Team TOO HARD: 101C Final Report

Junpeng Jiang; Wenxin Zhou; Rosy Zhou

Data Cleaning

- After careful examination, we did not see any missing values in both training and testing datasets. However, we found some repetitive columns in the data when we went through the data. For example, we found values of "VT.TS.fga" are exactly the same as values of "HT.OTS.fga". "VT.TS.fga" represents total 'field goals attempted' by Visiting Team; and "HT.OTS.fga" represents total 'field goals attempted' by Home Team's opposing team, which is just VT. The same logic applies to VT.TA.xxx versus HT.OTA.xxx, and VT.S1.xxx versus HT.OS1.xxx, etc. Therefore, we cleaned up all columns that has ".O" in it; this removes all repetitive columns.
- We also removed id, game id, date, in the train dataset and remove game id, date in the test dataset. Because these variables are unique in each row and shouldn't be significant indicators of the teams' future wins or losses. After cleaning the data, the train set has dimensions of 9520 rows and 113 columns and the test set has dimensions of 1648 rows and 113 columns.

New Variables Description (4 new variables)

Variable pht

- pht represents the prior winning rate of a team when it plays as a home team across all training data. For example, if team 1 played 100 games as home team in training data, and it won 50 games, all rows having team 1 as HT will have the pht value 0.5.

Variable pvt

- Similarly, pvt represents the prior winning rate of a team playing as visiting team across all training data. For example, if team 1 played 100 games as the visiting team in the training data, and it won 50 games, all rows having team 1 as VT will have the pvt value 0.5.

Variable winthisvt

- winthisvt is a home team's winning rate when it is against a specific visiting team. It is calculated using all training data. For example, if a row has HT=Team 1, VT=Team 2, winthisvt = 0.9, then it means of all games Team 1 plays as HT against Team 2 in training data, Team 1 wins Team 2 90% of the time.

Variable p

- Variable p represents the total prior winning probability for a team. It is the winning rate of this team as both the visiting team and the home team. So p represents the overall winning rate of a team based on the data in training dataset.
- For example, if a team plays 100 games; 40 of them are home team and and 60 as the visiting team. Among these 100 games this team wins 60 times. So the p for this team is 0.6.

Adding New Variable Reasoning

- Our final goal is to predict whether a home team wins. We believe adding variables related to the winning rate will assist in our prediction. Our first step is to add each team's winning rate as HT/VT. This is a reasonable step to take, because these winning rates show historically how probable a team wins as either HT or VT. In addition, calculating the total winning rate of a team could also be effective in forecasting a team's win or loss in future games.
- To explore winning rate even further, we decided to look at the specific rivalry history. We believe historically how well team 1(HT) played against team 2 could be a very strong win/loss indicator of team 1's future game as HT against team 2.

Model Description

- Model: Ridge (HTWins ~ 113 original variables + pht + pvt + p)

Explanation of the Models.

- We applied a variety of classification algorithms, including logistic regression, ridge regression, lasso regression, gradient boosting, random forest and partial least squares regression, and ensembling. Among all the models we have tried, ridge performs the best. It is most reasonable because there are too many variables, and in the ridge and lasso models having too many variables is penalized. They are more restrictive and less flexible than the least squares model. Furthermore, the logistic, LDA, QDA models require linear independence between variables but our variables have strong collinearality.
- While testing out the models, we tried to explore whether our added 4 variables significantly improve prediction. By spliting training data and applying cross validation, we compared misclassfication rates before and after adding these variables.

```
# Note: The codes for calculating these misclassification rates are in the R Code section
Misclassification_Rate <- c(0.3206349,0.3174603,0.2853175)
MODEL <- c("Ridge Original","Ridge + pht/pvt/p","Ridge + all four")
result <- cbind(MODEL,Misclassification_Rate)
print(as.data.frame(result))
```

```
##               MODEL Misclassification_Rate
## 1    Ridge Original              0.3206349
## 2 Ridge + pht/pvt/p              0.3174603
## 3  Ridge + all four              0.2853175
```

- Our finding was that (winthisvt,pht,pvt,p) all improves prediction.
- Adding winthisvt, we had reached the score of 0.715 in our own testing(by spliting training data). However, adding winthisvt made our trial submission scores very low(0.638). We reached a conclusion that this variable winthisvt is so strong that it overfits. We finally decided to remove winthisvt from our submission model.

Best Misclassification Rate with Our Model

- Model: Ridge (HTWins ~ 113 original variables + pht + pvt + p)
- Submission score: 0.67233(Private) 0.66383(Public)
- In our own testing (cv in training dataset), we had a score of (1-0.3174603) = 0.6825397.

R Code Part 1

- Clarification: On kaggle score board, we have 0.67718. This score is obtained by fitting a ridge model on the original training dataset without data cleaning and new added variables. It has the second highest public score(0.67961), and highest private score(0.67718) on our end. But we do think it is not reasonable to choose this model to talk about in our report, because there is minimal work done with this approach. So we discussed how to fit another ridge regression on cleaned data with new variables to show to work we have done, although this approach gives a lower private score on kaggle(0.67354).
- The following code produce the same result as the our submission with privated score(0.67718), it is ridge regression on original training dataset without data cleaning and without any new variables.

```
train <- read.csv("train.csv")
test<-read.csv("test.csv")
usetrain<-train
usetest<-test

x <- model.matrix(HTWins~.,data = usetrain)
y <- ifelse(train$HTWins=="Yes", 1, 0)

library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-18
```

```
lambda <- 10^seq(10,-2,length=100)
ridge <- glmnet(x,y,alpha = 0,lambda = lambda)
cvridge <- cv.glmnet(x,y,alpha = 0,lambda = lambda)
#plot(cvridge)
best <- cvridge$lambda.min
bestridge <- glmnet(x,y,alpha = 0,lambda = best)
pred_ridge<-predict(bestridge,newx=model.matrix(~.,data = usetest),s=best,type="response")
pred_ridge<-ifelse(pred_ridge>0.5,"Yes","No")
result<-cbind(usetest$id,pred_ridge)
colnames(result)<-c("id","HTWins")
#write.csv(result,"simple_method_submission.csv")
```

R Code Part 2

- Simple_method_submission should produce the exact same submission of our highest private score on kaggle (0.67718), the rest of the code from here reflects the model discussed in our report.

```
train <- read.csv("train.csv")
test<-read.csv("test.csv")

#DATA CLEANING
```

```r
#clean repeated columns which has ".0" in its column names
train_names<- colnames(train)
which_col_repetitive<- which(grepl(".0", train_names, fixed=TRUE)) #return No.col
train<- train[,-which_col_repetitive]

#remove id, gameid, date
train<- train[,-c(1,2,8)]

train$HTWins<-ifelse(train$HTWins=="Yes", 1, 0)

#cleaning for test data
test_names<- colnames(test)
which_col_repetitive1<- which(grepl(".0", test_names, fixed=TRUE)) #return No.col
test<- test[,-which_col_repetitive1]
test<- test[,-c(2,7)]

#create list of winning teams and home teams for train set
vt<-unique(train$VT)
ht<-unique(train$HT)

#we can see the teams in vt and ht are the same.
setdiff(vt,ht)
```

```
## character(0)
```

```r
setdiff(ht,vt)
```

```
## character(0)
```

```r
#vt ht for test set
vt1<-unique(train$VT)
ht1<-unique(train$HT)

setdiff(vt1,ht1)
```

```
## character(0)
```

```r
setdiff(ht1,vt1)
```

```
## character(0)
```

```r
setdiff(vt1,vt)
```

```
## character(0)
```

```r
setdiff(ht1,ht)
```

```
## character(0)
```

```r
#thus all teams exist in HT and VT in both train and test

#prior win rate for each team

teams<-vt
prior_vt<-rep(NA,length(teams))
prior_ht<-rep(NA,length(teams))
prior_total<-rep(NA,length(teams))

for (i in 1:length(teams)){
  team<-teams[i]
  p=mean(train[which(train$VT==team),1])
  prior_vt[i]=p
  p2=mean(train[which(train$HT==team),1])
  prior_ht[i]=p2
  prior_total[i]=(p+p2)/2
}


train$pvt<-NA
train$pht<-NA
train$p<-NA
for (i in 1:length(teams)){
  team=teams[i]
  train[which(train$VT==team),"pvt"]=prior_vt[i]
  train[which(train$HT==team),"pht"]=prior_ht[i]
  train[which(train$HT==team),"p"]=prior_total[i]
}

test$pvt<-NA
test$pht<-NA
test$p<-NA
for (i in 1:length(teams)){
  team=teams[i]
  test[which(test$VT==team),"pvt"]=prior_vt[i]
  test[which(test$HT==team),"pht"]=prior_ht[i]
  test[which(test$HT==team),"p"]=prior_total[i]
}


priortable<-data.frame("team"=teams,"prior_ht"=prior_ht,"prior_vt"=prior_vt,"prior_total"=prior_total)
priortable
```

```
##    team  prior_ht  prior_vt prior_total
## 1  BATA 0.7223720 0.4435262   0.5829491
## 2  MANL 0.6388140 0.5890411   0.6139276
## 3  RICE 0.6413043 0.5609756   0.6011400
## 4  PARQ 0.4931880 0.6767956   0.5849918
## 5  PASV 0.5962060 0.6233062   0.6097561
## 6  SJNK 0.6361186 0.5351351   0.5856269
## 7  BCCS 0.5121294 0.6939891   0.6030592
## 8  BATG 0.4054795 0.7235772   0.5645283
## 9  IMUS 0.6500000 0.4972678   0.5736339
```

```
## 10 MTLP 0.6380697 0.5757576   0.6069136
## 11 BOAN 0.4817927 0.6567568   0.5692747
## 12 CEBS 0.5510753 0.6621253   0.6066003
## 13 HZTL 0.6756757 0.4876712   0.5816735
## 14 KABO 0.6555556 0.5714286   0.6134921
## 15 LLOL 0.5013699 0.7178082   0.6095890
## 16 LUCA 0.6495957 0.5464481   0.5980219
## 17 MIND 0.8243243 0.3621622   0.5932432
## 18 QUEZ 0.6005435 0.6016484   0.6010959
## 19 NAVO 0.5786517 0.5589041   0.5687779
## 20 BASI 0.4739726 0.6933702   0.5836714
## 21 PANG 0.4277778 0.7095890   0.5686834
## 22 AJAX 0.5725806 0.6108108   0.5916957
## 23 RIZL 0.6084507 0.5558511   0.5821509
## 24 SOCC 0.6765499 0.5835616   0.6300558
## 25 CALO 0.6436464 0.6104972   0.6270718
## 26 SPVT 0.5611111 0.5828729   0.5719920
```

```
ptotal<-mean(train$HTWins)
ptotal
```

```
## [1] 0.5933824
```

```
#################### create a list of winrates tables
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
```

```
newrt <- train %>% select(HTWins,HT,VT)
newrt$HTWins <- newrt$HTWins
#View(newrt)
facerate <- list(NA,26)
for (i in 1:26) {
  facerate[[i]] <- newrt[newrt$HT==unique(newrt$HT)[i],] %>%
    group_by(VT) %>%
    summarise(face = mean(as.numeric(HTWins)))
  facerate[[i]] <-as.data.frame(facerate[[i]])
}
####################### add rates in train
train$winthisvt <- rep(NA,9520)
for (i in 1:9520) {
  which_ht <- which(unique(newrt$HT) == train$HT[i])
  use_this_table <- facerate[[which_ht]]
```

```
  which_vt <- train$VT[i]
  add_rate <- use_this_table[use_this_table$VT == which_vt,2]
  train$winthisvt[i] <- add_rate
}
######################### add rates in test
test$winthisvt <- rep(NA,1648)
for (i in 1:1648) {
  which_ht <- which(unique(newrt$HT) == test$HT[i])
  use_this_table <- facerate[[which_ht]]
  which_vt <- test$VT[i]
  add_rate <- use_this_table[use_this_table$VT == which_vt,2]
  test$winthisvt[i] <- add_rate
}

#PREDICTION ON TEST DATASET
#prediction after data cleaning and adding 3 new variables
#this should produce the exact same results as the 12.7.csv file
usetrain<-train[,-117]
usetest<-test[,-117]
####### Ridge #######
set.seed(731)
x <- model.matrix(HTWins~.,data = usetrain)
y <- usetrain$HTWins

library(glmnet)
lambda <- 10^seq(10,-2,length=100)
ridge <- glmnet(x,y,alpha = 0,lambda = lambda)
cvridge <- cv.glmnet(x,y,alpha = 0,lambda = lambda)
#plot(cvridge)
best <- cvridge$lambda.min
bestridge <- glmnet(x,y,alpha = 0,lambda = best)
pred_ridge<-predict(bestridge,newx=model.matrix(id~.,data = usetest),s=best,type="response")
pred_ridge<-ifelse(pred_ridge>0.5,"Yes","No")
result<-cbind(usetest$id,pred_ridge)
colnames(result)<-c("id","HTWins")
write.csv(result,"result.csv")
```

R Code for the misclassification rates in the report

```
#split the training data to test the models
set.seed(1)
index <- sample(1:9520,7000)
usetrain1 <- train[index,]
usetest1 <- train[-index,]
#View(train)

#Ridge model on 113 original variables
usetrain<-usetrain1[,c(1:113)]
usetest<-usetest1[,c(1:113)]
set.seed(731)
x <- model.matrix(HTWins~.,data = usetrain)
```

```
y <- usetrain$HTWins
library(glmnet)
lambda <- 10^seq(10,-2,length=100)
ridge <- glmnet(x,y,alpha = 0,lambda = lambda)
cvridge <- cv.glmnet(x,y,alpha = 0,lambda = lambda)
#plot(cvridge)
best <- cvridge$lambda.min
bestridge <- glmnet(x,y,alpha = 0,lambda = best)
pred_ridge<-predict(bestridge,newx=model.matrix(HTWins~.,data = usetest),s=best,type="response")
pred_ridge<-ifelse(pred_ridge>0.5,1,0)
table(usetest$HTWins,pred_ridge)
```

```
##    pred_ridge
##        0    1
##   0  474  496
##   1  312 1238
```

```
#missclassification rate is 0.3206349
mean(usetest$HTWins!=pred_ridge)
```

```
## [1] 0.3206349
```

```
orig113<-0.3206349

#Ridge model on 113 original variables +3 new variables
usetrain<-usetrain1[,-117]
usetest<-usetest1[,-117]
set.seed(731)
x <- model.matrix(HTWins~.,data = usetrain)
y <- usetrain$HTWins
library(glmnet)
lambda <- 10^seq(10,-2,length=100)
ridge <- glmnet(x,y,alpha = 0,lambda = lambda)
cvridge <- cv.glmnet(x,y,alpha = 0,lambda = lambda)
#plot(cvridge)
best <- cvridge$lambda.min
bestridge <- glmnet(x,y,alpha = 0,lambda = best)
pred_ridge<-predict(bestridge,newx=model.matrix(HTWins~.,data = usetest),s=best,type="response")
pred_ridge<-ifelse(pred_ridge>0.5,1,0)
table(usetest$HTWins,pred_ridge)
```

```
##    pred_ridge
##        0    1
##   0  476  494
##   1  306 1244
```

```
#missclassification rate is 0.3174603
mean(usetest$HTWins!=pred_ridge)
```

```
## [1] 0.3174603
```

```
orig113and3<-0.3174603

#Ridge model on 113 original variables + 4 new variables
usetrain<-usetrain1
usetest<-usetest1
set.seed(731)
x <- model.matrix(HTWins~.,data = usetrain)
y <- usetrain$HTWins
library(glmnet)
lambda <- 10^seq(10,-2,length=100)
ridge <- glmnet(x,y,alpha = 0,lambda = lambda)
cvridge <- cv.glmnet(x,y,alpha = 0,lambda = lambda)
#plot(cvridge)
best <- cvridge$lambda.min
bestridge <- glmnet(x,y,alpha = 0,lambda = best)
pred_ridge<-predict(bestridge,newx=model.matrix(HTWins~.,data = usetest),s=best,type="response")
pred_ridge<-ifelse(pred_ridge>0.5,1,0)
table(usetest$HTWins,pred_ridge)
```

```
##    pred_ridge
##        0    1
##   0  555  415
##   1  304 1246
```

```
#missclassification rate is 0.2853175
mean(usetest$HTWins!=pred_ridge)
```

```
## [1] 0.2853175
```

```
orig113and4<-0.2853175

rbind(orig113,orig113and3,orig113and4)
```

```
##                  [,1]
## orig113     0.3206349
## orig113and3 0.3174603
## orig113and4 0.2853175
```