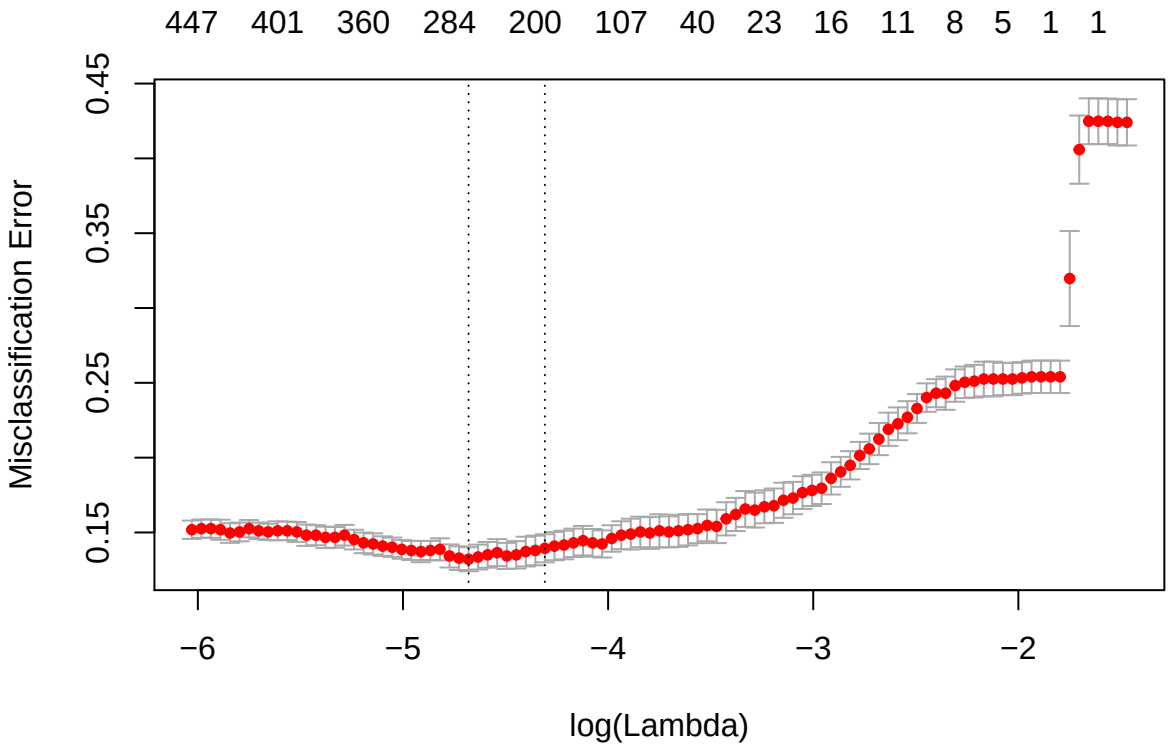
```
library(caret)
library(e1071)
set.seed(1)
```

1. Lasso for variable selection

The # of variables (p=2284) was greater than the # of observations (n=1370), so we performed a Lasso for variable selection.

For the lambda (0.0093) with best CV error rate (13.21%), 255 variables were selected with non-zero coefficients.

```
x=model.matrix(Mortality~.-ENCOUNTER_ID, data=Total)
y=Total$Mortality
grid=10^seq(10,-2,length=100)
lasso.mod=glmnet(x, y, alpha=1, lambda=grid, family='binomial')
cv.lasso.out=cv.glmnet(x, y, alpha=1, family='binomial', type.measure="class")
plot(cv.lasso.out)
```




```
best.lam=cv.lasso.out$lambda.min
best.lam
```

```
## [1] 0.009276003
```

```
##CV error
cv.lasso.error=cv.lasso.out$cvm[cv.lasso.out$lambda==cv.lasso.out$lambda.min]
cv.lasso.error
```

```
## [1] 0.1321168
```

```
##Selected variables with non-zero coefficients
lasso.coef=predict(lasso.mod, type='coefficients', s=best.lam)
#lasso.coef
lasso.nzcoef=lasso.coef[lasso.coef[,1]!=0,]
#lasso.nzcoef
length(lasso.nzcoef)
```

```
## [1] 255
```

```
form1=paste0(names(lasso.nzcoef)[-(1:3)],collapse='+')
form=paste0("Mortality~First_VASPRESOR+MARITAL_STATUS+",form1)
```

(1-1) (optional) 2nd Lasso for smaller variable selection

We tried 1 more Lasso on the selected subset to check if there were any other variables to be dropped.

This 2nd Lasso resulted in only 1 more variable reduction, so we decided to stick to the 1st Lasso subset.

```
#x=model.matrix(as.formula(form), data=Total)
#lasso.mod=glmnet(x, y, alpha=1, lambda=grid, family='binomial')
#cv.lasso.out=cv.glmnet(x, y, alpha=1, family='binomial')
#best.lam=cv.lasso.out$lambda.min
#best.lam
#cv.lasso.error=cv.lasso.out$cvm[cv.lasso.out$lambda==cv.lasso.out$lambda.min]
#cv.lasso.error
#lasso.coef=predict(lasso.mod, type='coefficients', s=best.lam)
#lasso.coef
#lasso.nzcoef=lasso.coef[lasso.coef[,1]!=0,]
#lasso.nzcoef
#length(lasso.nzcoef)
#form2=paste0(names(lasso.nzcoef)[-(1:3)],collapse='+')
#form=paste0("Mortality~First_VASPRESOR+MARITAL_STATUS+",form2)
```

2. Logistic regression

As the first step, we performed a Logistic regression with 255 selected predictors.

The training error rate was 1.53%, and the CV error rate was 12.96%, indicating that the linear classification boundary already works well.

```
glm.fit=glm(formula=as.formula(form), data=Total, family=binomial)
#summary(glm.fit)
#summary(glm.fit)$coef
##training error rate
glm.probs=predict(glm.fit, type='response')
glm.pred=rep(0,length(Total$Mortality))
glm.pred[glm.probs>0.5]=1
table(glm.pred,Total$Mortality)
```

```
##
## glm.pred   0   1
##        0 778  10
##        1  11 571
```

```
mean(glm.pred!=Total$Mortality)
```

```
## [1] 0.01532847
```

```
##CV error rate
cv.glm.error=cv.glm(Total, glm.fit, K=10)
cv.glm.error$delta[1]
```

```
## [1] 0.1296143
```

3. Linear discriminant analysis

To test another linear classification boundary that works well for separated classes, we performed a LDA with 255 predictors.

The training error rate was 3.36%, but the CV error rate was 49.20%, possibly due to multi-collinearity of predictors, violation of equal variance-covariance matrix between the 2 classes, and the not-clear-separation between the 2 classes.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fit=lda(as.formula(form), data=Total)
#lda.fit
##training error
lda.pred=predict(lda.fit)
table(lda.pred$class, Total$Mortality)
```

```
##
##       0   1
##   0 766  23
##   1  23 558
```

```
mean(lda.pred$class!=Total$Mortality)
```

```
## [1] 0.03357664
```

```
##LOOCV error
cv.lda=lda(as.formula(form), data=Total, CV=TRUE)
table(cv.lda$class,Total$Mortality)
```

```
##
##       0   1
##   0 394 279
##   1 395 302
```

```
mean(cv.lda$class!=Total$Mortality)
```

```
## [1] 0.4919708
```

4. Ridge logistic regression

Although the performance of logistic regression was good, we still have 255 predictors for 1370 observations, which could have produced an overfitting of the logistic model. Thus, to reduce the variance of the linear model with slightly higher bias, we performed a Ridge logistic regression.

For the lambda of 0.0175, the CV error rate was 7.52% (i.e., CV accuracy ~ 92.5%), which improved that of logistic regression quite a bit. Additionally, the training error rate was 2.55%.

```
x=model.matrix(as.formula(form), data=Total)
ridge.mod=glmnet(x, y, alpha=0, lambda=grid, family='binomial')
cv.ridge.out=cv.glmnet(x, y, alpha=0, lambda=grid, family='binomial', type.measure="class")
best.lam=cv.ridge.out$lambda.min
best.lam
```

```
## [1] 0.01747528
```

```
##CV error
ridge.coef=predict(ridge.mod, type='coefficients', s=best.lam)
#ridge.coef
cv.ridge.error=cv.ridge.out$cvm[cv.ridge.out$lambda==cv.ridge.out$lambda.min]
cv.ridge.error
```

```
## [1] 0.07518248
```

```
##training error
ridge.pred=rep(0,length(Total$Mortality))
ridge.pred[predict(ridge.mod, type='response', newx=x, s=best.lam)>0.5]=1
table(ridge.pred,Total$Mortality)
```

```
##
## ridge.pred   0   1
##          0 772  18
##          1  17 563
```

```
mean(ridge.pred!=Total$Mortality)
```

```
## [1] 0.02554745
```

Additionally, because this Ridge logistic model is our final model, we predicted mortality for the test data using this model, and generated a csv file.

```
##test data prediction
x.test=model.matrix(as.formula(form), data=Total_test)
ridge.test.pred=rep(0,length(Total_test$NEW_ENCOUNTER_ID))
ridge.test.pred[predict(ridge.mod, type='response', newx=x.test, s=best.lam)>0.5]=1
sum(ridge.test.pred)
```

```
## [1] 150
```

```
test.data=data.frame(NEW_ENCOUNTER_ID=Total_test$NEW_ENCOUNTER_ID, Pred_Mortality=ridge.test.pred)
write.csv(test.data, file='Predicted Mortality.csv')
```

5. Random forest

Although the linear models worked quite well, we also decided to fit nonparametric tree-based models to check if they provide a better prediction performance. We did not try a single tree model with pruning process, or bagging due to their possibility of overfitting.

At first, we performed random forest models with different mtry and ntree values to check which combination produces the best OOB error rate.

The best CV error rate for OOB (12.63%) was achieved for mtry=90 and ntrees=500. Additionally, the training error was 0%, indicating that the model was overfitted. However, note that in spite of overfitting, the CV error was rather good, thanks to the random forest's averaging method across many trees for lower variance.

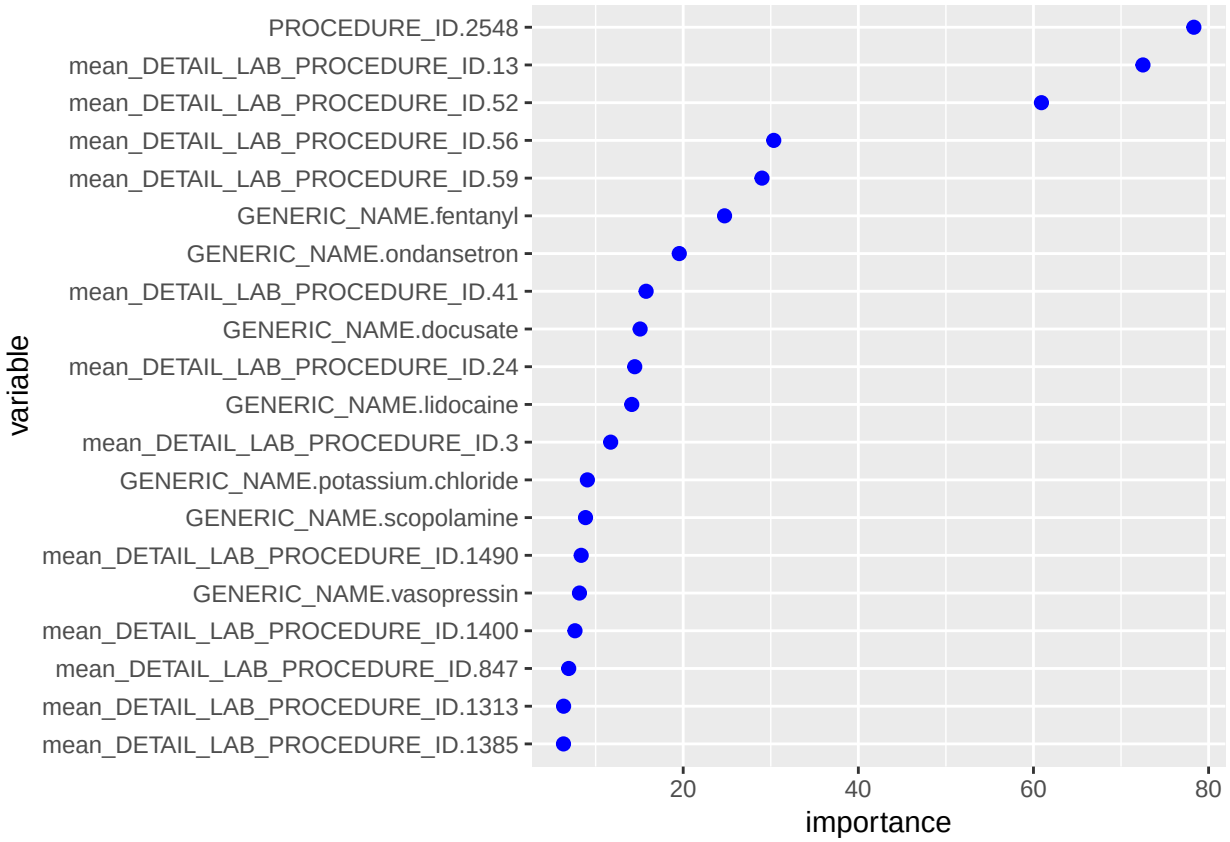
```
cv.rf.error=matrix(NA, 5, 4, dimnames=list(paste(seq(10,100,20)),paste(c(50,100,200,500))))
k=0
for (i in seq(10,100,20)){
  k=k+1
  l=1
  for (j in c(50,100,200,500)){
    rf.fit=randomForest(as.formula(form), data=Total, mtry=i, ntree=j, importance=TRUE)
    cv.rf.error[k,l]=tail(rf.fit$err.rate[,1],n=1)
    l=l+1
  }
}
##CV error
cv.rf.error
```

```
##           50       100       200       500
## 10 0.1379562 0.1394161 0.1328467 0.1291971
## 30 0.1416058 0.1262774 0.1328467 0.1313869
## 50 0.1350365 0.1350365 0.1343066 0.1321168
## 70 0.1350365 0.1262774 0.1270073 0.1306569
## 90 0.1467153 0.1357664 0.1277372 0.1255474
```

```
rf.fit=randomForest(as.formula(form), data=Total, mtry=90, ntree=500, importance=TRUE)
tail(rf.fit$err.rate[,1],n=1)
```

```
## [1] 0.1262774
```

```
#varImpPlot(rf.fit) is modified
var.imp=importance(rf.fit)
var_importance=data_frame(variable=rownames(var.imp), importance=var.imp[,'MeanDecreaseGini'])
var_importance=arrange(var_importance, desc(importance))
var_importance$variable=factor(var_importance$variable, levels=var_importance$variable)
ggplot(var_importance[1:20,], aes(x=importance, y=variable))+
  geom_point(colour="blue",size=2)+scale_y_discrete(limits = unique(rev(var_importance$variable[1:20])))
```




```
##training error
rf.pred=predict(rf.fit, newdata=Total)
table(rf.pred,Total$Mortality)
```

```
##
## rf.pred   0   1
##       0 789   0
##       1   0 581
```

```
mean(rf.pred!=Total$Mortality)
```

```
## [1] 0
```

6. Gradient boosting

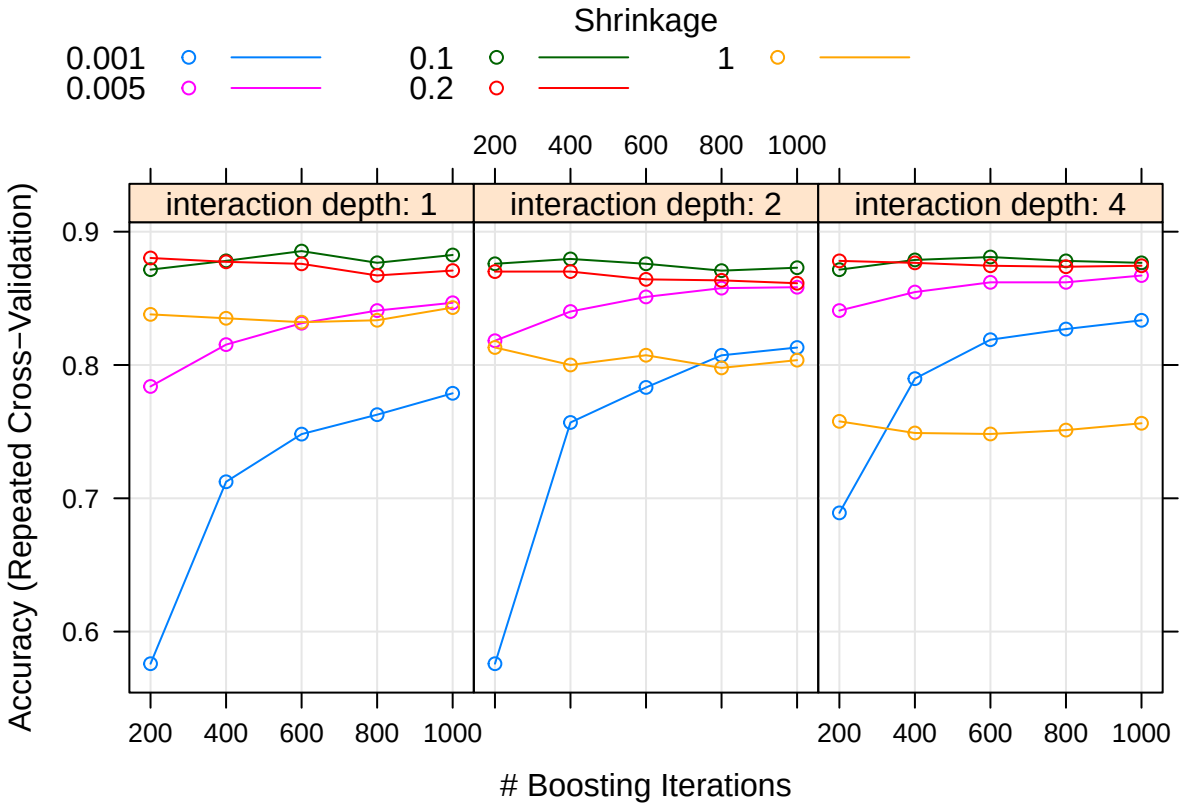
Also, we performed a tree-based gradient boosting to test if this slow learning method works well for the data. We used combinations of shirinkage parameters of [0.001, 0.005, 0.1, 0.2, 1], interaction depths of [1, 2, 4], and ntrees of [200, 400, 600, 800, 1000].

The best CV error (11.46%) was achieved for shrinkage=0.1, int.depth=1, ntrees=600. Additionally, the training error rate was 6.50%.

This gbm method did not overfit the data and is better than random forest, but their performance is not quite better than logistic regression (12.96%; which could be a little overfitted with 255 predictors) and quite worse than Ridge logistic (7.52%; which prevented overfitting to some extent by giving bias).

This finding that nonparametric methods do not perform better than linear models, reinforces the idea that the true classification boundary is close to linearity.

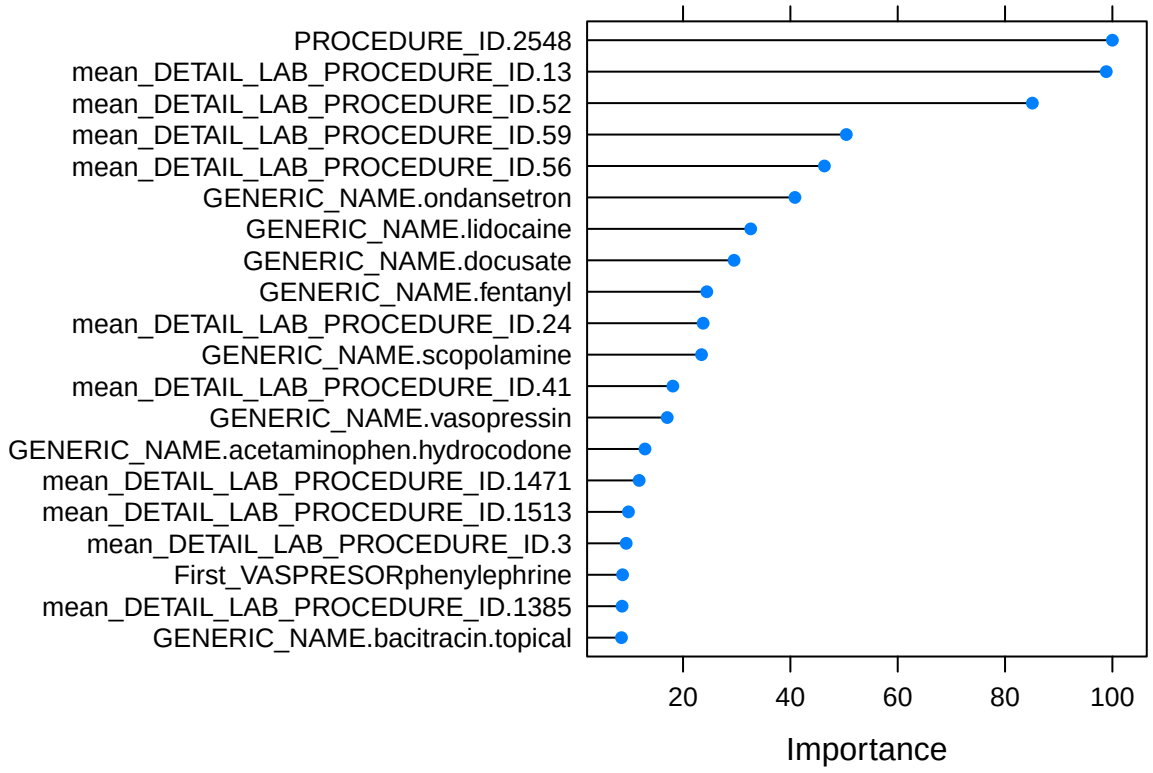
```
fitControl=trainControl(method="repeatedcv", number=10, repeats=1)
gbmGrid=expand.grid(n.trees=seq(2,10,2)*100, interaction.depth=c(1, 2, 4),
                    shrinkage=c(0.001, 0.005, 0.1, 0.2, 1), n.minobsinnode=20)

gbm.form=train(as.formula(form), data=Total, method="gbm", distribution="bernoulli",
               trControl=fitControl, verbose=FALSE, tuneGrid=gbmGrid)
plot(gbm.form)
```




```
##CV error
gbm.form$results[which.max(gbm.form$results$Accuracy),]
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 33       0.1                 1             20     600 0.8854265 0.7647251
##    AccuracySD    KappaSD
## 33 0.02861552 0.05901599
```

```
##Var imprtance plot
var.imp=varImp(gbm.form)
plot(var.imp, top=20)
```




```
##training error
gbm.pred=predict(gbm.form, newdata=Total, n.trees=600)
table(gbm.pred,Total$Mortality)
```

```
##
## gbm.pred   0   1
##        0 751  51
##        1  38 530
```

```
mean(gbm.pred!=Total$Mortality)
```

```
## [1] 0.0649635
```

7. Support vector classifier

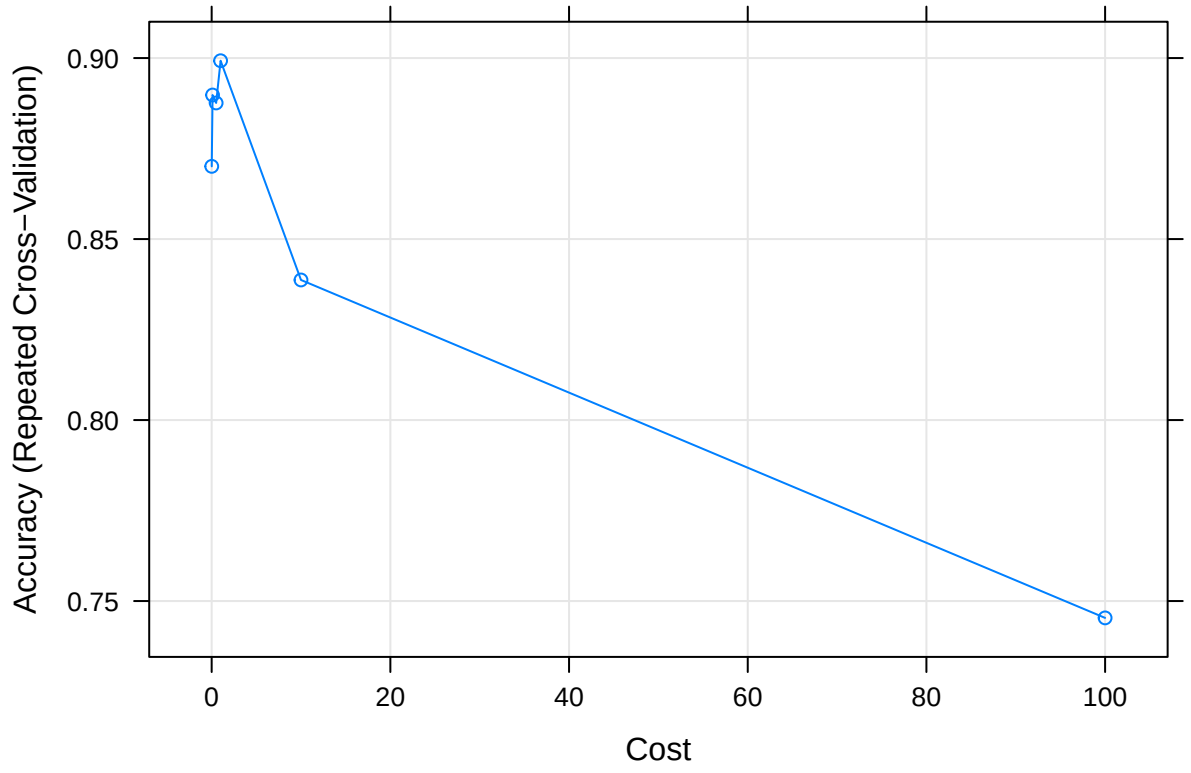
Further, we performed a linear support vector classfier with different costs of [0.01, 0.1, 0.5, 1, 10, 100].

The best CV error (10.81%) was achieved with the cost=0.1. Additionally, the training error was 2.04%.

This result of linear svm is better than that of logistic (12.96%; 1.53%) because it has a larger bias and smaller variance due to the low value of cost (0.1). The participation of many support vectors (277 out of 1370) due to this low cost resulted in a little higher training error and quite smaller CV error than logistic.

However, this result is still worse than that of Ridge logistic regression, which can give a little more bias than this model and reduce quite more variance eventually.

```
svmGrid=expand.grid(C=c(0.01, 0.1, 0.5, 1, 10, 100))
svm.Linear=train(as.formula(form), data=Total, method="svmLinear", trControl=fitControl, tuneGrid=svmGr
plot(svm.Linear)
```




```
##CV error
svm.Linear$results[which.max(svm.Linear$results$Accuracy),]
```

```
##   C  Accuracy     Kappa AccuracySD   KappaSD
## 4 1 0.8992639 0.7930123 0.02454201 0.0515993
```

```
svm.Linear$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 236
##
## Objective Function Value : -63.6112
## Training error : 0.012409
```

```
##training error
svm.Linear.pred=predict(svm.Linear, newdata=Total)
table(svm.Linear.pred,Total$Mortality)
```

```
##
## svm.Linear.pred   0   1
##               0 777   5
##               1  12 576
```

```
mean(svm.Linear.pred!=Total$Mortality)
```

```
## [1] 0.01240876
```

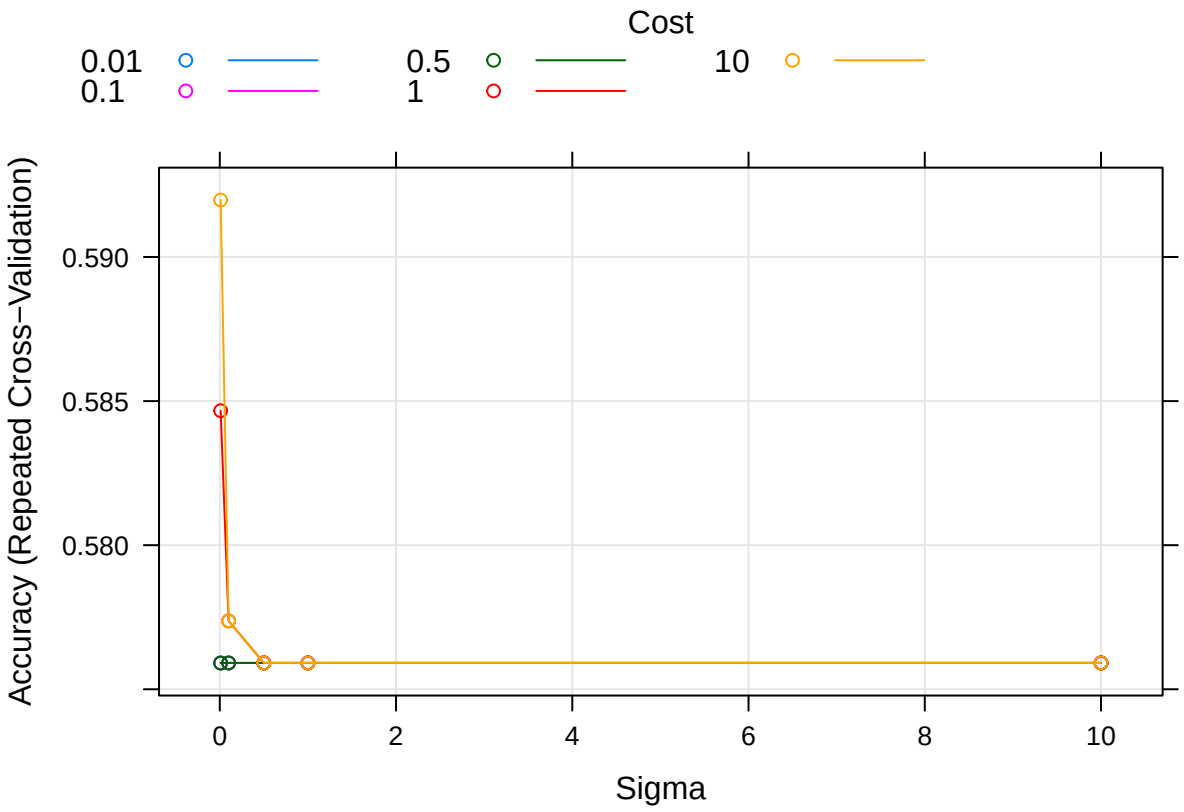
8. Support vector machine (using Radial kernel)

Finally, we performed a nonlinear radial svm. As expected from the previous findings, this nonlinear classifier did not work very well for the data.

The best CV error (40.36%) was achieved for sigma (gamma)=0.01 and cost=10, and the training error was 0.07%.

This model was also overfitted as random forest, but produces a way higher CV-error than random forest due to its inherent high nonlinearity; the random forest is nonparametric but it is quite well fitted to linearity as well due to its averaging method.

```
svmGrid=expand.grid(sigma=c(0.01, 0.1, 0.5, 1, 10), C=c(0.01, 0.1, 0.5, 1, 10))
svm.Radial=train(as.formula(form), data=Total, method="svmRadial", trControl=fitControl, tuneGrid=svmGr
plot(svm.Radial)
```




```
##CV error
svm.Radial$results[which.max(svm.Radial$results$Accuracy),]
```

```
##   sigma  C  Accuracy      Kappa AccuracySD   KappaSD
## 5  0.01 10 0.5919773 0.05058717 0.01879439 0.0471025
```

```
svm.Radial$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 10
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.01
##
## Number of Support Vectors : 805
##
## Objective Function Value : -498.2269
## Training error : 0.00073
```

```
##training error
svm.Radial.pred=predict(svm.Radial, newdata=Total)
table(svm.Radial.pred,Total$Mortality)
```

```
##
## svm.Radial.pred   0   1
##               0 788   0
##               1   1 581
```

```
mean(svm.Radial.pred!=Total$Mortality)
```

```
## [1] 0.000729927
```

Conclusion

The results from the series of machine learning models presented that a Ridge regression model produced the best CV misclassification rate for the data.

Since CV error is a reliabe estimator of test error, we can conclude that the Ridge regression model with the selected 255 predictors will be the best model (of all that we have tried) that can predict a new test data with the best accuracy.

The better performance of linear models, especially with shrinkage being applied (Ridge is the best and linear SVC is the next), and the finding that the complicated non-parametric methods (random forest and boosting) produced only very slightly better results than the non-shrunken linear model (logistic) and explicitly worse results than the shrunken linear models, indicate that the true classification boundary for mortality is quite close to linearity.

In the future, we may apply a more recent method for variable selection because the current 255 variables are still a lot. For now, we can depend on the parameter estimates in Ridge regression or the variance importance metric from random forest or gradient boosting. Actually, we may have selected a smaller subset (e.g., 50 or 100) of predictors from the variance importance metric of the tree-based methods, and may have applied Ridge regression or other shrinkage methods on them. However, since this method is quite arbitrary, we did not pursue it further.