How to win?

Purpose

1. 如何用十分鐘的數據預測遊戲結束時的勝敗結果 (決定哪個模型最好)

2. 如何快速的判斷出一場遊戲最終的勝敗結果 (決定哪個變數最重要)

motivation

由於 lol 是一個很熱門的遊戲，每個月都有幾百萬的活躍玩家在玩，並且每年都會舉辦世界賽，也因為我們也有在玩這款遊戲，所以很好奇該怎麼做才能贏，且因為 data 只有 10 分鐘的數據，對於一場可能 40 分鐘以上的遊戲，10 分鐘的數據很難預測其結果，準確率可能只有 50%，所以我們想透過機器學習的方法看電腦預測有多好，另外，我們也好奇有沒有一個簡單的方法，可以幫助我們一眼就可以猜出勝負，而且準確率比 50% 高。

data

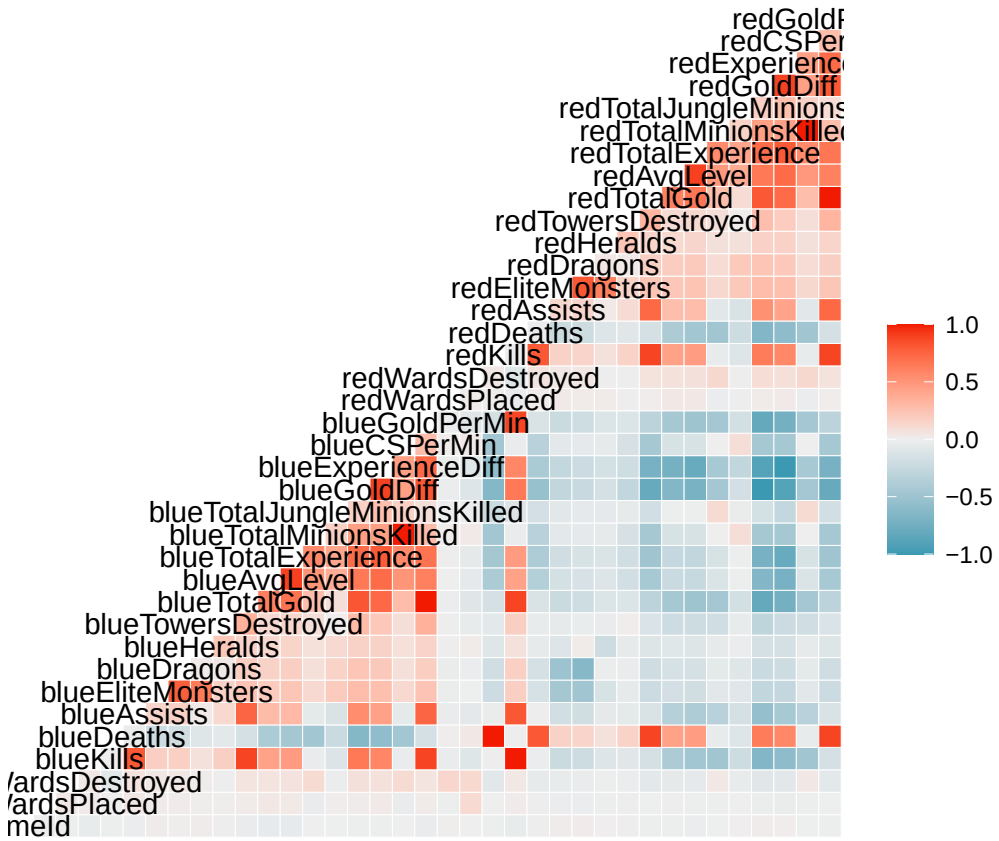
```
library(readr)
lol_data <- read.csv("C:/Users/stat_835/OneDrive/桌面/統計學習/high_diamond_ranked_10min.csv")
lol_data$blueWins <- factor(lol_data$blueWins)
lol_data$blueFirstBlood <- factor(lol_data$blueFirstBlood)
lol_data$redFirstBlood <- factor(lol_data$redFirstBlood)
```

這筆 data 是由 kaggle 上的資料 (https://www.kaggle.com/code/xiyuewang/lol-how-to-win/input)，收集了牌位從 high Diamond 到 low Master(高手場) 十分鐘的數據，其包含了 40 個變數，9879 場的遊戲紀錄；其中變數包含了 blueWins(藍方是否勝利)(response)、blueKills(藍方擊殺數)、blueDeaths(藍方死亡數)、redKills(紅方擊殺數).. 等等變數，我們想要用這些變數來預測 blueWins，其中值得注意的大部分的變數都是數值型。而有三個變數，blueFirstBlood(藍方是否首殺)、redFirstBlood(紅方是否首殺)、blueWins(藍方是否勝利) 這三個變數是類別變數。我們也有考慮要不要將 data 做標準化，但其效果與沒做標準化差不多，另外考慮做標準化後的 classification tree，節點部分會不好解釋，故我們在 train model and test 的 data 都不做標準化 (Lasso 還是有標準化)。

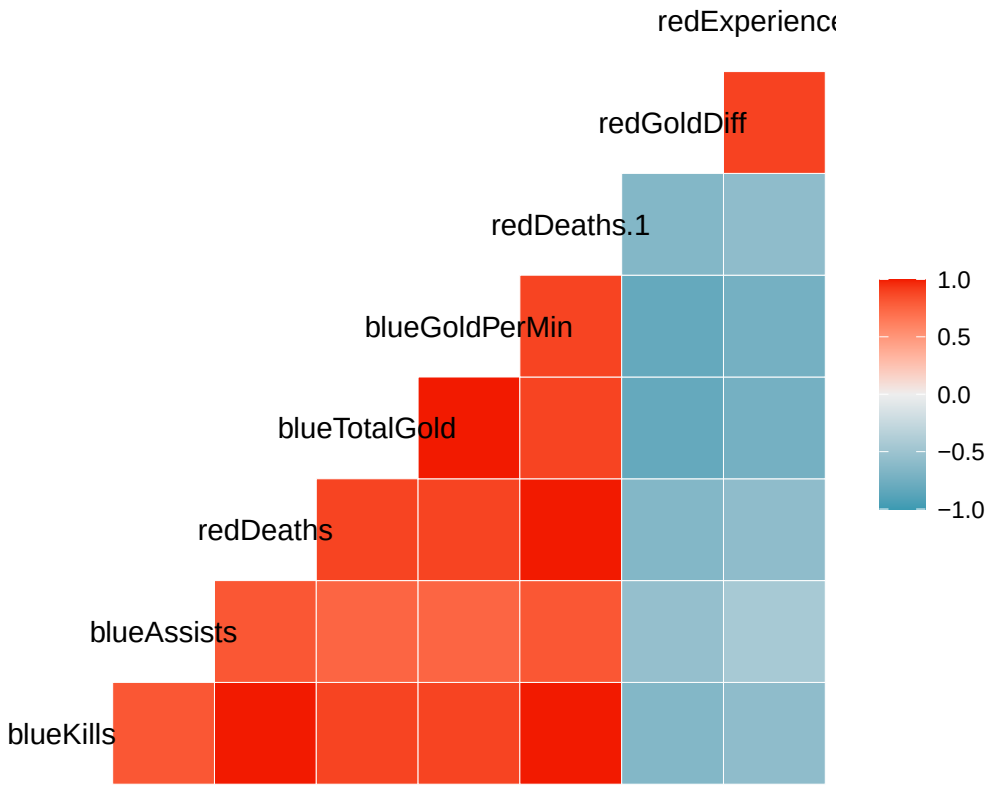
EDA

correlation plot

```
cor_p <- ggcorr(lol_data)
cor_p
```



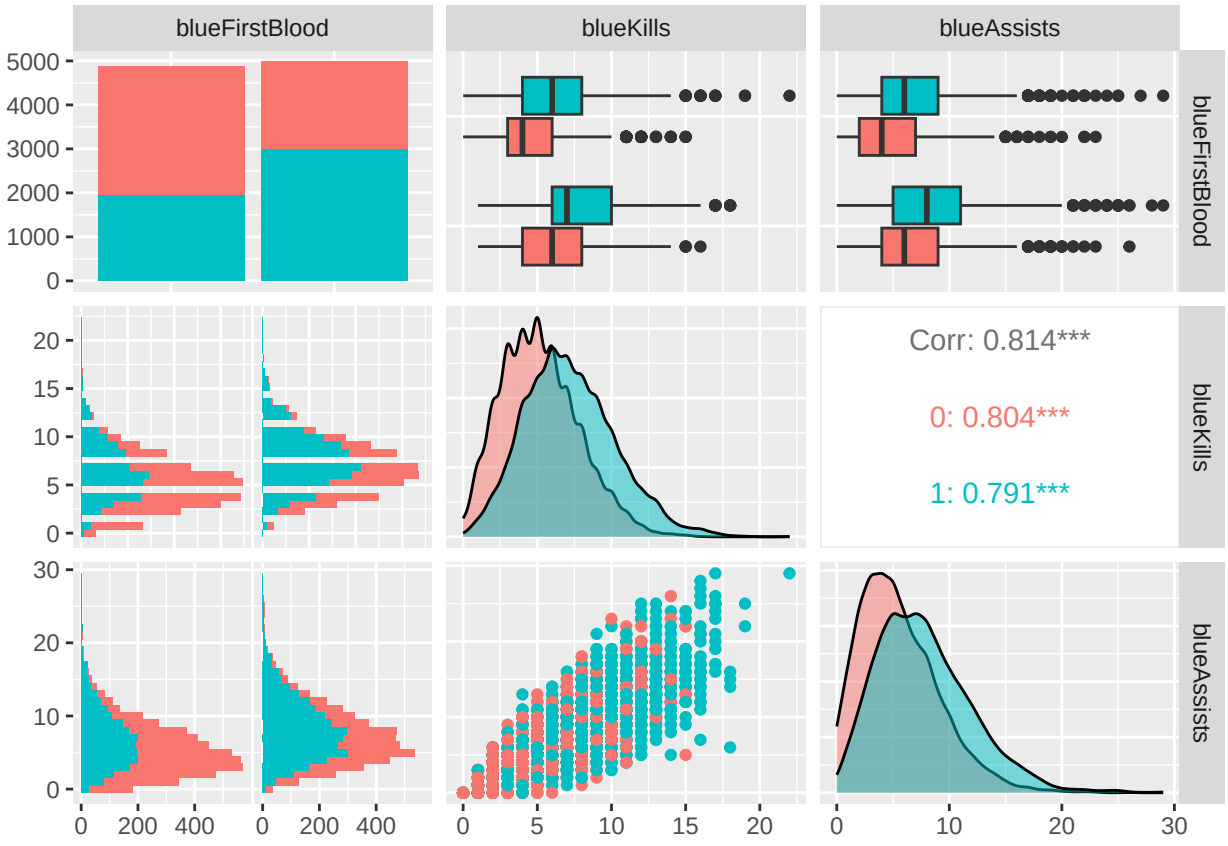

```
cor_p1 <- ggcorr(lol_data[,c(6,8,26,13,21,26,37,38)])
cor_p1
```



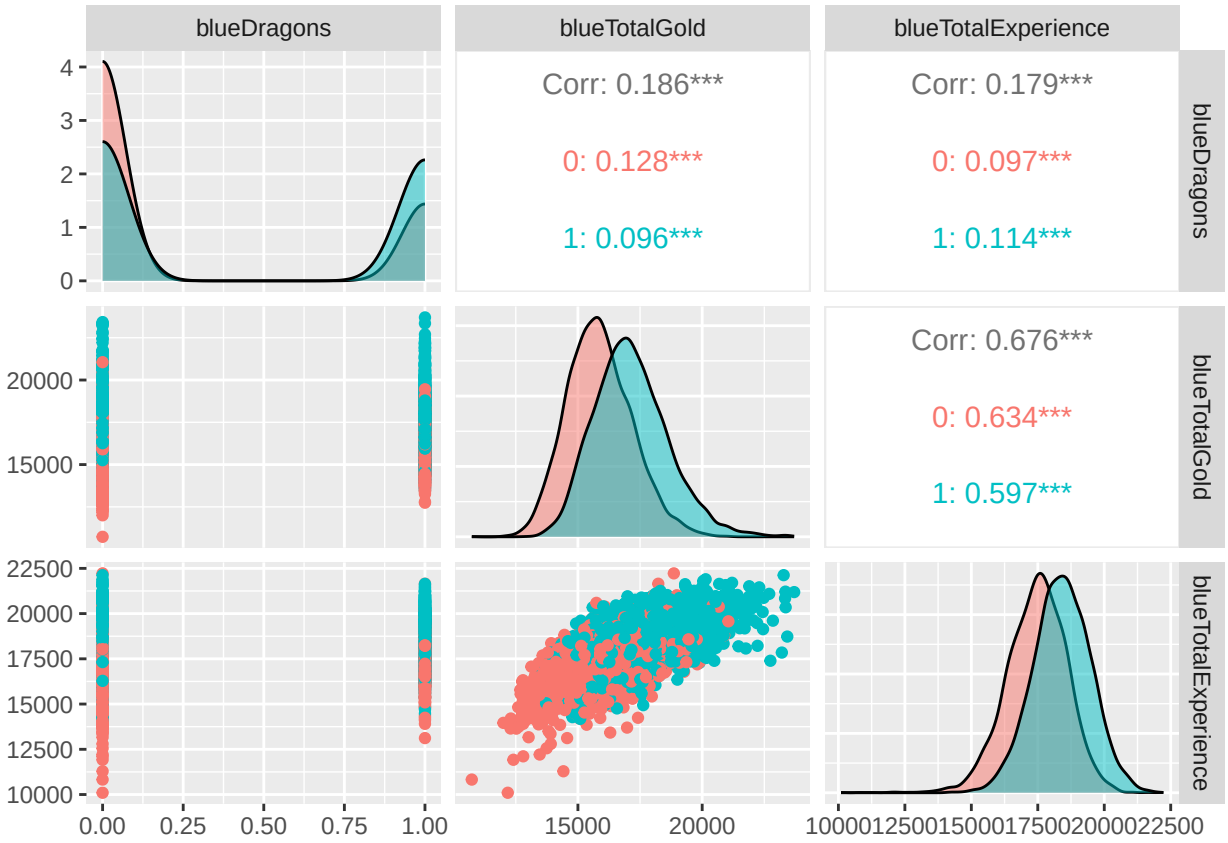
另外變數之間也有一些有完全共線性的問題，例如:blueKills 表示藍方擊殺多少個紅方的敵人，與 redDeaths(紅方死亡數) 數值會一模一樣；blueTotalGold(藍方總經濟) 為 10 分鐘藍方的總經濟，而 blueGoldPerMin(藍方平均一分鐘的經濟)，所以 blueTotalGold and blueGoldPerMin 就是 10 倍的關係；也可以藉由上面的圖發現，變數之間會有幾塊很紅 (相關係數接近 1)、也有幾塊非常的藍 (相關係數接近-1)，我們會先把有完全共線性的變數刪掉其中一個變數，例如:blueKills、redDeaths 變數會刪掉 redDeaths；blueTotalGold、blueGoldPerMin 會刪掉 blueGoldPerMin，把這些完全共線性的 data 刪掉後，我們接著會用 domain knowledge 刪出我們覺得重要的變數。

Important variables

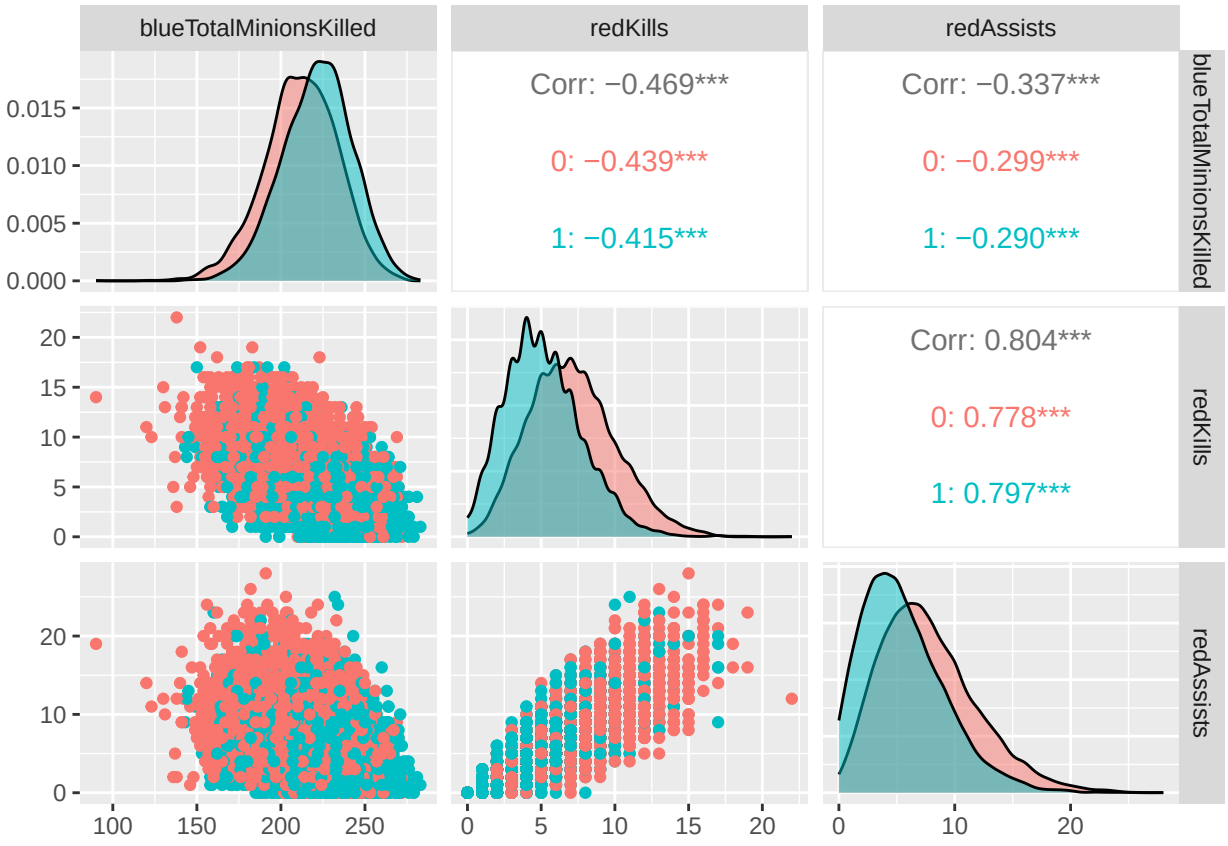
```
ggpairs(lol_data[,c(5,6,8)],ggplot2::aes(colour = lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```



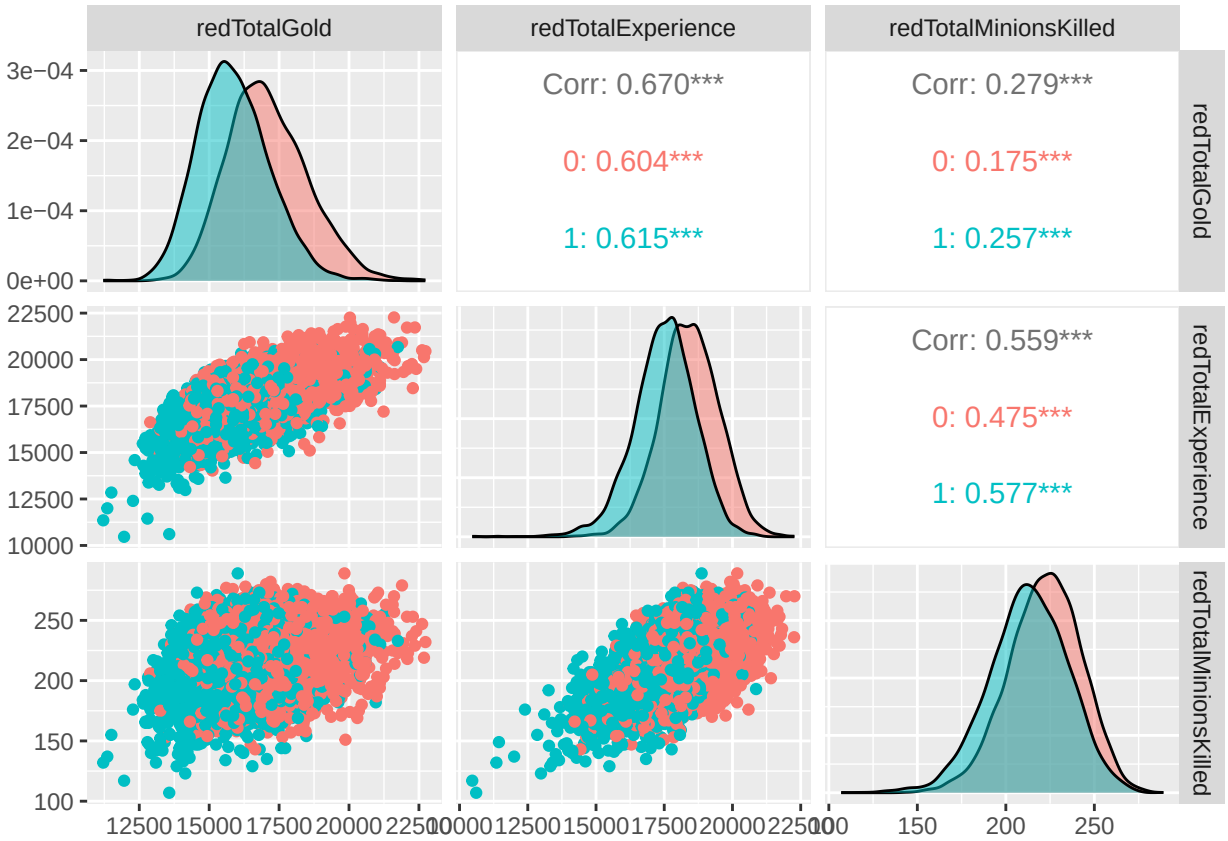

```
ggpairs(lol_data[,c(10,13,15)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```




```
ggpairs(lol_data[,c(16,25,27)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```



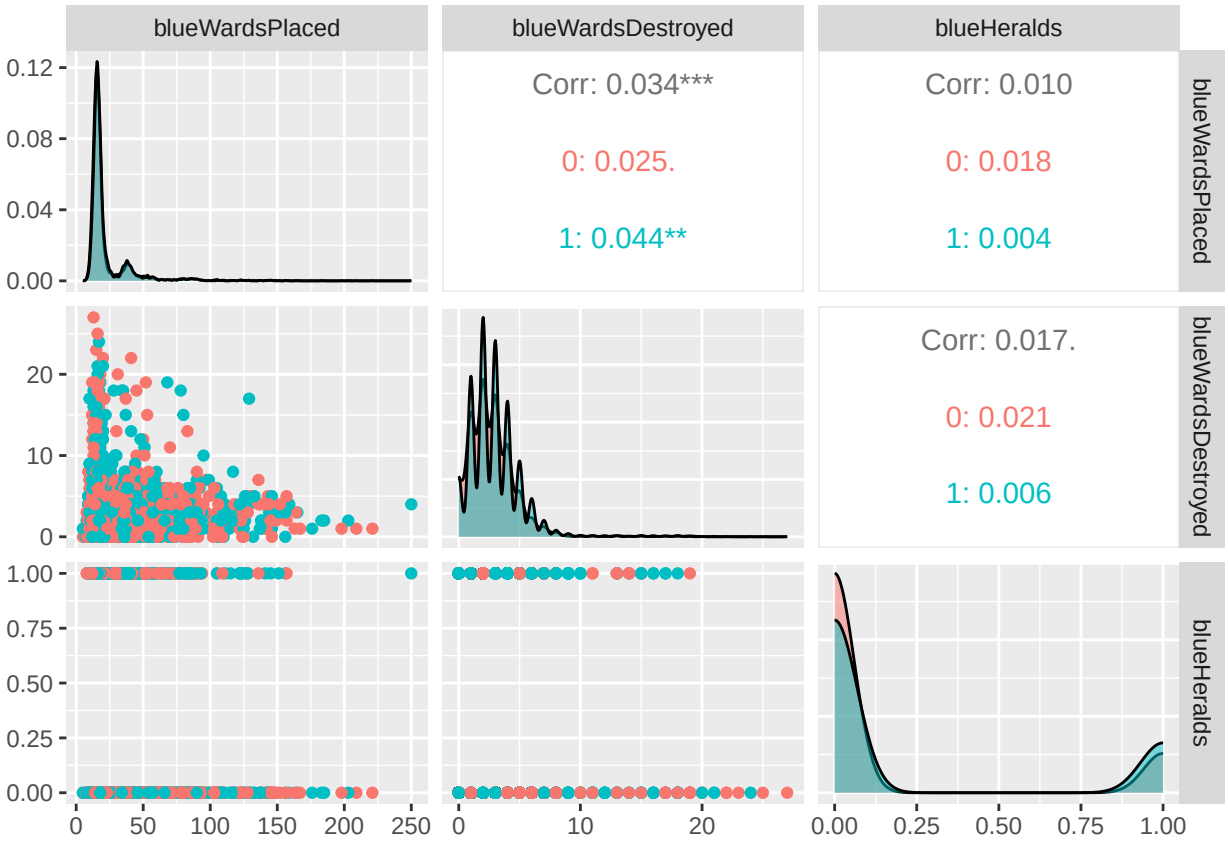

```
ggpairs(lol_data[,c(32,34,35)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```



而因為我們都有玩這個遊戲，我們先用我們的 domain knowledge 找出我們覺得重要的解釋變數，其找出的變數有 blueFirstBlood、blueKills、blueAssists(藍隊助攻數)、blueDragons(藍隊吃的小龍數)、blueTotalGold(藍隊總經濟)、blueTotalExperience(藍隊總經驗)、blueTotalMinionsKilled(藍隊總吃兵數)、redKills(紅隊擊殺數)、redAssists(紅隊助攻數)、redTotalGold(紅隊總經濟)、redTotalExperience(紅隊總經驗)、redTotalMinionsKilled(紅隊總吃兵數)，由上圖可以看到這 12 個變數可以讓很明顯的影響勝負。

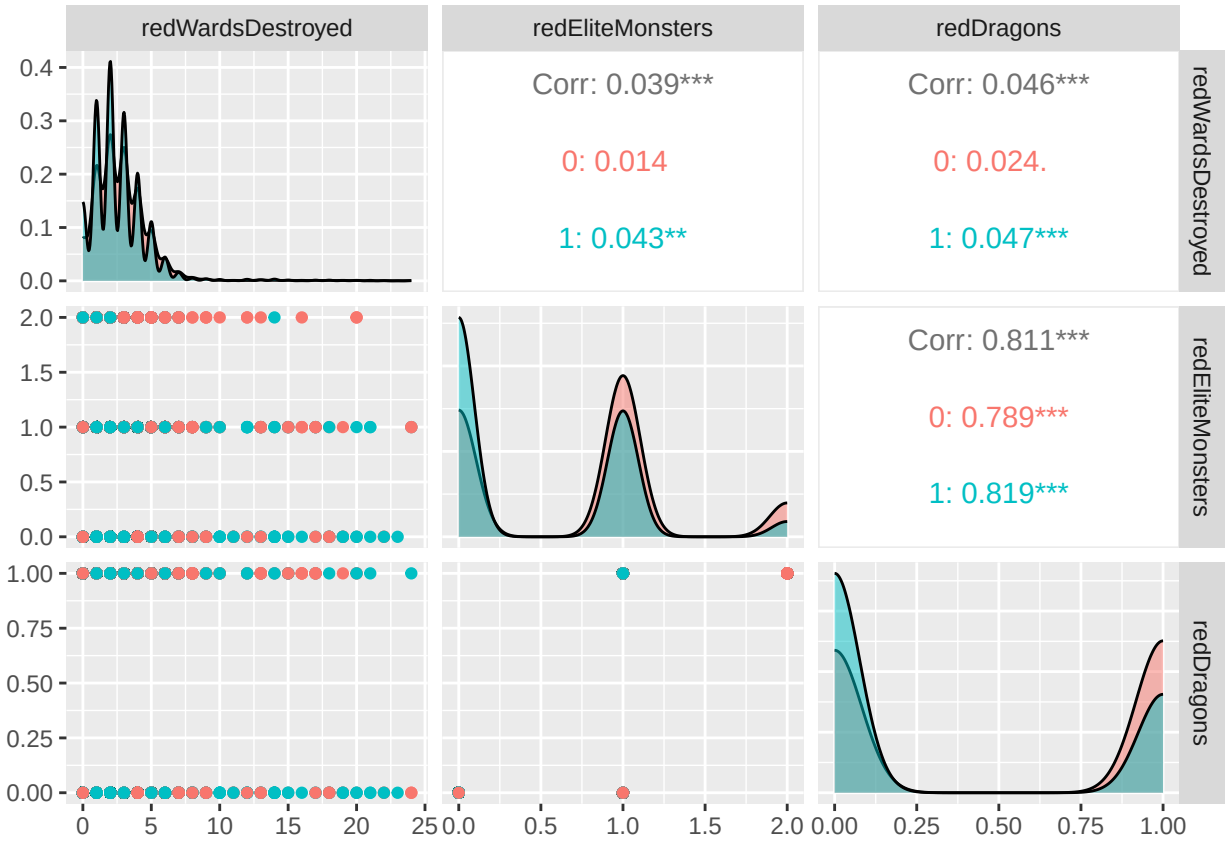
Unimportant variables

```
ggpairs(lol_data[,c(3,4,11)],ggplot2::aes(colour = lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```

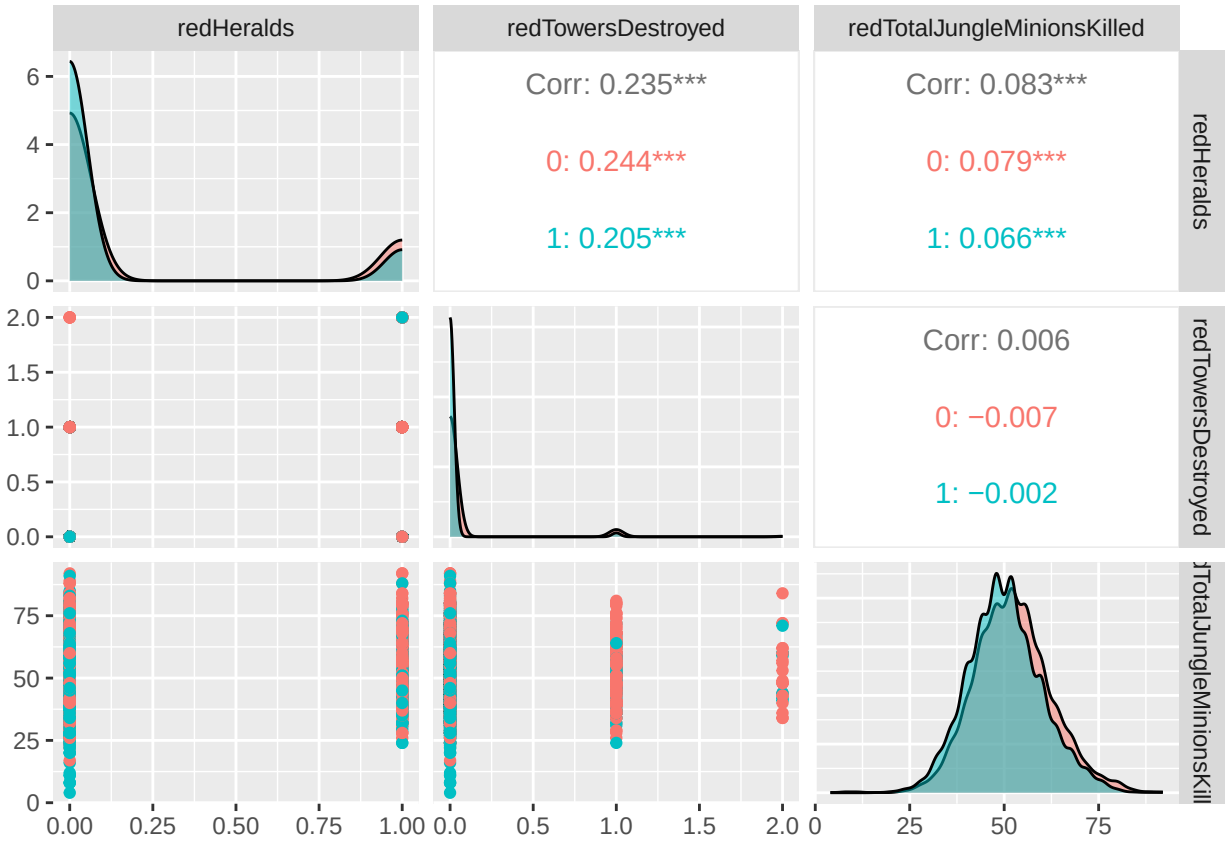



```
ggpairs(lol_data[,c(12,17,22)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```


```
ggpairs(lol_data[,c(23,28,29)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```




```
ggpairs(lol_data[,c(30,31,36)],ggplot2::aes(colour=lol_data$blueWins),
        diag = list(continuous = wrap("densityDiag", alpha = 0.5, colour = "black")))
```



而這些變數就是相對不重要的變數，可以看到當紅方或藍方獲勝時，對變數來說其紅色的分配和藍色的分配並無顯著的不同，也與我們玩遊戲的經驗一致，所以我們就將這些變數不考慮放在一些 model 中。

```
lol_data_sel <- lol_data[,c(2,5,6,8,10,13,15,16,25,27,32,34,35)]
```

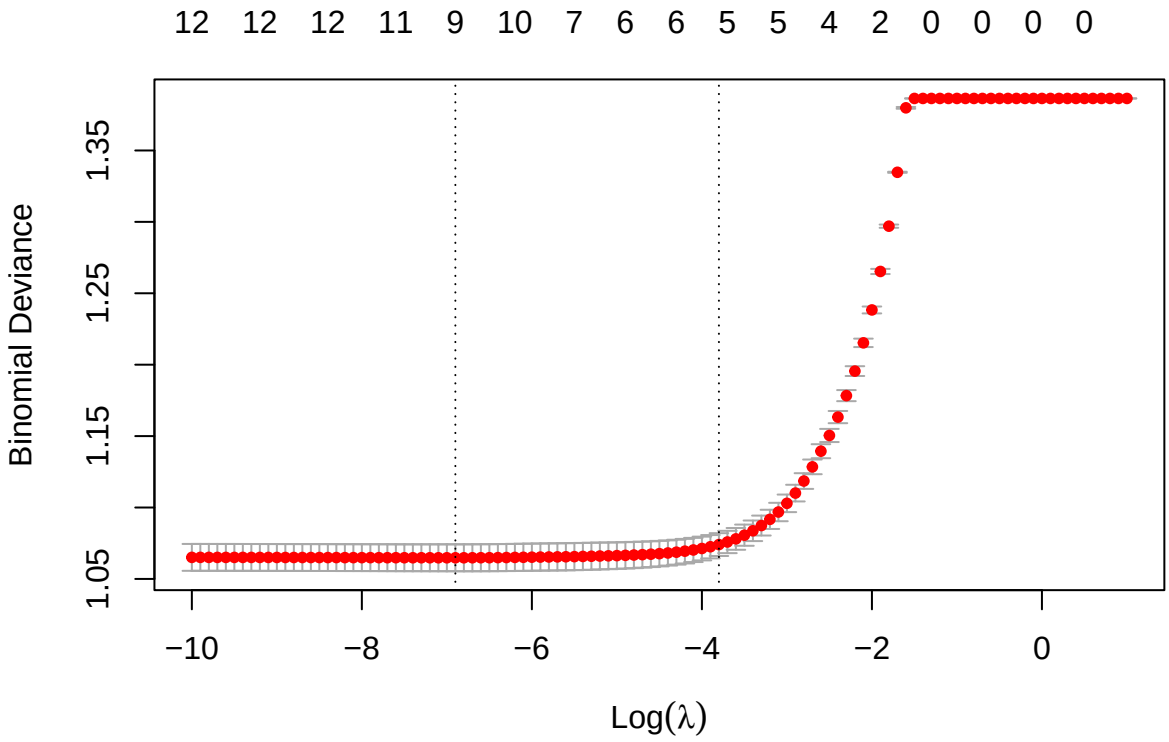
Cross validation

```
set.seed(10)
num <- sample(1:nrow(lol_data_sel), nrow(lol_data_sel), replace = F)
sel_train <- lol_data_sel[num[1:7903],]
sel_test <- lol_data_sel[num[7904:9879],]
all_train <- lol_data[num[1:7903],]
all_test <- lol_data[num[7904:9879],]
```

我們用 4/5 的 data 來做 training data，剩下 1/5 的 data 來做 testing data，所以 training data 有 7903 筆資料，testing data 有 1975 筆資料，而我們選擇 10 folds validation。

```
set.seed(101)
x <- scale(sel_train[,c(-1,-2)])
x_mat <- model.matrix(sel_train[,1] ~ x+sel_train$blueFirstBlood)[,-1]
mod_lasso <- cv.glmnet(x_mat, sel_train[,1], alpha = 1, nfolds = 10, family = "binomial",
                       lambda = exp(seq(-10, 1, 0.1)))

best_lasso <- mod_lasso$lambda.min
plot(mod_lasso)
```




```
mod_lasso_best <- glmnet(x_mat, sel_train[,1], alpha = 1, family = "binomial")
predict(mod_lasso_best, s = best_lasso, type = "coefficients")
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                      s1
## (Intercept)                -0.002186338
## xblueKills                  .
## xblueAssists               -0.053897359
## xblueDragons                0.275331303
## xblueTotalGold              0.665907476
## xblueTotalExperience        0.299188663
## xblueTotalMinionsKilled    -0.052336086
## xredKills                   .
## xredAssists                 .
## xredTotalGold              -0.582962011
## xredTotalExperience        -0.321849353
## xredTotalMinionsKilled      0.088450130
## sel_train$blueFirstBlood1   0.008555861
```

```
sel_train <- sel_train[,-c(3,9,10)]
sel_test <- sel_test[,-c(3,9,10)]
```

由於我們的變數很多，在有 model assumption 可能會有不收斂的問題 (例如 logistic regression 用全部變數會收斂不了)，且有經過篩選變數的 model 通常也會跑的比較快，也會是比較符合我們的經驗；另外對於 data 要求比較小的 model，我們會考慮使用全部的變數和篩選過後的變數分別做出 model，看會有什麼不同。

我們將 data 分割成 training data and testing data，並將篩出來的變數標準化後使用 LASSO 做 model selection，發現模型將 blueKills、redKills、redAssists 三個變數刪掉了，所以剩下的 9 個變數就是我們最終

選擇的變數，接下來的模型除非不能放太多變數 (如 logsitic regression) 我們都會做兩種模型，一種是 39 個 covariate 全放叫做 all data，另一種是放先經過我們選出來的變數再做 LASSO 所選出的 9 個變數叫做 sel data。

```
set.seed(10)
train_control <-  trainControl(method="cv", number=10)
# glm_cv <- train(blueWins ~ .,
#                 data = sel_train, method = "glm", trControl=train_control)
# glm_cv
# glm_cv_pred <- predict(glm_cv, sel_test)
```

knn

all data

```
#set.seed(9)
#fit_knn_all <- train(blueWins ~ .,
#                     data = all_train,
#                     method = "knn",
#                     trControl=train_control,
#                     tuneGrid= expand.grid(.k = seq(1,100,by=2))           )
#saveRDS(fit_knn_all, file = "fit_knn_all.rds")
fit_knn_all <- readRDS("fit_knn_all.rds")
pred_knn_all <- predict(fit_knn_all, all_test)
confusionMatrix(pred_knn_all, all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 567 442
##          1 427 540
##
##                Accuracy : 0.5602
##                  95% CI : (0.538, 0.5823)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : 2.009e-07
##
##                   Kappa : 0.1203
##
##  Mcnemar's Test P-Value : 0.6348
##
##             Sensitivity : 0.5704
##             Specificity : 0.5499
##          Pos Pred Value : 0.5619
##          Neg Pred Value : 0.5584
##              Prevalence : 0.5030
##          Detection Rate : 0.2869
##    Detection Prevalence : 0.5106
##       Balanced Accuracy : 0.5602
##
##        'Positive' Class : 0
##
```

sel data

```
#set.seed(9)
#fit_knn <- train(blueWins ~ .,
#                 data = sel_train,
#                 method = "knn",
#                 trControl=train_control,
#                 tuneGrid= expand.grid(.k = seq(1,100,by=2)))
#saveRDS(fit_knn, file = "fit_knn.rds")
fit_knn <- readRDS("fit_knn.rds")

pred_knn <- predict(fit_knn, sel_test)
confusionMatrix(pred_knn, sel_test$blueWins)
```

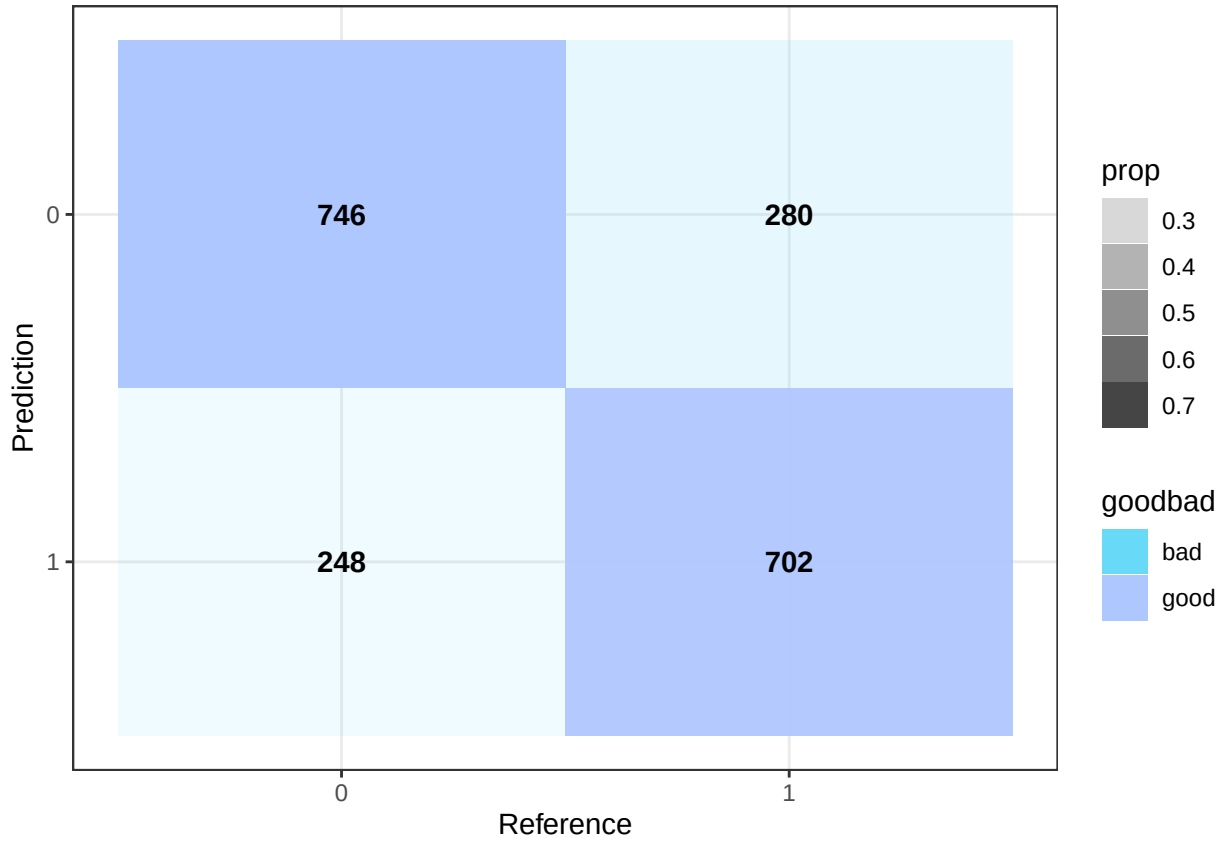
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 746 280
##          1 248 702
##
##                Accuracy : 0.7328
##                  95% CI : (0.7127, 0.7522)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4655
##
##  Mcnemar's Test P-Value : 0.1773
##
##             Sensitivity : 0.7505
##             Specificity : 0.7149
##          Pos Pred Value : 0.7271
##          Neg Pred Value : 0.7389
##              Prevalence : 0.5030
##          Detection Rate : 0.3775
##    Detection Prevalence : 0.5192
##       Balanced Accuracy : 0.7327
##
##        'Positive' Class : 0
##
```

```
table <- data.frame(confusionMatrix(pred_knn, sel_test$blueWins)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
```

```
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



首先，我們先用 knn，最簡單的方法想看一下其效果如何，用的 data 為最終篩出來變數的 data，k 的選擇我們採用奇數 (k 值設定偶數有平票問題) 且限制 k 的範圍在樣本數開根號下 (k 值的選擇有很多，我們參考一些文獻，建議採用樣本數開根號)，並做 cross-validation，所得出的 k 為 93，拿 testing data 去測試後，可由上面的 confusion Matrix 得 Accuracy 為 0.7328，Sensitivity 為 0.7505，Specificity 為 0.7149，其效果我們覺得不錯，相對於從人為猜由 50% 上升到 73.28%；另外，我們也有做將所有變數都放入 knn model，但因為變數比較多，電腦運行速度比較慢，另外所做出來的 Accuracy 也才 0.5602，故不討論。

logistic regression

```
fit_glmnet <- train(blueWins ~ .,
                    data = sel_train,
                    method = "multinom",
                    preProc=c("center", "scale"),
                    trControl = train_control,
                    trace=F)

fit_glmnet$finalModel
```

```
## Call:
## nnet::multinom(formula = .outcome ~ ., data = dat, decay = param$decay,
##     trace = ..1)
##
## Coefficients:
```

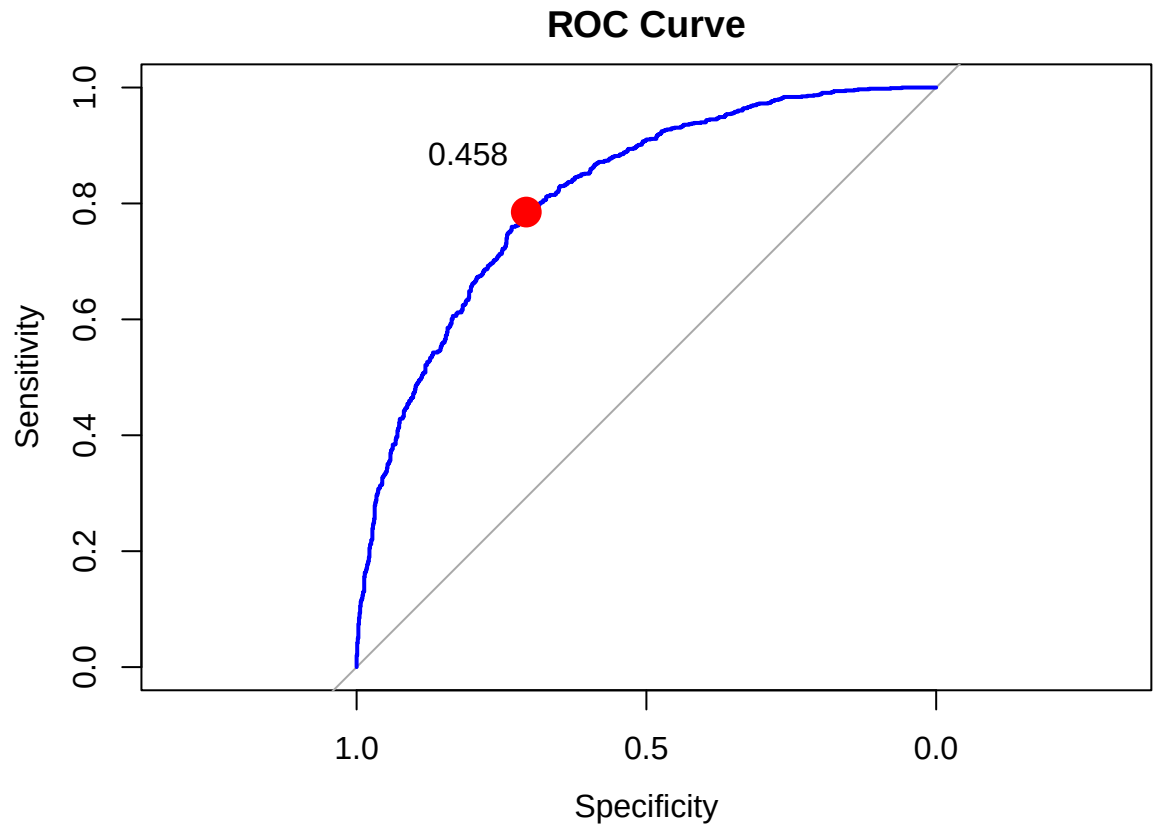
```
##            (Intercept)        blueFirstBlood1            blueAssists
##            0.002802223            0.006668092           -0.094390405
##            blueDragons          blueTotalGold    blueTotalExperience
##            0.281031076            0.716491786            0.296101503
## blueTotalMinionsKilled           redTotalGold     redTotalExperience
##           -0.076371664           -0.593900834           -0.332159839
##  redTotalMinionsKilled
##            0.108523000
##
## Residual Deviance: 8391.444
## AIC: 8411.444
```

```
pred_prob <- predict(fit_glmnet, sel_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(sel_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, round(coords(r
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.4578353
```

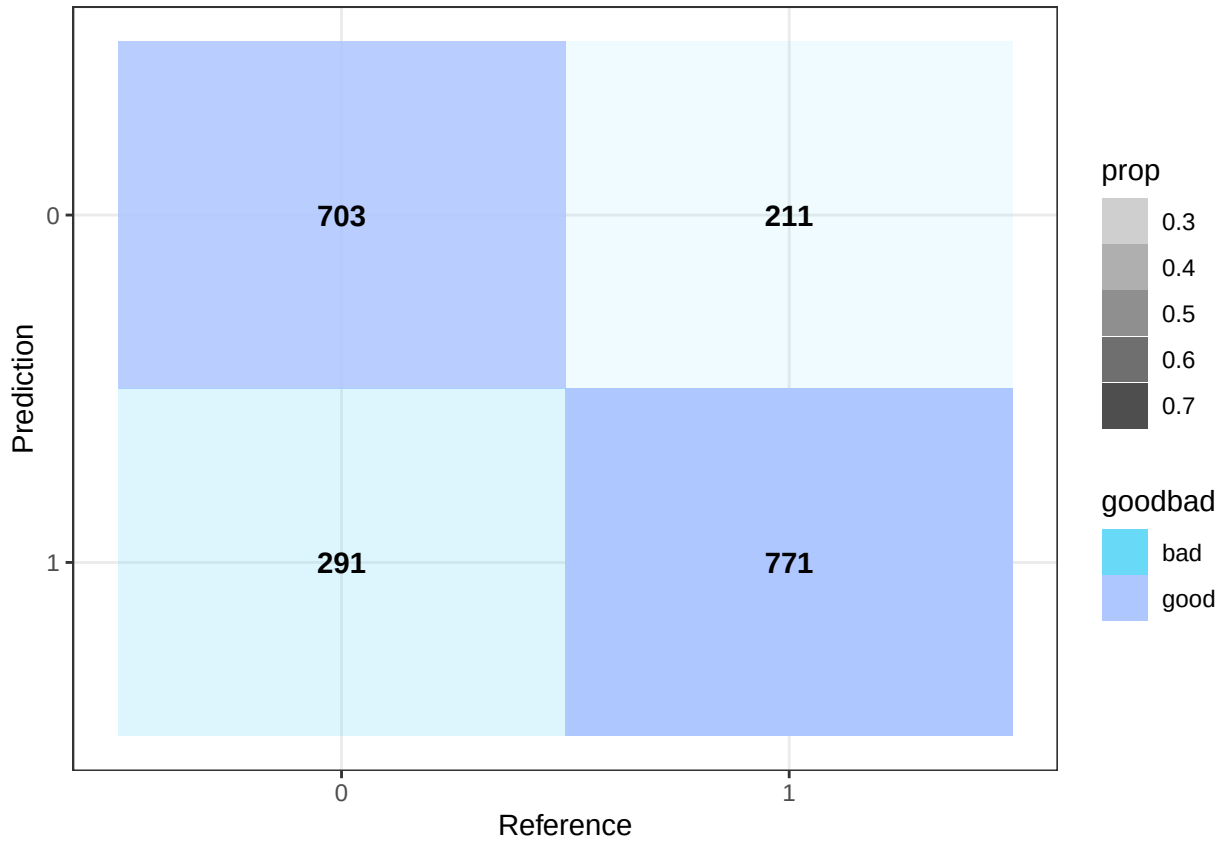
```r
pred_glmnet <- predict(fit_glmnet, sel_test, type = "prob")
pred_logistic <- rep(0, nrow(sel_test))
pred_logistic[pred_glmnet$"1" > best_cut] <- 1
pred_logistic <- factor(pred_logistic)
confusionMatrix(pred_logistic, sel_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 703 211
##          1 291 771
##
##                Accuracy : 0.746
##                  95% CI : (0.7261, 0.765)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4921
##
##  Mcnemar's Test P-Value : 0.000422
##
##             Sensitivity : 0.7072
##             Specificity : 0.7851
##          Pos Pred Value : 0.7691
##          Neg Pred Value : 0.7260
##              Prevalence : 0.5030
##          Detection Rate : 0.3558
##    Detection Prevalence : 0.4626
##       Balanced Accuracy : 0.7462
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_logistic, sel_test$blueWins)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```

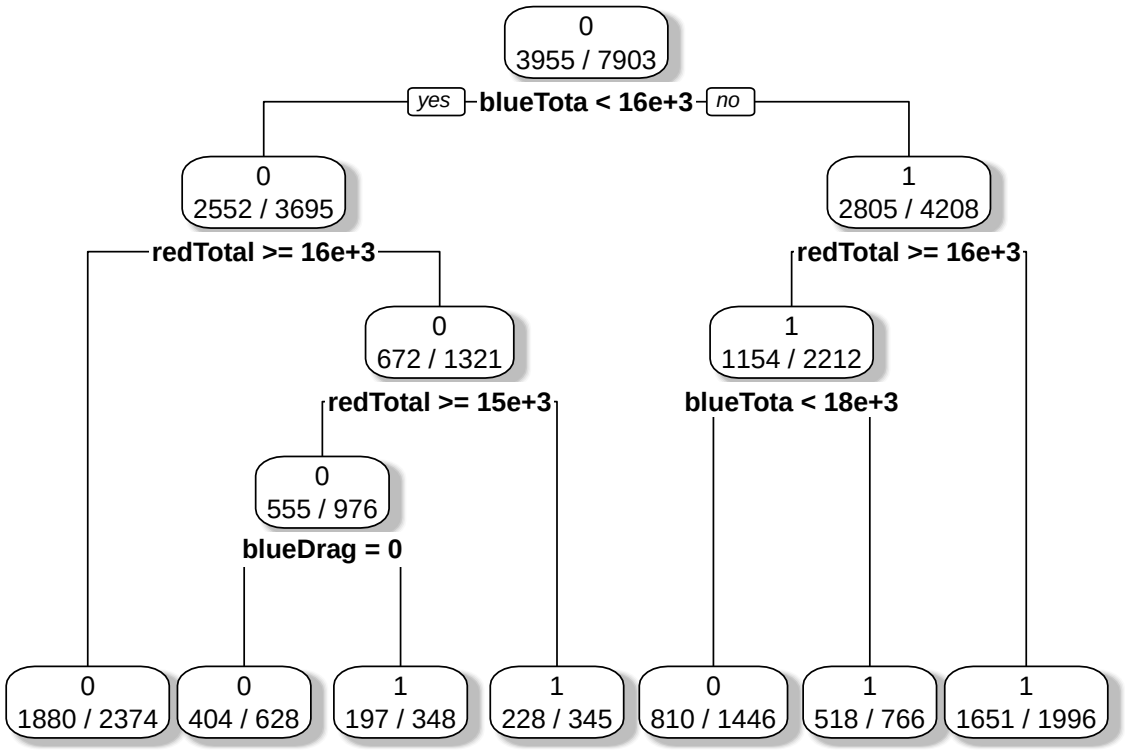


第二個方法我們考慮 logistic regression，放入 model 的變數為 Lasso 篩選出來的最終變數，一樣有用 cross-validation 讓 model 更好，cutpoint 用 ROC 選擇為 0.4578353，並用 testing data 來預測，其 confusion Matrix 為上圖，Accuracy 為 0.746，Sensitivity 為 0.7072，Specificity 為 0.7851，以 Accuracy、Specificity 來說，logistic regression 比 knn 還要好，Sensitivity 則是 knn 較優。而 logistic regression 我們有試過放入全部的變數，但模型收斂不了，故我們不考慮。

classification tree

```
tree_sel_cv <- train(blueWins ~ .,
                     data = sel_train,
                     method = "rpart",
                     trControl = train_control,
                     tuneGrid = expand.grid(cp = seq(0.01, 0.1, by = 0.01)))

prp(tree_sel_cv$finalModel,
    faclen=0,
    fallen.leaves=TRUE,
    type = 2,
    shadow.col="gray",
    extra = 2)
```




```
tree_sel_cv$finalModel
```

```
## n= 7903
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 7903 3948 0 (0.5004429 0.4995571)
##    2) blueTotalGold< 16276.5 3695 1143 0 (0.6906631 0.3093369)
##      4) redTotalGold>=16234 2374  494 0 (0.7919124 0.2080876) *
##      5) redTotalGold< 16234 1321  649 0 (0.5087055 0.4912945)
##       10) redTotalGold>=15070 976  421 0 (0.5686475 0.4313525)
##         20) blueDragons< 0.5 628  224 0 (0.6433121 0.3566879) *
##         21) blueDragons>=0.5 348  151 1 (0.4339080 0.5660920) *
##       11) redTotalGold< 15070 345  117 1 (0.3391304 0.6608696) *
##    3) blueTotalGold>=16276.5 4208 1403 1 (0.3334125 0.6665875)
##      6) redTotalGold>=15955.5 2212 1058 1 (0.4783002 0.5216998)
##       12) blueTotalGold< 17659.5 1446  636 0 (0.5601660 0.4398340) *
##       13) blueTotalGold>=17659.5 766  248 1 (0.3237598 0.6762402) *
##      7) redTotalGold< 15955.5 1996  345 1 (0.1728457 0.8271543) *
```

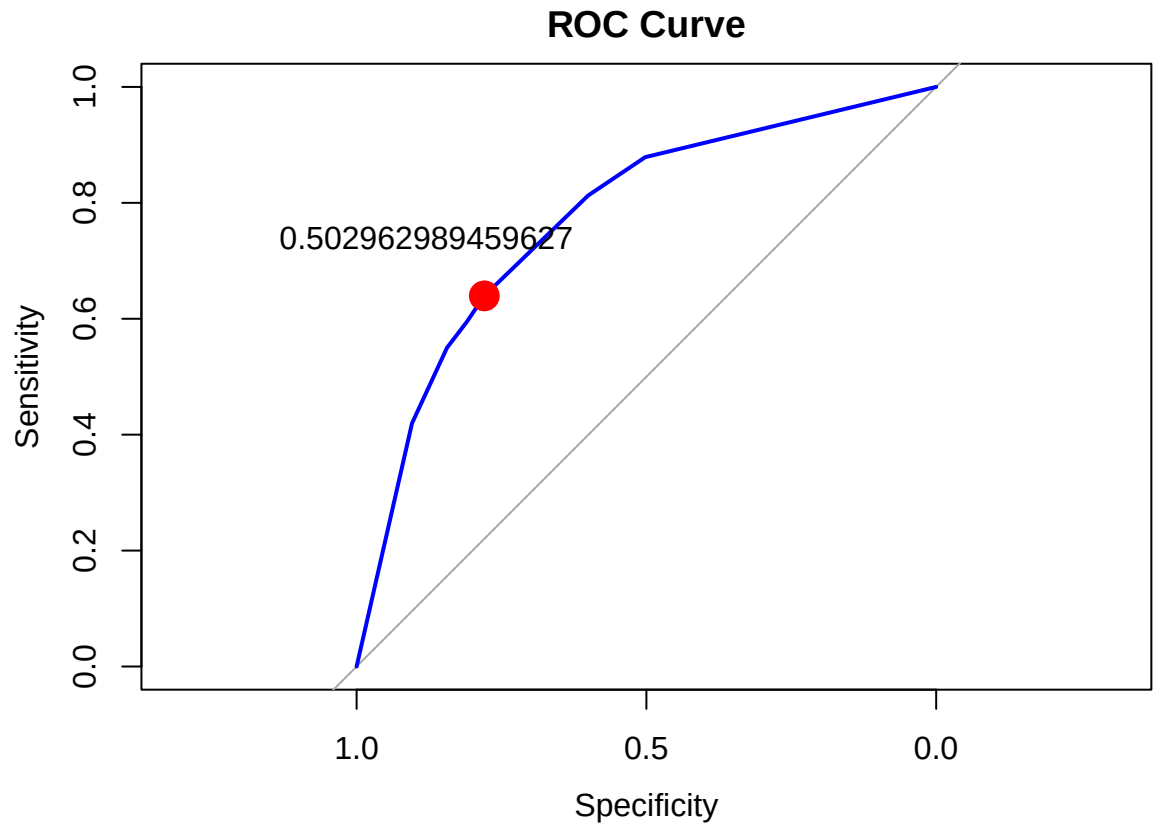
```
# pred_tree_sel_cv <- predict(tree_sel_cv, sel_test)

pred_prob <- predict(tree_sel_cv, sel_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(sel_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.502963
```

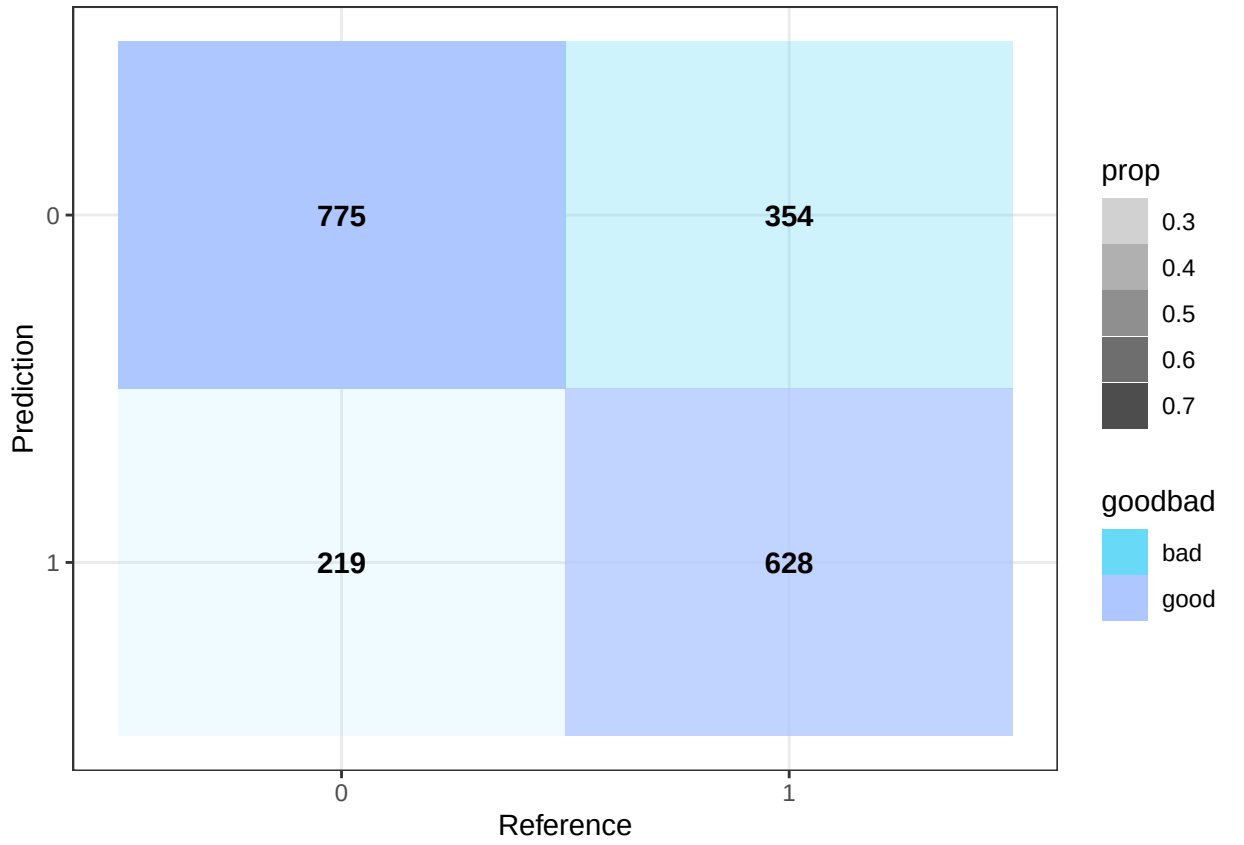
```
pred_tree_sel_cv <- rep(0, nrow(sel_test))
pred_tree_sel_cv[pos_prob > best_cut] <- 1
pred_tree_sel_cv <- factor(pred_tree_sel_cv)
confusionMatrix(pred_tree_sel_cv, sel_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 775 354
##          1 219 628
##
##                Accuracy : 0.71
##                  95% CI : (0.6895, 0.73)
##     No Information Rate : 0.503
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4195
##
##  Mcnemar's Test P-Value : 2.169e-08
##
##             Sensitivity : 0.7797
##             Specificity : 0.6395
##          Pos Pred Value : 0.6864
##          Neg Pred Value : 0.7414
##              Prevalence : 0.5030
##          Detection Rate : 0.3922
##    Detection Prevalence : 0.5714
##       Balanced Accuracy : 0.7096
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_tree_sel_cv, sel_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))


ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



我們先對篩出來的變數，用 cross-validation，並有用剪枝讓 model 效果更好，其做出來的 tree 如上圖，共有 7 個 terminal nodes，其 Accuracy 為 0.71，其效果還是不錯，雖然相較 knn 和 logistic regression 比較差一點，但 Sensitivity 卻來到了最高，達到 0.7797，而 Specificity 為 0.6395，與前面兩個 model 比起來有蠻明顯比較不好。

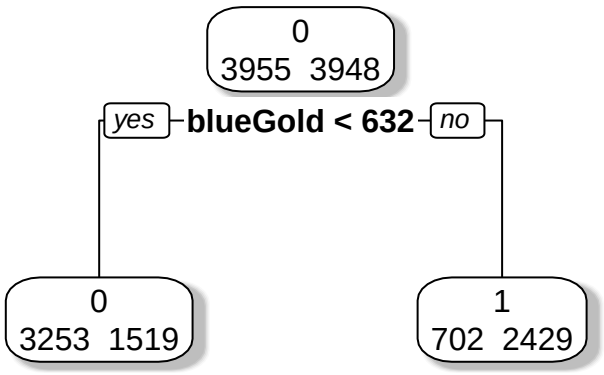
另外，可以注意到大部分的節點都在訴說經濟量這件事，一下說藍方總經濟量，一下說紅方總經濟量，所以感覺與經濟差有關，而恰巧全部的變數中有經濟差這個變數，再加上 tree 比較不會受到共線性的影響，我們考慮放所有的變數到 tree 裡。

```
tree_all_cv <- train(blueWins ~ .,
                data = all_train,
                method = "rpart",
                trControl = train_control,
                tuneGrid = expand.grid(cp = seq(0.01, 0.1, by = 0.01)))
tree_all_cv$finalModel
```

```
## n= 7903
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 7903 3948 0 (0.5004429 0.4995571)
##   2) blueGoldDiff< 632 4772 1519 0 (0.6816848 0.3183152) *
##   3) blueGoldDiff>=632 3131  702 1 (0.2242095 0.7757905) *
```

```
prp(tree_all_cv$finalModel,
    faclen=0,
    fallen.leaves=TRUE,
```

```
    type = 2,
    shadow.col="gray",
    extra = 1)
```




```
pred_tree_all_cv <- predict(tree_all_cv, all_test)
confusionMatrix(pred_tree_all_cv, all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 811 372
##          1 183 610
##
##                Accuracy : 0.7191
##                  95% CI : (0.6987, 0.7389)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4376
##
##  Mcnemar's Test P-Value : 1.461e-15
##
##             Sensitivity : 0.8159
##             Specificity : 0.6212
##          Pos Pred Value : 0.6855
```

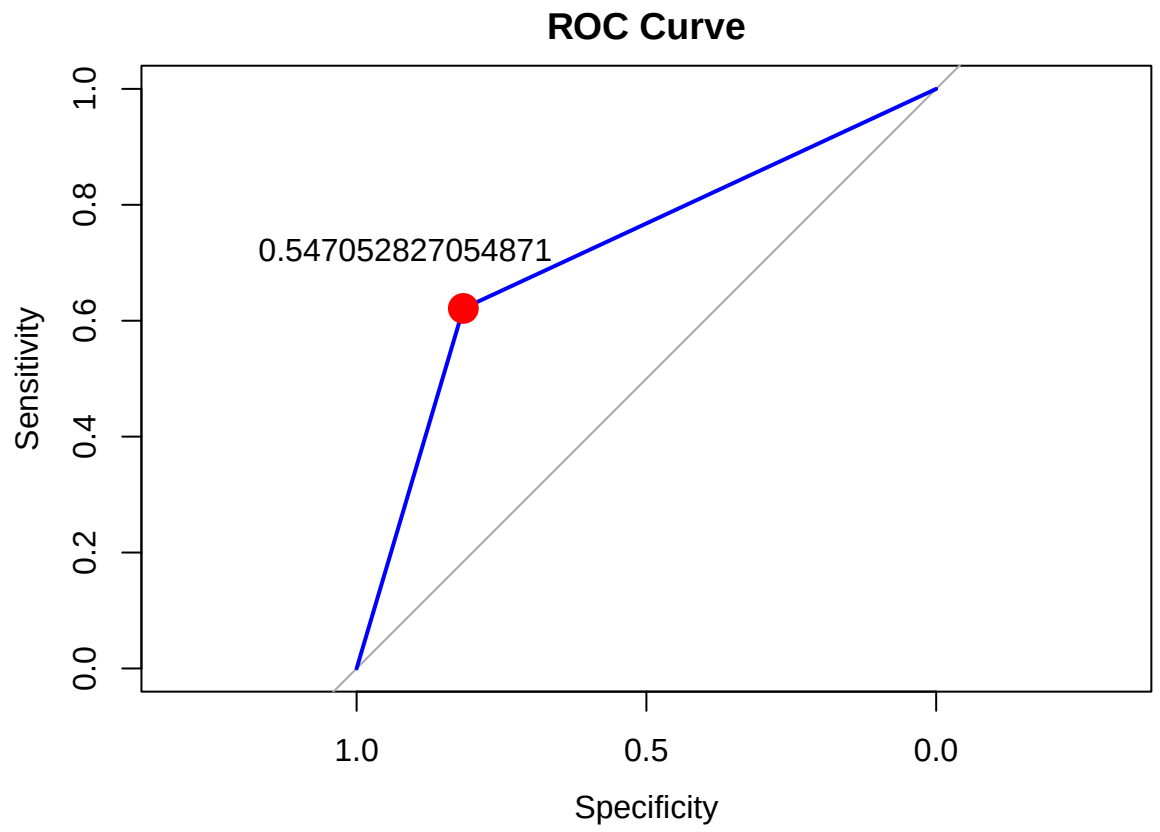
```
##          Neg Pred Value : 0.7692
##              Prevalence : 0.5030
##          Detection Rate : 0.4104
##    Detection Prevalence : 0.5987
##       Balanced Accuracy : 0.7185
##
##        'Positive' Class : 0
##
```

```
pred_prob <- predict(tree_all_cv, all_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(all_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.5470528
```

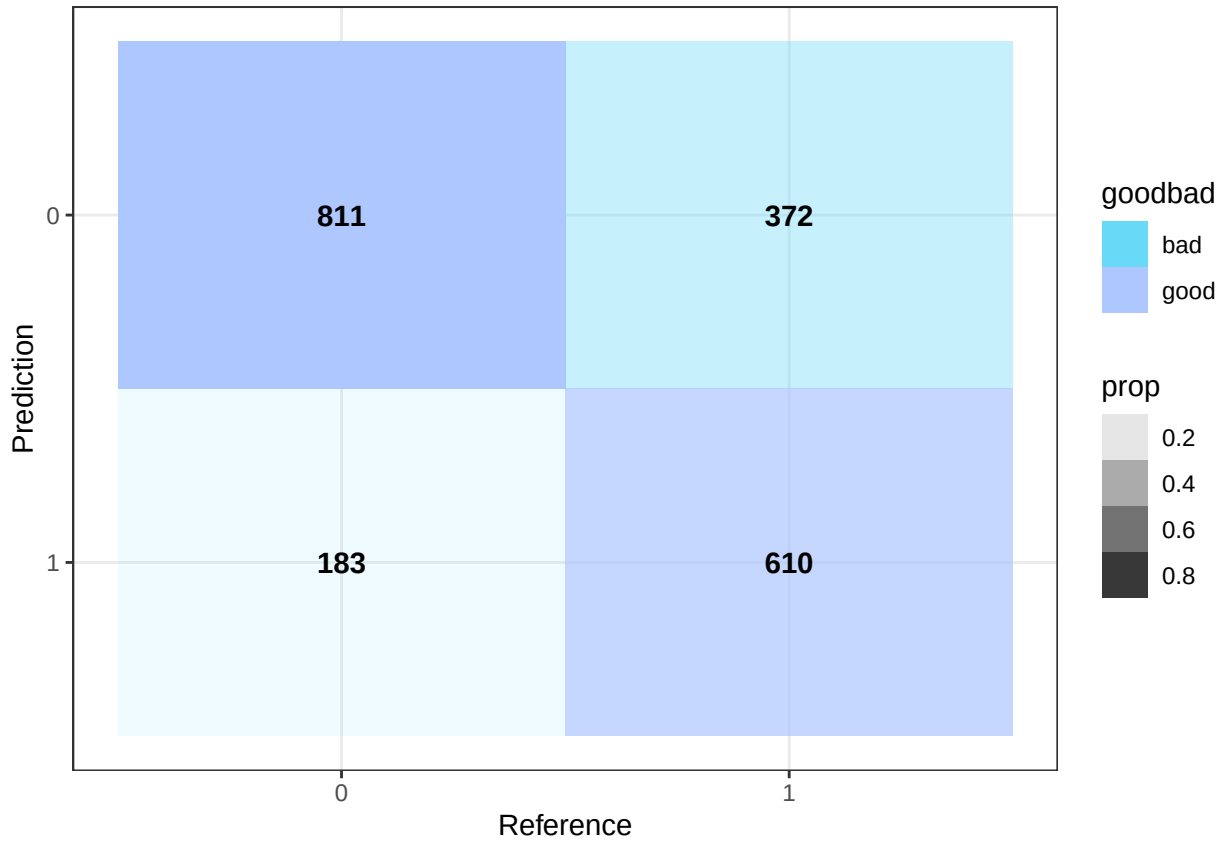
```
pred_tree_all_cv_roc <- rep(0, nrow(all_test))
pred_tree_all_cv_roc[pos_prob > best_cut] <- 1
pred_tree_all_cv_roc <- factor(pred_tree_all_cv_roc)
```

```r
confusionMatrix(pred_tree_all_cv_roc, all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 811 372
##          1 183 610
##
##                Accuracy : 0.7191
##                  95% CI : (0.6987, 0.7389)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4376
##
##  Mcnemar's Test P-Value : 1.461e-15
##
##             Sensitivity : 0.8159
##             Specificity : 0.6212
##          Pos Pred Value : 0.6855
##          Neg Pred Value : 0.7692
##              Prevalence : 0.5030
##          Detection Rate : 0.4104
##    Detection Prevalence : 0.5987
##       Balanced Accuracy : 0.7185
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_tree_all_cv_roc, all_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))


ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



將全部變數放入 model 後 (有做剪枝)，最後只有 2 個 terminal nodes，其 Accuracy 為 0.7191，而 Sensitivity 又更高了，達到了 0.8159，Specificity 則是 0.6212，與放入篩選變數做的 tree 一樣，都有 Sensitivity 較高、Specificity 較低的現象發生。

放入 39 個變數後所做出來的 tree 很明顯比放 Lasso 篩過的 tree 還要簡單很多，節點只有一個，而該節點也正是在講經濟差這件事，這是一個很有趣的結果，原本可能以為放入的變數變多，tree 會變得更複雜，但事實上並沒有，且因為只有探討經濟差是否小於 632 這件事，而其這 632 元並不是一個很大的差距，平均一個人多 126.4 元，但這微小的差距就會對其勝利有所影響；我們以後可以藉由這個準則，很迅速的猜出比賽勝負，另外，很多玩家可能會知道經濟是一個贏得勝利的重點，但很少人知道大概要贏多少的經濟，才能讓自己比較有勝利機會，而這個 tree 也幫我們回答了答案。

random forest

all data

```
library("randomForest")
```

```
## Warning: 套件 'randomForest' 是用 R 版本 4.3.2 來建造的
```

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

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

```
##
## 載入套件：'randomForest'
```

```
## 下列物件被遮斷自 'package:dplyr':
##
##     combine
```

```
## 下列物件被遮斷自 'package:ggplot2':
##
##     margin
```

```
customRF <- list(type = "Classification",
                 library = "randomForest",
                 loop = NULL)

customRF$parameters <- data.frame(parameter = c("mtry", "ntree"),
                                  class = rep("numeric", 2),
                                  label = c("mtry", "ntree"))

customRF$grid <- function(x, y, len = NULL, search = "grid") {}

customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs) {
  randomForest(x, y,
               mtry = param$mtry,
               ntree=param$ntree)
}

#Predict label
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata)

#Predict prob
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")

customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes



# rf_cv <- train(blueWins ~ .,
#                   data = all_train, method = customRF,
#                   tuneGrid=expand.grid(.mtry = c(1,2,3,4,5,8,10, 15, 20, 25, 30), .ntree = c(300, 500,
#                   trControl=train_control)
# saveRDS(rf_cv, file = "rf_cv.rds")
# rf_cv <- train(blueWins ~ .,
#                data = all_train, method = "rf",
#                tuneGrid=expand.grid(.mtry = c(1,2,3,4,5,8,10, 15, 20, 25, 30)),
#                trControl=train_control)
# rf_cv
# rf_cv_pred <- predict(rf_cv, all_test)
# saveRDS(rf_cv, file = "rf_cv.rds")
rf_cv <- readRDS("rf_cv.rds")
rf_cv$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = param$ntree, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 1
##
```

```
##         OOB estimate of  error rate: 27.48%
## Confusion matrix:
##      0    1 class.error
## 0 2984  971   0.2455120
## 1 1201 2747   0.3042047
```

```
rf_cv$bestTune
```

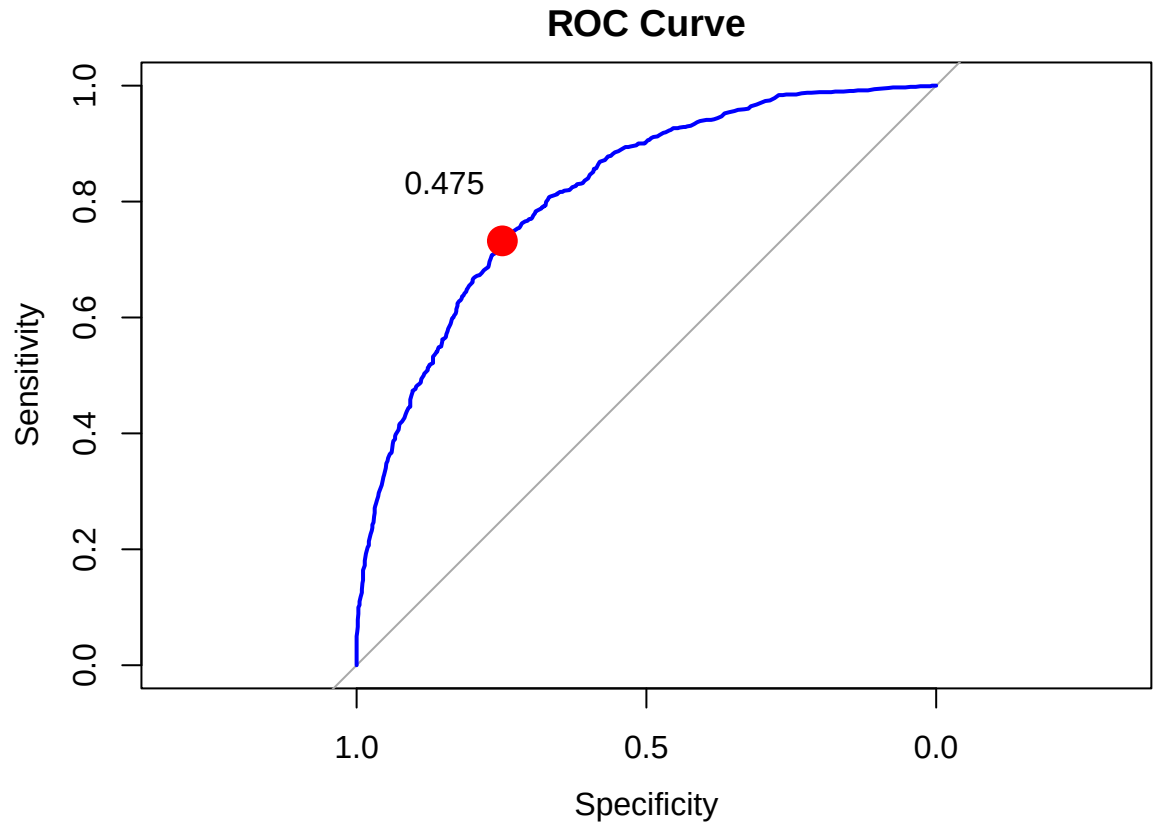
```
##   mtry ntree
## 1    1   300
```

```
pred_prob <- predict(rf_cv, all_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(all_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.475
```

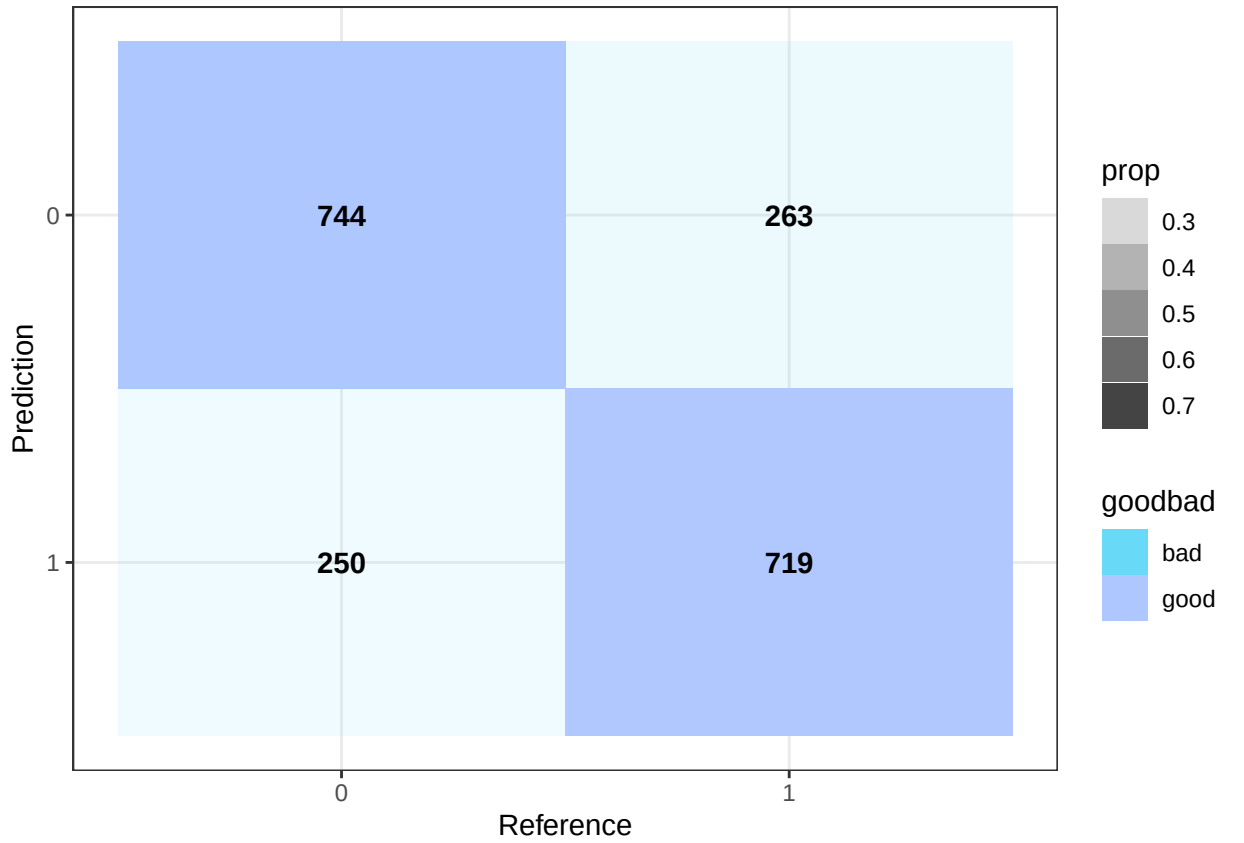
```
pred_rf <- rep(0, nrow(all_test))
pred_rf[pos_prob > best_cut] <- 1
```

```r
pred_rf <- factor(pred_rf)
confusionMatrix(pred_rf, all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 744 263
##          1 250 719
##
##                Accuracy : 0.7404
##                  95% CI : (0.7205, 0.7596)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4807
##
##  Mcnemar's Test P-Value : 0.5962
##
##             Sensitivity : 0.7485
##             Specificity : 0.7322
##          Pos Pred Value : 0.7388
##          Neg Pred Value : 0.7420
##              Prevalence : 0.5030
##          Detection Rate : 0.3765
##    Detection Prevalence : 0.5096
##       Balanced Accuracy : 0.7403
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_rf, all_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within reference groups (see
ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



將全部變數放入 random forest 做 cv 後，因為 train 這個 function 裡面不能同時調整 ntree 和 mtry，所以要自己使用 customRF 才可以同時調整這兩個 tunning parameter，最終的最佳結果為 n.tree=300 也就是做 300 棵不剪枝的 tree，mtry=1 每棵 tree 放進去做的變數都是由 39 個 covariate 隨機選 1 個的效果最好
其 Accuracy 為 0.7404，而 Sensitivity 達到了 0.7485，Specificity 則是 0.7322，此結果與 knn、logistic regression，結果比較相似，以 Accuracy 來說，與 logistic regression 很接近，但還是 logistic regression 略勝一籌；但因為我們是在 kaggle 看到這筆資料，其他人做的 Accuracy 也是 0.71~0.73 之間，所以猜測我們做的 random forest 其 Accuracy 已經達到極限。並不是這個 model 比 logistic regression 差。

sel data

```
# set.seed(10)
# rf_lasso_cv <- train(blueWins ~ .,
#                      data = sel_train, method = customRF,
#                      tuneGrid=tunegrid <- expand.grid(.mtry=c(1:9),.ntree=c(300, 500, 700, 1000)),
#                      trControl=train_control)
# saveRDS(rf_lasso_cv, file = "rf_lasso_cv.rds")
rf_lasso_cv <- readRDS("rf_lasso_cv.rds")
rf_lasso_cv$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = param$ntree, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
```

```
##         OOB estimate of  error rate: 28.14%
## Confusion matrix:
##      0    1 class.error
## 0 2896 1059   0.2677623
## 1 1165 2783   0.2950861
```

```
rf_lasso_cv$bestTune
```

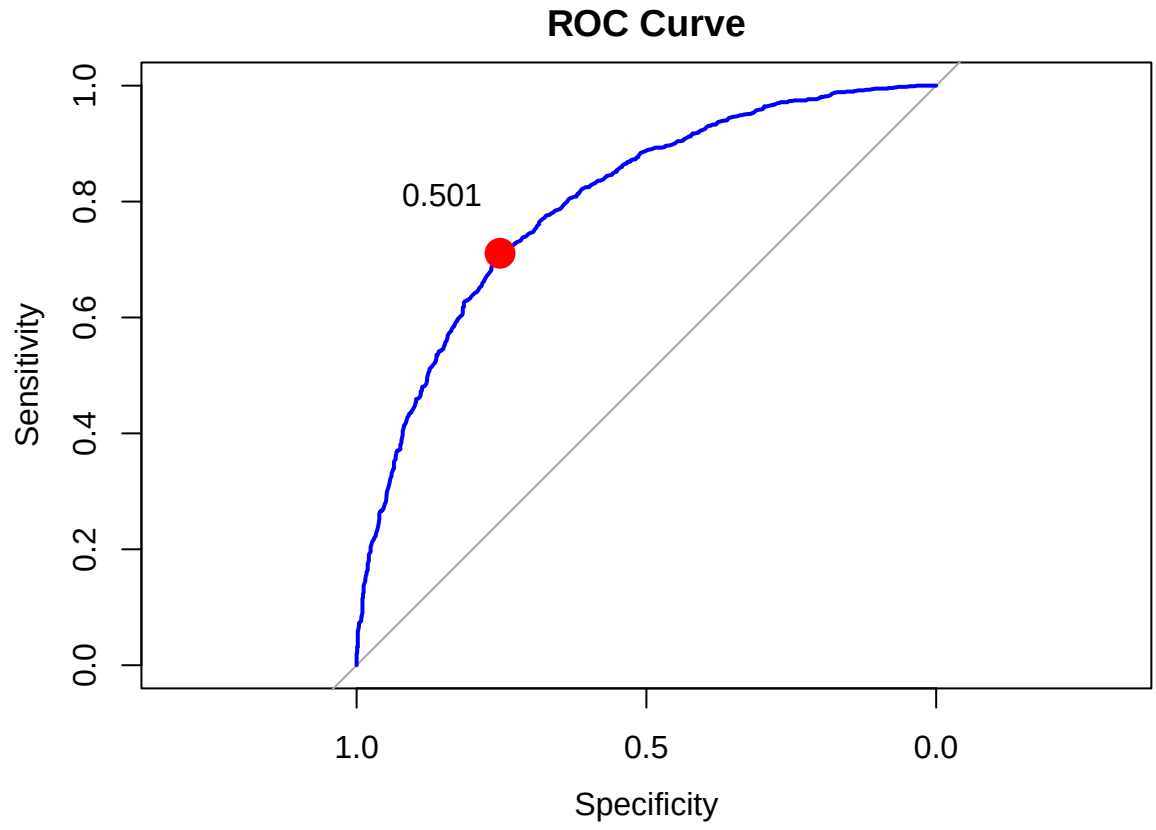
```
##    mtry ntree
## 10    3   500
```

```
pred_prob <- predict(rf_lasso_cv, sel_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(sel_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.501
```

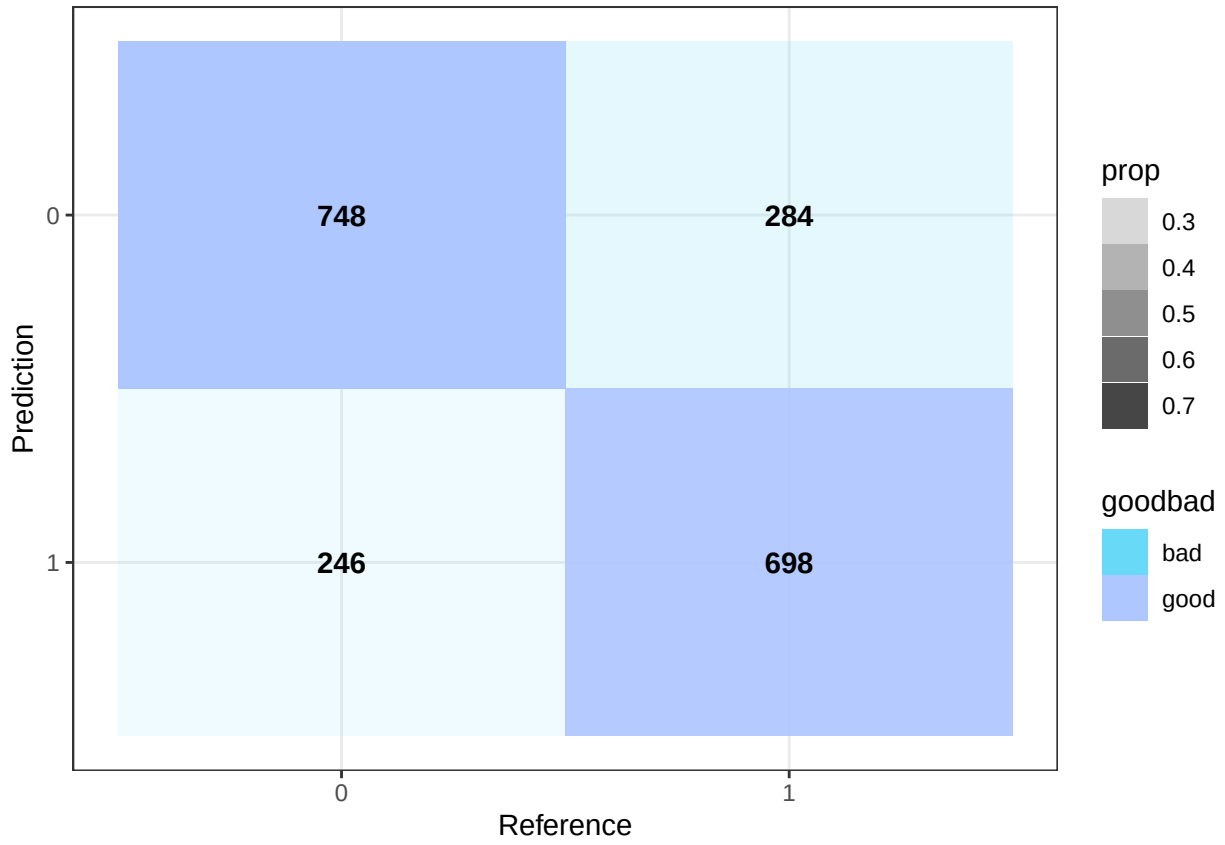
```
pred_lasso_rf <- rep(0, nrow(sel_test))
pred_lasso_rf[pos_prob > best_cut] <- 1
```

```r
pred_lasso_rf <- factor(pred_lasso_rf)
confusionMatrix(pred_lasso_rf, sel_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 748 284
##          1 246 698
##
##                Accuracy : 0.7318
##                  95% CI : (0.7117, 0.7512)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4634
##
##  Mcnemar's Test P-Value : 0.108
##
##             Sensitivity : 0.7525
##             Specificity : 0.7108
##          Pos Pred Value : 0.7248
##          Neg Pred Value : 0.7394
##              Prevalence : 0.5030
##          Detection Rate : 0.3785
##    Detection Prevalence : 0.5223
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_lasso_rf, sel_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by proportional outcomes within reference groups (see
ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



將篩過的變數放入 random forest 後，n.tree=500 也就是做 500 棵不剪枝的 tree，mtry=3 每棵 tree 放進去做的變數都是由 9 個 covariate 隨機選 3 個的效果最好
其 Accuracy 為 0.7318，而 Sensitivity 達到了 0.7525，Specificity 則是 0.7108，此結果與 knn、logistic regression，結果比較相似，以 Accuracy 來說，比 logistic regression and 放入所有變數的 random forest 還來的低，但由如上面的解釋，我們認為其 Accuracy 已經到達了極限，也很難再有所突破。

gboost

```
# boost_sel <- train(blueWins ~ .,
#                    data = sel_train,
#                    method = "gbm",
#                    tuneGrid = expand.grid(.n.trees = c(500, 1000, 1500),
#                                           .interaction.depth = c(3, 4, 5),
#                                           .shrinkage = c(0.01, 0.05, 0.1),
#                                          .n.minobsinnode = c(10, 50, 100)),
#                    preProc = c("center", "scale"),
#                    trControl = train_control)
#saveRDS(boost_sel, file = "boost_sel.rds")
boost_sel <- readRDS("boost_sel.rds")
boost_sel$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 10     500                 4      0.01             10
```

```
pred_boost_sel <- predict(boost_sel , sel_test)
```

```r
confusionMatrix(pred_boost_sel , sel_test$blueWins)
```

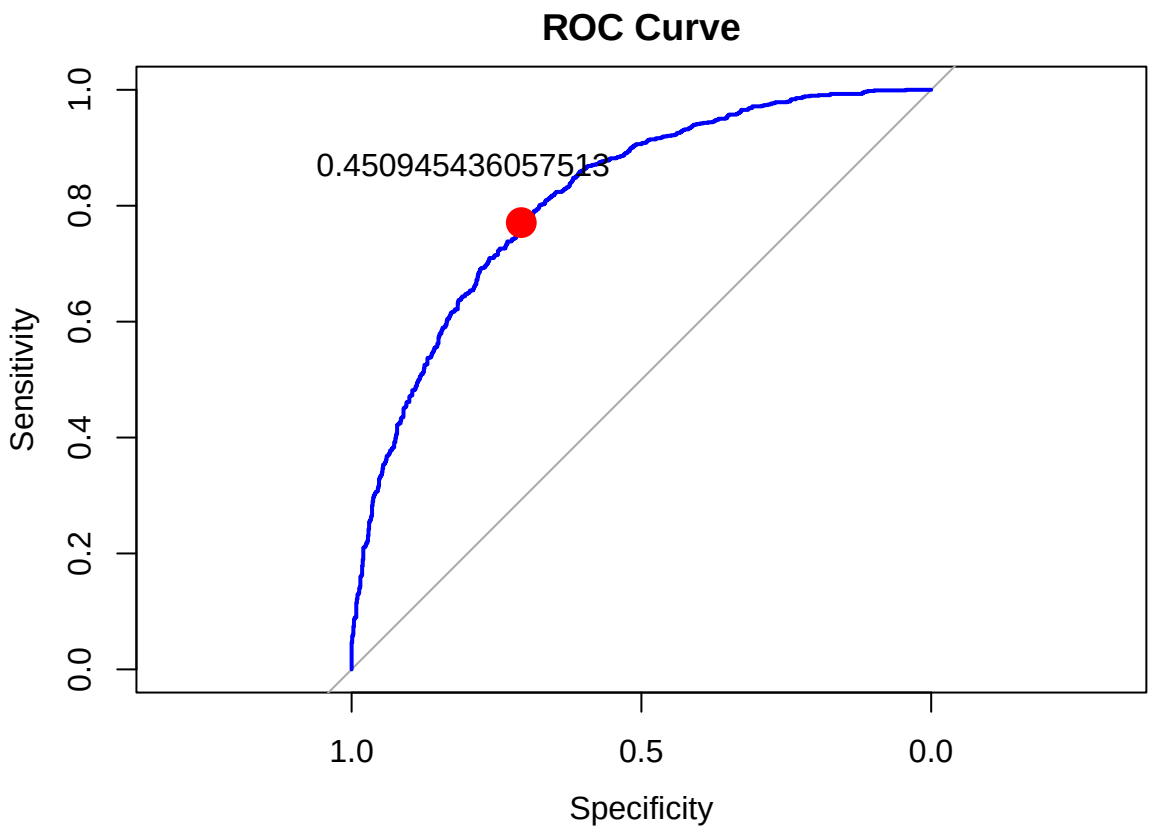
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 743 276
##          1 251 706
##
##                Accuracy : 0.7333
##                  95% CI : (0.7132, 0.7527)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4665
##
##  Mcnemar's Test P-Value : 0.2958
##
##             Sensitivity : 0.7475
##             Specificity : 0.7189
##          Pos Pred Value : 0.7291
##          Neg Pred Value : 0.7377
##              Prevalence : 0.5030
##          Detection Rate : 0.3760
##    Detection Prevalence : 0.5157
##       Balanced Accuracy : 0.7332
##
##        'Positive' Class : 0
##
```

```r
pred_prob <- predict(boost_sel, sel_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(sel_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.4509454
```

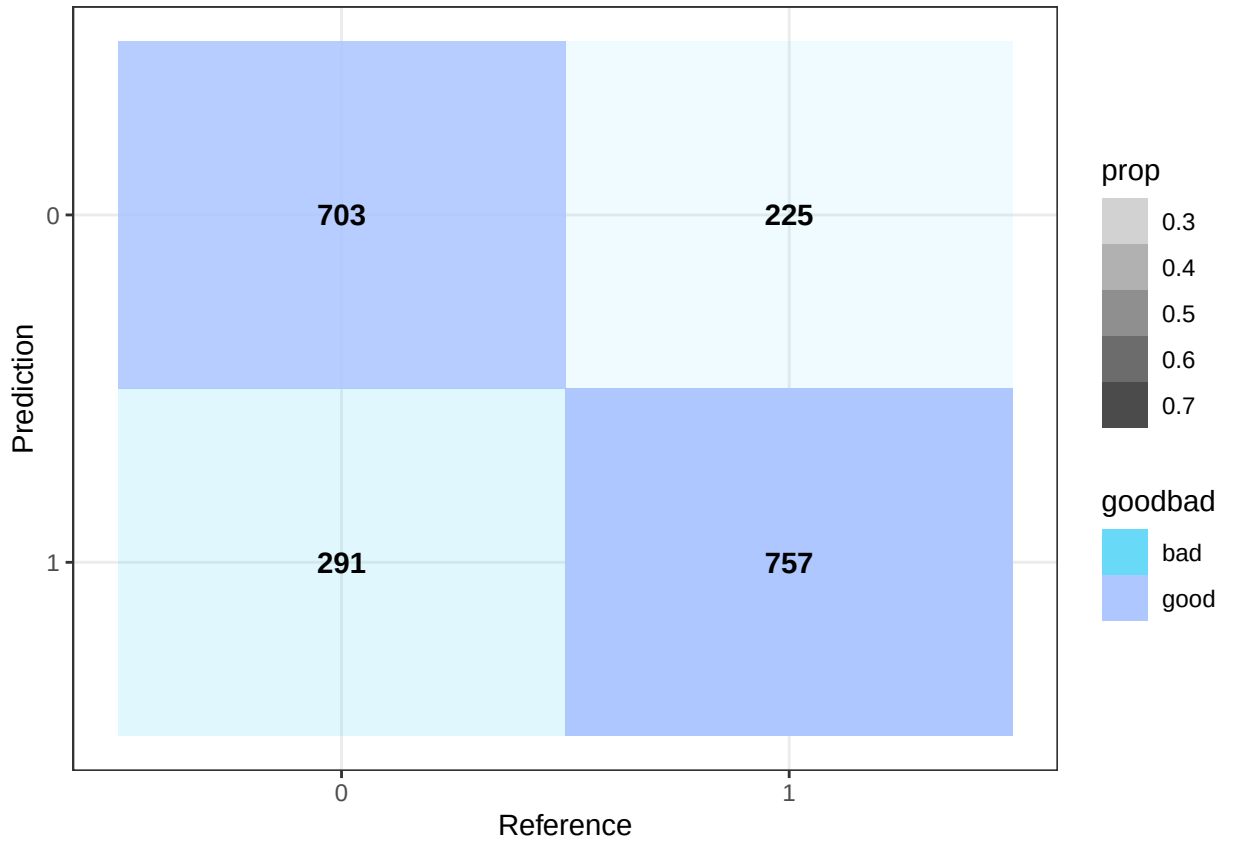
```
pred_boost_sel_roc <- rep(0, nrow(sel_test))
pred_boost_sel_roc[pos_prob > best_cut] <- 1
pred_boost_sel_roc <- factor(pred_boost_sel_roc)
confusionMatrix(pred_boost_sel_roc, sel_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 703 225
##          1 291 757
##
##                Accuracy : 0.7389
##                  95% CI : (0.7189, 0.7581)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4779
##
##  Mcnemar's Test P-Value : 0.004217
##
##             Sensitivity : 0.7072
```

```
##             Specificity : 0.7709
##          Pos Pred Value : 0.7575
##          Neg Pred Value : 0.7223
##              Prevalence : 0.5030
##          Detection Rate : 0.3558
##    Detection Prevalence : 0.4696
##       Balanced Accuracy : 0.7391
##
##        'Positive' Class : 0
##
```

```
table <- data.frame(confusionMatrix(pred_boost_sel_roc, sel_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



將篩過的變數放入 gboost 後，經過 cross validation 選出來的 tunning parameter 為 n.trees=500 也就是總共有 500 棵不剪枝的樹，interaction.depth=4 也就是樹的深度為 4，shrinkage=0.01 也就是 gradient 一次走 0.01

步，n.minobsinnode=10 也就是每個節點允許的最小觀測數是 10
其 Accuracy 為 0.7389，而 Sensitivity 達到了 0.7072，Specificity 則是 0.7709，此結果與 knn、logistic regression，結果比較相似，以 Accuracy 來說，比 logistic regression and random forest 還要差，但由如上面的解釋，我們認為其 Accuracy 已經到達了極限，也很難再有所突破。

全部變數 gboost

```
# boost_all <- train(blueWins ~ . ,
#                    data = all_train ,
#                    method = "gbm",
#                    tuneGrid = expand.grid(.n.trees = c(500, 1000, 1500),
#                                           .interaction.depth = c(3, 4, 5),
#                                           .shrinkage = c(0.01, 0.05, 0.1),
#                                           .n.minobsinnode = c(10, 50, 100)),
#                    preProc = c("center" , "scale"),
#                    trControl = train_control)
#saveRDS(boost_all, file = "boost_all.rds")
boost_all <- readRDS("boost_all.rds")
boost_all$finalModel
```

```
## A gradient boosted model with bernoulli loss function.
## 500 iterations were performed.
## There were 39 predictors of which 29 had non-zero influence.
```

```
boost_all$bestTune
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 1     500                 3      0.01             10
```

```
pred_boost_all <- predict(boost_all , all_test)
confusionMatrix(pred_boost_all , all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 752 277
##          1 242 705
##
##                Accuracy : 0.7373
##                  95% CI : (0.7173, 0.7566)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4746
##
##  Mcnemar's Test P-Value : 0.1356
##
##             Sensitivity : 0.7565
##             Specificity : 0.7179
##          Pos Pred Value : 0.7308
##          Neg Pred Value : 0.7445
##              Prevalence : 0.5030
##          Detection Rate : 0.3806
##    Detection Prevalence : 0.5207
##       Balanced Accuracy : 0.7372
##
```

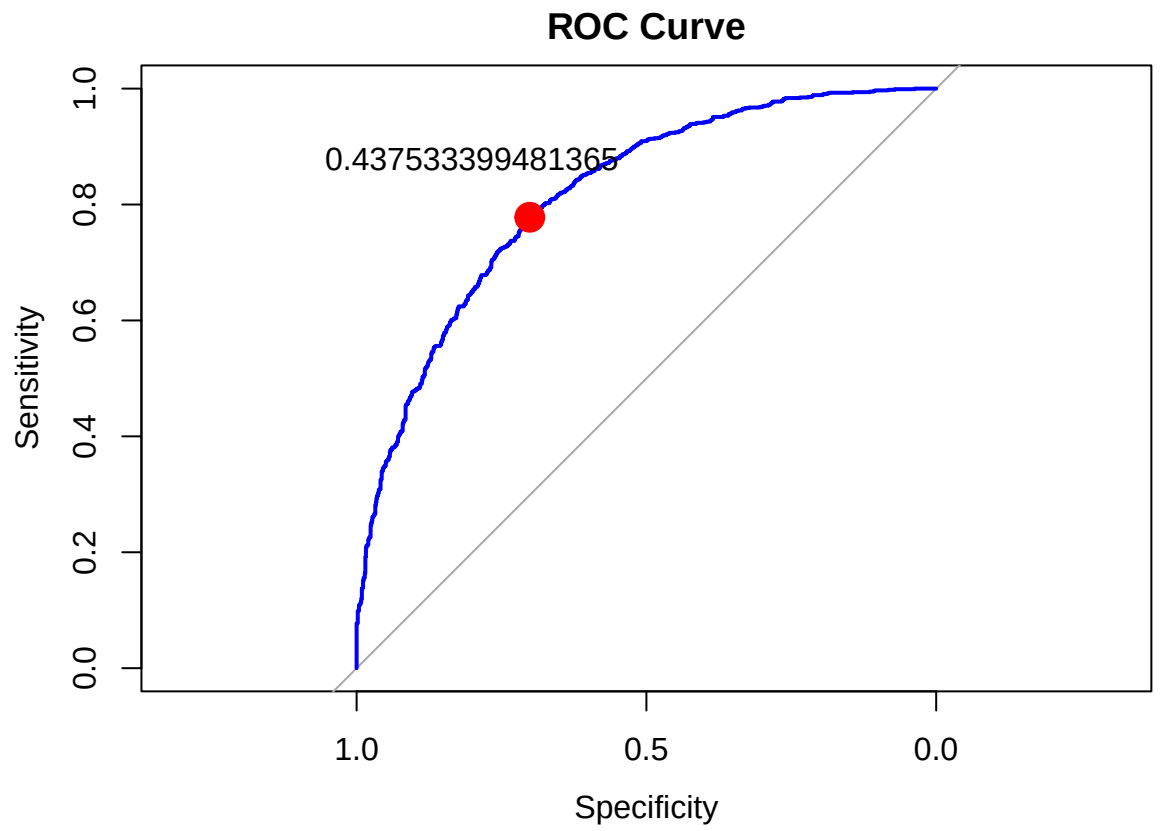
```
##        'Positive' Class : 0
##
```

```
pred_prob <- predict(boost_all, all_test, type = "prob")
pos_prob <- pred_prob[, "1"]
roc_data <- roc(all_test$blueWins, pos_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_data, col = "blue", main = "ROC Curve")
points(coords(roc_data, "best")$specificity, coords(roc_data, "best")$sensitivity, cex = 2, pch = 19, c
text(coords(roc_data, "best")$specificity+0.1, coords(roc_data, "best")$sensitivity+0.1, coords(roc_data
```




```
best_cut <- coords(roc_data, "best")$threshold
best_cut
```

```
## [1] 0.4375334
```

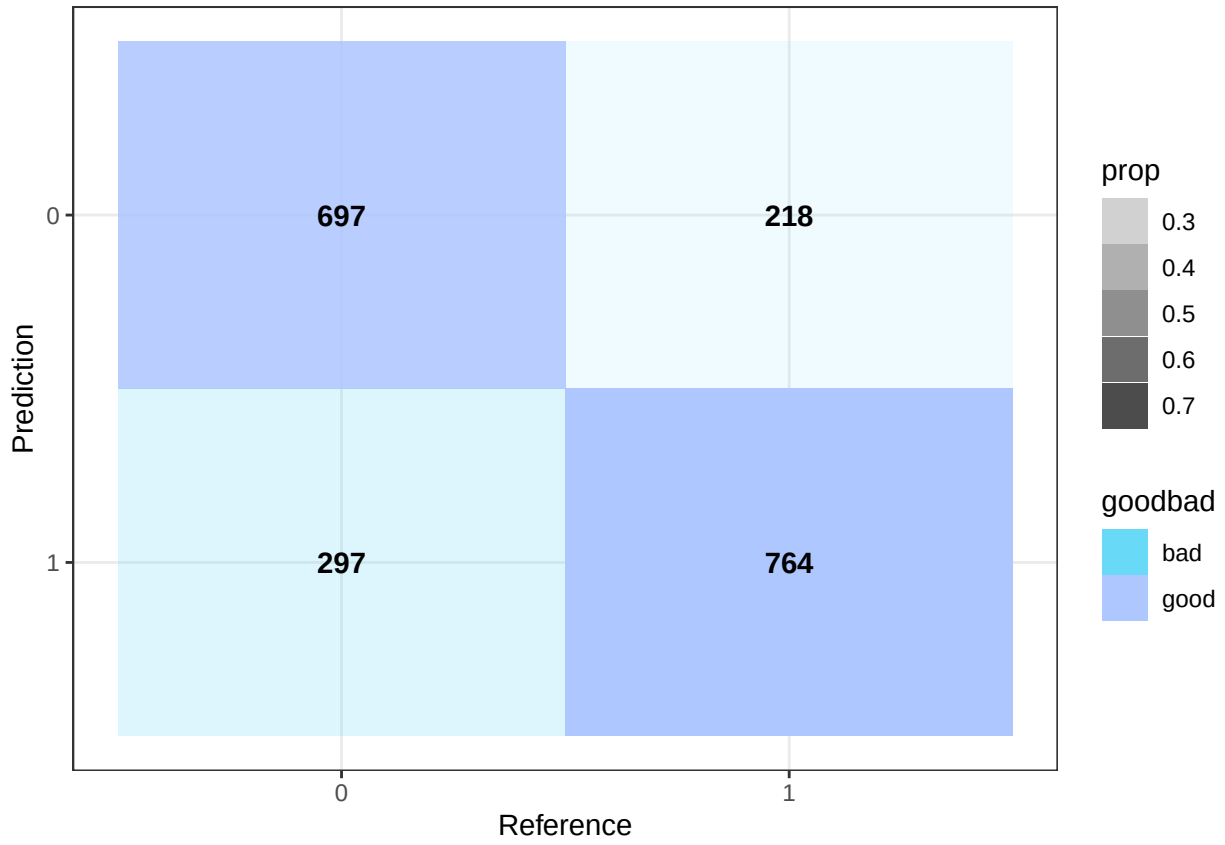
```
pred_boost_all_roc <- rep(0, nrow(all_test))
pred_boost_all_roc[pos_prob > best_cut] <- 1
pred_boost_all_roc <- factor(pred_boost_all_roc)
confusionMatrix(pred_boost_all_roc, all_test$blueWins)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 697 218
##          1 297 764
##
##                Accuracy : 0.7394
##                  95% CI : (0.7194, 0.7586)
##     No Information Rate : 0.503
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.479
##
##  Mcnemar's Test P-Value : 0.000588
##
##             Sensitivity : 0.7012
##             Specificity : 0.7780
##          Pos Pred Value : 0.7617
##          Neg Pred Value : 0.7201
##              Prevalence : 0.5030
##          Detection Rate : 0.3527
##    Detection Prevalence : 0.4631
##       Balanced Accuracy : 0.7396
##
##        'Positive' Class : 0
##
```

```r
table <- data.frame(confusionMatrix(pred_boost_all_roc, all_test$blueWins)$table)
plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference, "good","bad")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

ggplot(data = plotTable, mapping = aes(x = Reference, y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(good = "#afc7ff", bad = "#68daf8")) +
  theme_bw() +
  xlim(levels(table$Reference))+
  ylim(rev(levels(table$Prediction)))
```



將所有的變數放入 gboost 後，經過 cross validation 選出來的 tunning parameter 為 n.trees=500 也就是總共有 500 棵不剪枝的樹，interaction.depth=3 也就是樹的深度為 3，shrinkage=0.01 也就是 gradient 一次走 0.01 步，n.minobsinnode=10 也就是每個節點允許的最小觀測數是 10
將所有的變數放入 gboost 後，其 Accuracy 為 0.7394，而 Sensitivity 達到了 0.7012，Specificity 則是 0.7780，以 Accuracy 來說，比 logistic regression and random forest 還要差，但由如上面的解釋，我們認為其 Accuracy 已經到達了極限，也很難再有所突破。

conclusion

```
# varImp(fit_knn$finalModel)
# varImp(fit_knn_all$finalModel)
# varImp(fit_glmnet)
# varImp(tree_sel_cv)
# varImp(tree_all_cv)
# varImp(rf_lasso_cv$finalModel)
# varImp(rf_cv$finalModel)
# varImp(boost_sel)
# varImp(boost_all)
auc <- c(0.7328,0.5602,0.746,0.71,0.7191,0.7318,0.7404,0.7389,0.7373)
sen <- c(0.7505,0.5704,0.7072,0.7797,0.8159,0.7525,0.7485,0.7072,0.7565)
spe <- c(0.7149,0.5499,0.7851,0.6395,0.6212,0.7108,0.7322,0.7709,0.7179)
most_imp_var <- c("NA","NA","blueTotalGold, redTotalGold","redTotalGold, blueTotalGold","blueGoldDiff, 
final_conclusion <- data.frame(auc = auc, sen = sen, spe = spe, most_important_variable = most_imp_var)
rownames(final_conclusion) <- c("knn_sel","knn_all","logistic","tree_sel","tree_all","rf_sel","rf_all",
kable(final_conclusion)
```

	auc	sen	spe	most_important_variable
knn_sel	0.7328	0.7505	0.7149	NA
knn_all	0.5602	0.5704	0.5499	NA
logistic	0.7460	0.7072	0.7851	blueTotalGold, redTotalGold
tree_sel	0.7100	0.7797	0.6395	redTotalGold, blueTotalGold
tree_all	0.7191	0.8159	0.6212	blueGoldDiff, redGoldDiff
rf_sel	0.7318	0.7525	0.7108	redTotalGold, blueTotalGold
rf_all	0.7404	0.7485	0.7322	blueGoldDiff, redGoldDiff
gboost_sel	0.7389	0.7072	0.7709	redTotalGold, blueTotalGold
gboost_all	0.7373	0.7565	0.7179	redGoldDiff, blueGoldDiff

回顧我們提出的要解決的兩個問題
1. 如何用十分鐘的數據預測遊戲結束時的勝敗結果 (決定哪個模型最好)
2. 如何快速的判斷出一場遊戲最終的勝敗結果 (決定哪個變數最重要)
這是我們所做的所有模型的總整理表格，其中我們想要在準確率 auc 高的情況下還能夠兼顧判斷正確率也都高，因為在藍方贏時誤判成紅方贏的帶來的損失，和紅方贏時誤判成藍方贏的帶來的損失是相同的，所以我們 sensitivity、specificity 兩個值同等重要，在這個基礎之下我們可以回答提出的兩個問題
1. 我們會使用 rf_all 也就是 39 個 covariate 全部放進去 fit random forest 的模型，因為它不僅準確率高，sensitivity 和 specifcity 也同時都很高，不會有一個特別低。
2 一樣由表格可以看出幾乎所有模型選出來的前兩個重要變數都相同，都和藍方經濟和紅方經濟有關，而這兩個變數組合起來就可以得到紅方經濟差、藍方經濟差這兩個變數，所以最重要的變數就是雙方的經濟差，只要觀察雙方經濟差就可以快速判斷這場的遊戲勝敗結果。

reference

https://www.kaggle.com/code/xiyuewang/lol-how-to-win/input