Data 622 HW 1

Monu Chacko

10/10/2020

Question 1

Load data

```
data <- read.csv("data622hw1.csv", header = TRUE)
```

Examine the data

```
data[] <- lapply(data, as.factor)
head(data)
```

```
##   X Y label
## 1 5 a  BLUE
## 2 5 b BLACK
## 3 5 c  BLUE
## 4 5 d BLACK
## 5 5 e BLACK
## 6 5 f BLACK
```

```
summary(data)
```

```
##   X     Y       label
##  5 :6   a:6   BLACK:22
##  19:6   b:6   BLUE :14
##  35:6   c:6
##  51:6   d:6
##  55:6   e:6
##  63:6   f:6
```

```
dim(data)
```

```
## [1] 36  3
```

```
str(data)
```

```
## 'data.frame':    36 obs. of  3 variables:
##  $ X    : Factor w/ 6 levels "5","19","35",..: 1 1 1 1 1 1 2 2 2 2 ...
##  $ Y    : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6 1 2 3 4 ...
##  $ label: Factor w/ 2 levels "BLACK","BLUE": 2 1 2 1 1 1 2 2 2 2 ...
```

Train

```
#  applying Cross Validation
ctrl <- trainControl(method="boot", n=100, classProbs=T,  savePredictions =
```

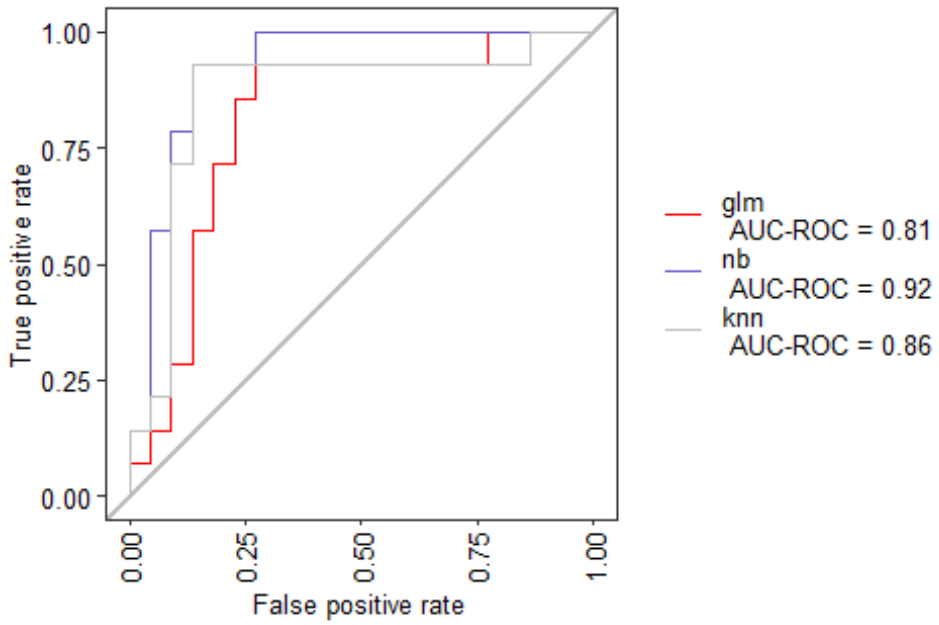
```
T)

fit_glm <- train(label ~ .,data=data, method="glm",family="binomial",
trControl=ctrl)
fit_nb <- train(label ~ .,data=data, method="naive_bayes", trControl=ctrl)
fit_knn <- train(label ~ .,data=data, method="knn", trControl=ctrl)
```

Load data

```
library(caret)
set.seed(300)

res <- evalm(list(fit_glm,fit_nb,fit_knn), gnames=c('glm','nb', 'knn'),
rlinethick=0.5, fsize=10, plots='r')
```




```
m_glm <- cbind(AUC=res$stdres$glm['AUC-ROC','Score'], Accuracy =
mean(fit_glm$results[,'Accuracy']), FPR=res$stdres$glm['FPR','Score'], TPR =
res$stdres$glm['TP','Score']/(res$stdres$glm['TP','Score']+res$stdres$glm['FN
','Score']),
TNR=res$stdres$glm['TN','Score']/(res$stdres$glm['TN','Score']+res$stdres$glm
['FP','Score']),
FNR=res$stdres$glm['FN','Score']/(res$stdres$glm['TP','Score']+res$stdres$glm
['FN','Score']))

m_nb <- cbind(AUC=res$stdres$nb['AUC-ROC','Score'],Accuracy =
mean(fit_nb$results[,'Accuracy']), FPR=res$stdres$nb['FPR','Score'],TPR =
res$stdres$nb['TP','Score']/(res$stdres$nb['TP','Score']+res$stdres$nb['FN','
```

```
Score']),
TNR=res$stdres$nb['TN','Score']/(res$stdres$nb['TN','Score']+res$stdres$nb['F
P','Score']),
FNR=res$stdres$nb['FN','Score']/(res$stdres$nb['TP','Score']+res$stdres$nb['F
N','Score']))

m_knn <- cbind(AUC=res$stdres$knn['AUC-ROC','Score'], Accuracy =
mean(fit_knn$results[,'Accuracy']), FPR=res$stdres$knn['FPR','Score'],TPR =
res$stdres$knn['TP','Score']/(res$stdres$knn['TP','Score']+res$stdres$knn['FN
','Score']),
TNR=res$stdres$knn['TN','Score']/(res$stdres$knn['TN','Score']+res$stdres$knn
['FP','Score']),
FNR=res$stdres$knn['FN','Score']/(res$stdres$knn['TP','Score']+res$stdres$knn
['FN','Score']))
```

GLM

```
m_glm

##       AUC  Accuracy   FPR       TPR       TNR       FNR
## [1,] 0.81 0.7144881 0.227 0.7857143 0.7727273 0.2142857

confusionMatrix(fit_glm)

## Bootstrapped (100 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction BLACK BLUE
##      BLACK  46.0 12.5
##      BLUE   16.2 25.4
##
##  Accuracy (average) : 0.7131
```

NB

```
m_nb

##       AUC  Accuracy   FPR TPR       TNR FNR
## [1,] 0.92 0.6365383 0.545   1 0.4545455   0

confusionMatrix(fit_nb)

## Bootstrapped (100 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction BLACK BLUE
##      BLACK  30.1  3.2
##      BLUE   32.1 34.6
```

```
##
##  Accuracy (average) : 0.6471
```

KNN

```
m_knn
```

```
##       AUC  Accuracy   FPR TPR       TNR FNR
## [1,] 0.86 0.6997561 0.091 0.5 0.9090909 0.5
```

```
confusionMatrix(fit_knn)
```

```
## Bootstrapped (100 reps) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction BLACK BLUE
##      BLACK  50.5 20.1
##      BLUE    9.7 19.8
##
##  Accuracy (average) : 0.7029
```

```
summary = rbind(m_glm, m_nb, m_knn)
rownames(summary) <- c("LR", 'NB',"KNN")
summary
```

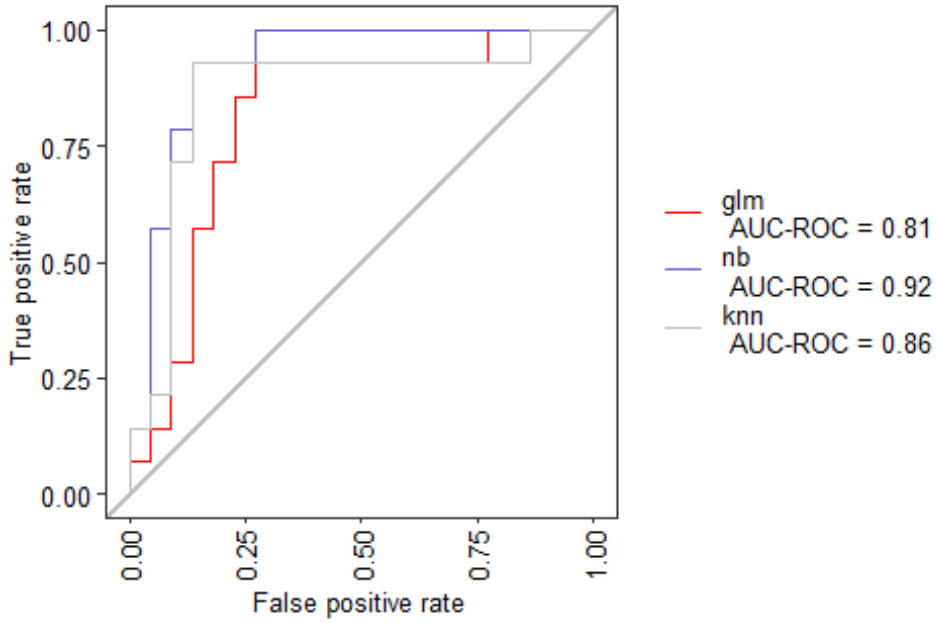
```
##      AUC  Accuracy   FPR       TPR       TNR       FNR
## LR  0.81 0.7144881 0.227 0.7857143 0.7727273 0.2142857
## NB  0.92 0.6365383 0.545 1.0000000 0.4545455 0.0000000
## KNN 0.86 0.6997561 0.091 0.5000000 0.9090909 0.5000000
```

Conclusion

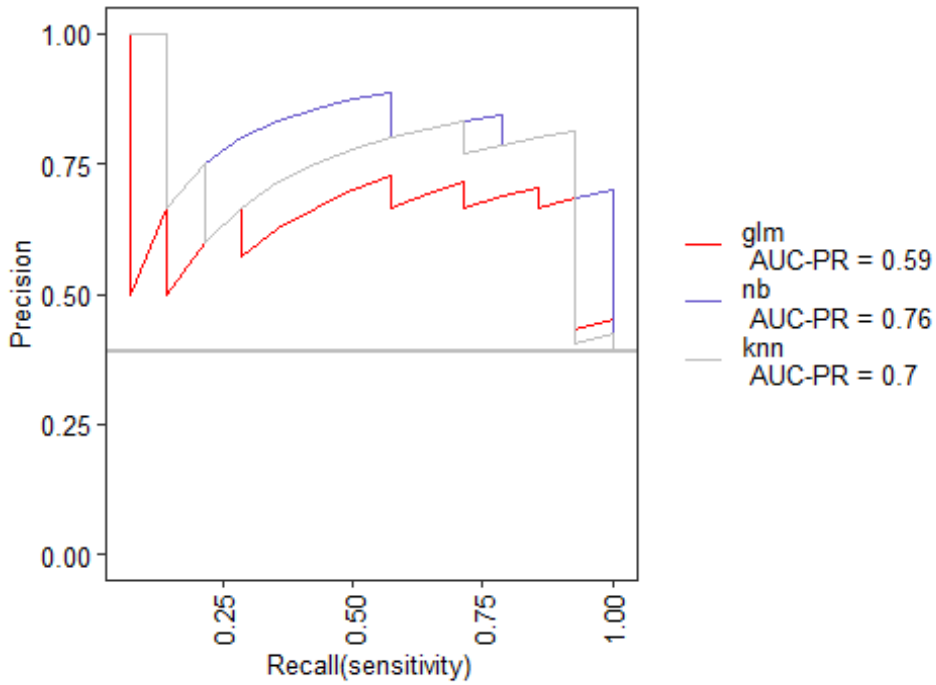
From the above matrix we see that LR has a high True Positive Rate (TPR) and True negative Rate (TNR) and a smaller False Positive Rate (FPR) and False Negative Rate (FNR). NB and KNN on the other hand have interesting results. In the case of NB it has high TPR and a zero FNR but 60% FPR. KNN has 91% TNR and a lower TPR compared to LR. So I would think LR is a better model.

Visuals

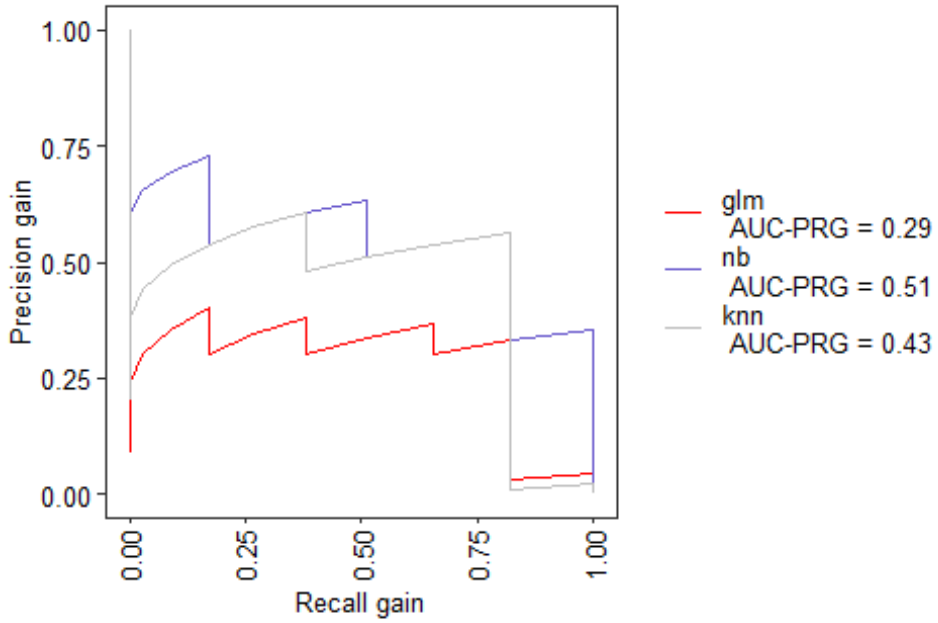
```
res$roc
```



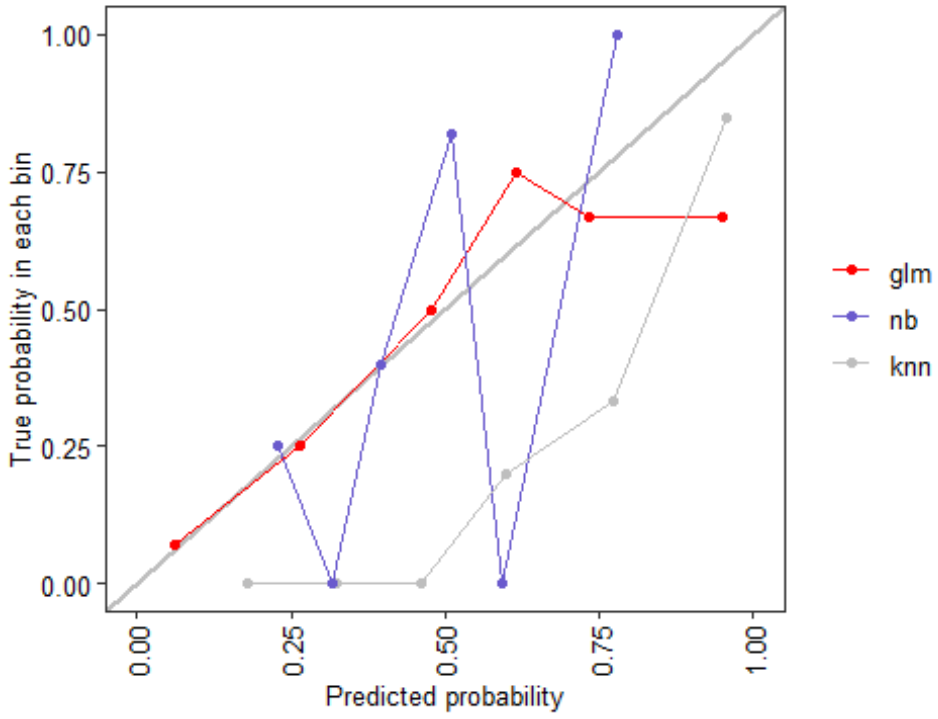

```
res$proc
```




```
res$prg
```


Precision gain
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Recall gain
glm
AUC-PRG = 0.29
nb
AUC-PRG = 0.51
knn
AUC-PRG = 0.43

```
res$cc
```


True probability in each bin
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
Predicted probability
glm
nb
knn

Question 2

What aspects of the data and or aspects of the algorithms, explain these performance differences

Linear Regression is a regression model, meaning, it'll take features and predict a continuous output, eg : stock price,salary etc. Linear regression as the name says, finds a linear curve solution to every problem.

LR is easy and simple to implement. It is fast in training and is good at space complex solution. LR however is applicable only if the solution is linear. In many real life scenarios, it may not be the case. The algorithm assumes the input residuals (error) to be normal distributed, but may not be satisfied always. It assumes input features to be mutually independent(no co-linearity).

Naive bayes is a generative model whereas LR is a discriminative model. Naive bayes works well with small datasets, whereas LR+regularization can achieve similar performance. LR performs better than naive bayes upon colinearity, as naive bayes expects all features to be independent.

KNN is a non-parametric model, where LR is a parametric model. It is comparatively slower than Logistic Regression. KNN supports non-linear solutions where LR supports only linear solutions. LR can derive confidence level (about its prediction), whereas KNN can only output the labels. KNN is slow in real time as it have to keep track of all training data and find the neighbor nodes, whereas LR can easily extract output from the tuned θ coefficients.