Supervised Learning Classification - Statistical Learning on Self-Assessed Health Status

Jiazhou Liang

Background

This project is an improvement of the final project of the upper-year Statistic course "Statistical Learning - Classification" at the University of Waterloo by Bolun Cui and Joe Liang.

The dataset in this project corresponds to the responses in the German General Social Survey (ALLBUS) between 2005 and 2019. The target variable for machine learning is the last variable "health". It is an ordinal variable with five categories from 1 to 5 and represents the "self-asset financial health" of each survey response.

There are two parts of the dataset, "train.csv" and "test.csv". the samples in "train.csv" include "health" variables, which are used for model training. And "test.csv" does not have the "health" variable, which is used for examining the performance.

The goal of this project is to train a classification model on this dataset to classify survey responses into one of the financial health categories.

Preprocessing

Set Up

Importing training and testing dataset

```
# both datasets can be found on the project Github. The dataset should be in the same
# folder as this file for execution or change the below code into the complete path of each dataset
train <- read.csv("train.csv")
test <- read.csv("test.csv")
```

Removing the variables with more than 80% missing values

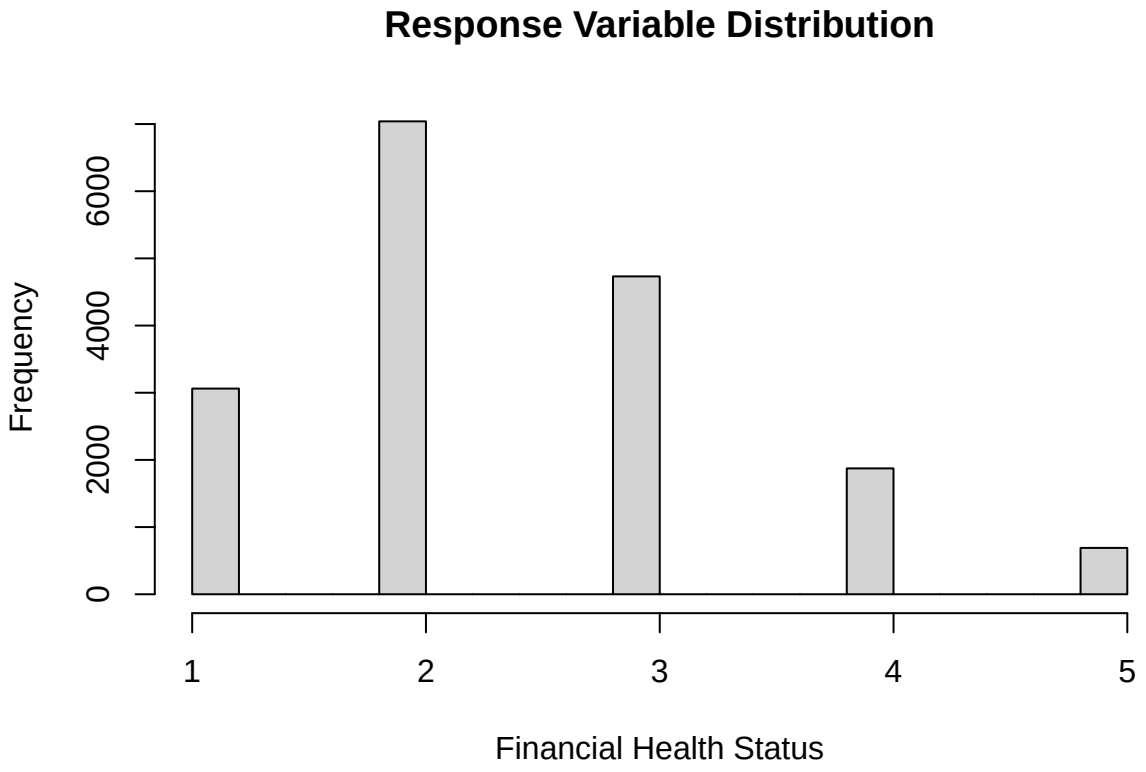
```
missing_threshold = 0.8
train = train[,-grep(TRUE,colSums(is.na(train))/nrow(train) > missing_threshold)]
```

Removing person id, interviewer id, and unique id from training data since they have very small correlation with the target variable health

```
train = train[,!(colnames(train) %in% c("x1181","uniqueid","personid"))]
```

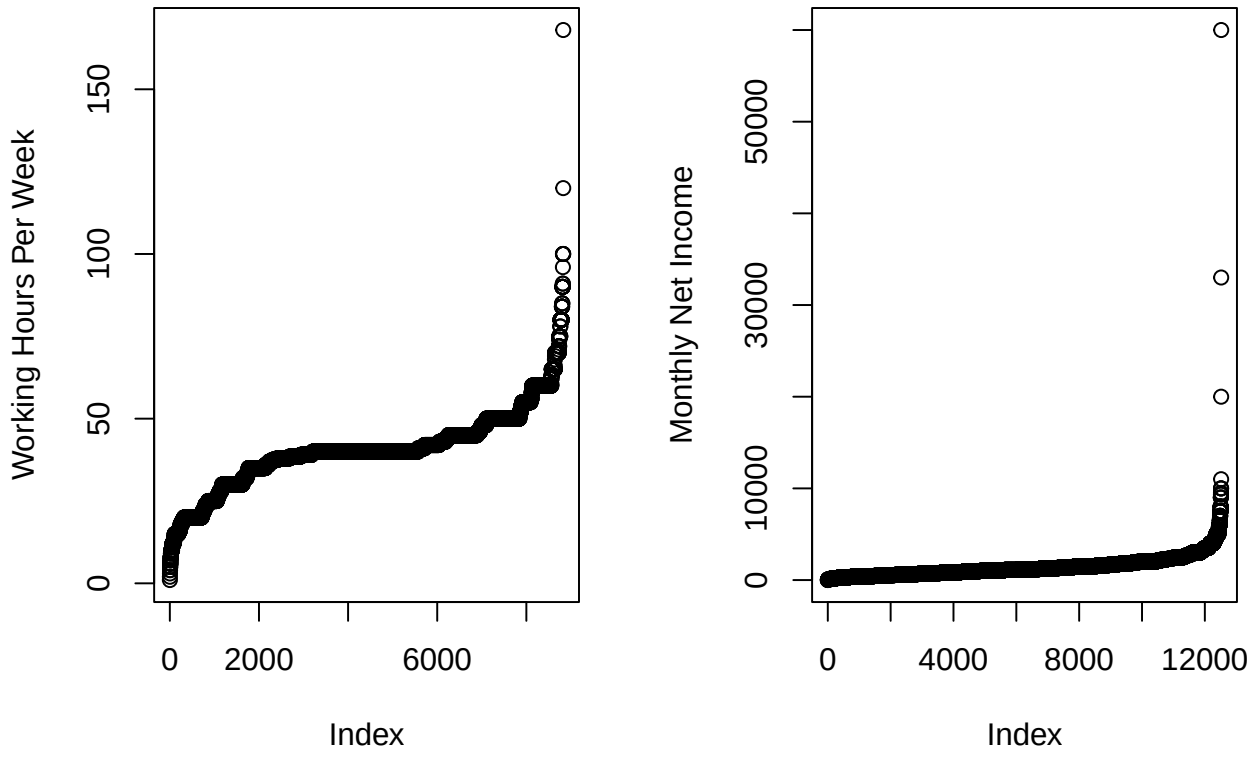
Highlights unusual circumstances from exploratory data analysis

```
hist(train$health, main = "Response Variable Distribution", xlab = "Financial Health Status")
```



During data analysis, we found there are two variables with unusual values

```
par(mfrow = c(1, 2))
plot(sort(train[,"x715"]), ylab = "Working Hours Per Week")
plot(sort(train[,"x723"]), ylab = "Monthly Net Income")
```



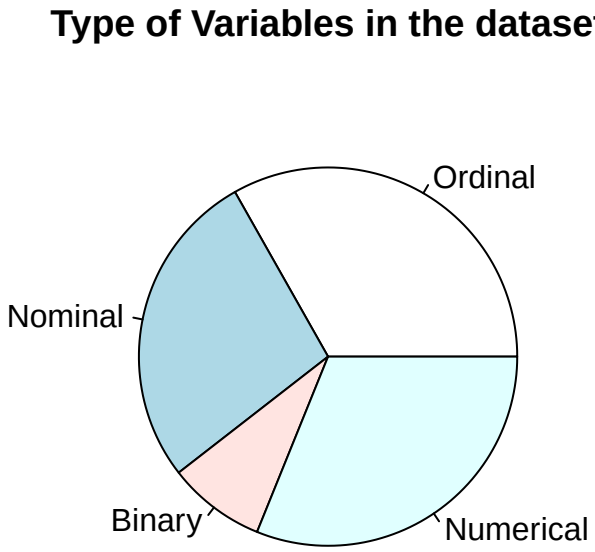
```
train <- train[-c(which(train,"x715"] > 150)),]
train <- train[-c(which(train,"x723"] > 50000)),]
```

After we manual divide all the variables into four subsets, each subset represents one of the binary, nominal, ordinal, and numerical variables in the training dataset.

```
##   numerical yes_no order non_order
## 1      year    x15    x1        x7
## 2  personid    x18    x2        x8
## 3       x22    x19    x3        x9
## 4       x23    x73    x4       x10
## 5       x24   x175    x5       x11
## 6       x25   x227    x6       x12
```

The pie chart represent the type of variables in the dataset

```
slice = c(length(order),length(non_order),length(yes_no), length(numerical))
label = c("Ordinal","Nominal","Binary","Numerical")
pie(slice,labels = label, main = "Type of Variables in the dataset")
```



Handling categorical and ordinal variables

Before converting all categorical and ordinal variables into factors, we merge the training and testing dataset to ensure they are encoded with the same levels

```
# Making sure the testing dataset has the same columns as the training dataset
test <-test[,(colnames(test) %in% colnames(train))]
# Ensuring both datasets has same dimension for merging
fake_health = rep(NA,length(test[,1]))
test$health = fake_health
# Merging
total = rbind(test,train)
```

Factoring categorical and ordinal variables

```
# factorize ordinal variable
for (names in order){
  if (names %in% colnames(total)){
total[,names] = factor(total[,names],order=TRUE)
```

```
}
}
# factorize non-ordinal variables
for (names in c(non_order)){
  if (names %in% colnames(total)){
total[,names] = factor(total[,names])
}
}
# factorize binary variables
for (names in c(yes_no)){
  if (names %in% colnames(total)){
total[,names] = factor(total[,names])
}
}

for (names in numerical){
  if (names %in% colnames(total)){
total[,names] = scale(total[,names])
}
}
```

Restoring training and testing data from merging

```
test_factorized = total[c(1:nrow(test)),-c(length(total))]
train_factorized = total[c((nrow(test)+1):length(total[,1])),]
```

Since we want to perform classification, factorizing "health" into ordinal variables

```
train_factorized$health <- factor(train_factorized$health, order = TRUE)
```

Missing value

Using mode to impute categorical variables and median to impute numerical variables. We also try several other imputing methods, for example, the MissForest. It uses a random forest algorithm to predict the missing value based on other information. However, there is no significant improvement of performance. But it has a much higher computational cost than imputing by median and mode.

```
library(randomForest)
```

```
## randomForest 4.7-1
```

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

```
train_imputed = na.roughfix(train_factorized)
test_imputed = na.roughfix(test_factorized)
```

Feature Engineering

During analysis, based on our domain knowledge, we derived a new x-variable. The first one is the average living space in m^2 per person in the household. This variable is derived by the ratio of living space in m^2 "x1134" and number of person in the household "x963".

```
train_imputed$average_space = train_imputed[,"x1134"]/train_imputed[,"x963"]
test_imputed$average_space = test_imputed[,"x1134"]/test_imputed[,"x963"]
```

Modeling

Splitting the train dataset into training data and validation data for tuning and stacking in further sections. The validation dataset will be used to test the performance of the model and prevents overfitting.

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Loading required package: lattice
```

```
set.seed(20288122)
trainIndex <- createDataPartition(train_imputed$health, p = .8, list = FALSE, times = 1)
validation_imputed <- train_imputed[-trainIndex,]
train_imputed <- train_imputed[trainIndex,]
```

There will be three different modelling techniques in this project, which are random forest, gradient boosting, and feed-forward neural network. We will tune the parameter for each model and test the performance using validation data with cross-entropy as the metric.

Random Forest

If a variable has more than 53 categories, there will be more than 2^{53}˘2 choices for the Random Forest algorithm in corresponding splits. This is an infeasible Computational Cost. Therefore, we decide to remove all the categorical variables with more 53 level.

```
morethan53_categories = c()
for (names in c(order,non_order)){
  if (names %in% colnames(train_imputed)){
    if (length(levels(train_imputed[,names])) > 53){
       morethan53_categories=c(morethan53_categories,names)
       }
    }
}
train_rf = train_imputed[,!(colnames(train_imputed) %in% morethan53_categories)]
validation_rf = validation_imputed[,!(colnames(validation_imputed) %in% morethan53_categories)]
test_rf = test_imputed[,!(colnames(test_imputed) %in% morethan53_categories)]
```

Since lower number of variables to choose from at each split will make resulted tree unreliable (high bias). But higher number will also cause overfitting (high variance). Therefore, we need to tune this number with validation dataset to find the balance of bias-variance trade-off.

```
set.seed(2022)
bestMtry <- tuneRF(train_rf[,-c(ncol(train_rf)-1)],train_rf$health,
                   mtryStart = sqrt(ncol(train_rf)),stepFactor = 1.5,
                   improve = 1e-5, ntree = 100)
```

Then, applying the Random Forest algorithm with above mtry and ntree = 2000

```
set.seed(2022)
rf_model <- randomForest(health~.,train_rf, mtry = 49, ntree = 100, importance=TRUE,do.trace=TRUE)
```

Examining the performance using validation dataset

```
pred_rf <- predict(rf_model,validation_rf, type = 'prob')
```

Checking the cross entropy of the validation dataset

```
result = c()
pred_rf[pred_rf == 0] = 0.0000000000001
for (i in c(1:length(validation_imputed$health))){
result = c(result,log(pred_rf[i,as.integer(validation_imputed$health[i])]))
}
-mean(result)
```

Nerual Network

Producing indicator variables for categorical variables

```
library(fastDummies)
default = getOption("warn")
options(warn = -1)
train_transformed = dummy_cols(train_imputed[-c(length(train_imputed)-1)],
                           select_columns = c(order,non_order,yes_no),
                           remove_selected_columns = TRUE)
validation_transformed = dummy_cols(validation_imputed[-c(length(validation_imputed)-1)],
                           select_columns = c(order,non_order,yes_no),
                           remove_selected_columns = TRUE)
test_transformed = dummy_cols(test_imputed,
                           select_columns = c(order,non_order,yes_no),
                           remove_selected_columns = TRUE)
options(warn=default)
```

Preprocessing the data frame for neural network model

```
library(keras)
train_neural = as.matrix(train_transformed)
validation_neural = as.matrix(validation_transformed)
test_neural = as.matrix(test_transformed)
train.y = to_categorical(as.numeric(train_imputed$health)-1)
```

```
## Loaded Tensorflow version 2.8.0
```

```
colnames(train_neural) <- NULL
colnames(test_neural) <- NULL
```

Performing a neural network model with two hidden layers, each has 128 and 256 neurals

```
set.seed(202122)
checkpoint_path <- "checkpoints/cp.ckpt"
cp_callback <- callback_model_checkpoint(
  filepath = checkpoint_path,
  save_weights_only = TRUE,
  save_best_only = TRUE,
  verbose = 0
)
inputshape = ncol(train_neural)
model <-keras_model_sequential()%>%
  layer_dense(units = 128, activation = "relu", input_shape =c(inputshape))%>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 256, activation = "relu")%>%
```

```
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 5, activation = "softmax")
model %>%
  compile(optimizer = optimizer_adam(lr = 0.001), loss = "categorical_crossentropy")
```

```
## Warning in backcompat_fix_rename_lr_to_learning_rate(...): the `lr` argument has
## been renamed to `learning_rate`.
```

```
model %>%
  fit(train_neural, train.y, epochs =  5, batch_size = 32, validation_split = 0.2,
      callbacks = list(cp_callback))

model %>% load_model_weights_tf(filepath = checkpoint_path)
```

We used Checkpoint function to save the model with lowest validation cross entropy during model Training. Then Examining the performance using our own validation dataset with cross entropy

```
result = c()
pred_nerual[pred_nerual == 0] = 0.0000000000001
for (i in c(1:length(validation_imputed$health))){
result = c(result,log(pred_nerual[i,as.integer(validation_imputed$health[i])]))
}
-mean(result)
```

Generalized Boosting Model

Remove all variables that has more than 53 categories for boosting algorithm.

```
train_gbm = train_imputed[,!(colnames(train_imputed) %in% morethan53_categories)]
validation_gbm = validation_imputed[,(colnames(validation_imputed)
                                      %in% colnames(train_gbm))]
test_gbm = test_imputed[,(colnames(test_imputed) %in% colnames(train_gbm))]
```

Tuning the maximum depth of each tree

```
library(gbm)
depth = c(1,3,5,7,10)
error = c()
for (i in depth){
model_gbm <- gbm(health~., data = train_gbm, distribution = "multinomial",
                 n.trees = 200,shrinkage = 0.03, n.cores = 3,
                 interaction.depth = i,verbose = TRUE)
pred = predict(model_gbm,validation_gbm[,-c(length(validation_gbm)-1)],type = "response")
pred = matrix(unlist(pred), ncol = 5)

result = c()
pred[pred == 0] = 0.0000000000001
for (i in c(1:length(validation_gbm$health))){
result = c(result,log(pred[i,as.integer(validation_gbm$health[i])]))
}

error = c(error,-mean(result))
}
error
```

We can see that when the maximum depth of tree is 3, the validation error is smallest. Therefore, we choose maximum depth of tree to be 3 and fit the boosting model.

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
model_gbm <- gbm(health~., data = train_gbm, distribution = "multinomial",
                 n.trees = 20,shrinkage = 0.03, n.cores = 3,interaction.depth = 3,verbose = TRUE)
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
## Warning in if (nrow(x) != ifelse(class(y) == "Surv", nrow(y), length(y))) {: the
## condition has length > 1 and only the first element will be used
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.0300    0.0397
##      2        1.5862             nan     0.0300    0.0358
##      3        1.5651             nan     0.0300    0.0334
##      4        1.5453             nan     0.0300    0.0314
##      5        1.5268             nan     0.0300    0.0286
##      6        1.5098             nan     0.0300    0.0265
##      7        1.4935             nan     0.0300    0.0248
##      8        1.4782             nan     0.0300    0.0227
##      9        1.4644             nan     0.0300    0.0216
##     10        1.4514             nan     0.0300    0.0203
##     20        1.3555             nan     0.0300    0.0106
```

Now let's predict the validation set.

```
pred_gbm = predict(model_gbm,validation_gbm[,-c(length(validation_imputed)-1)],type = "response")
pred_gbm = matrix(unlist(pred_gbm), ncol = 5)

result = c()
pred_gbm[pred_gbm == 0] = 0.0000000000001
for (i in c(1:length(validation_imputed$health))){
result = c(result,log(pred_gbm[i,as.integer(validation_imputed$health[i])]))
}
-mean(result)
```

Stacking

Combing the validation prediction results from above three models as base learners (RandomForest, Gbm, and Neural Network)

```
train_stack = as.data.frame(cbind(pred_rf,pred_gbm,pred_nerual,validation_imputed$health))
colnames(train_stack) <- c('rf1','rf2','rf3','rf4','rf5',
                           'gbm1','gbm2','gbm3','gbm4','gbm5',
                           'n1','n2','n3','n4','n5','health')
```

Training the Multinomial logistic regression model for stacking

```
library(nnet)
multinom_model <- multinom(health~ ., data = train_stack)
```

re-training the base models using entire dataset (training and validation) and predicting test data ### Random Forest

```
pred_test_rf <- predict(rf_model_all,test_rf, type = 'prob',importance = TRUE)
```

Neural Network

```
validation.y = to_categorical(as.numeric(validation_imputed$health)-1)
model %>%
  fit(rbind(train_neural,validation_neural), rbind(train.y,validation.y),
      epochs =  5, batch_size = 32, validation_split = 0.2,
      callbacks = list(cp_callback))
model %>% load_model_weights_tf(filepath = checkpoint_path)
pred_test_nerual = model %>% predict(test_neural)
```

GBM

```
model_gbm_all <- gbm(health~., data = rbind(train_gbm,validation_gbm),
                 distribution = "multinomial",
                 n.trees = 200,shrinkage = 0.03, n.cores = 3,
                 interaction.depth = 3,verbose = TRUE)
pred_test_gbm = predict(model_gbm_all,test_gbm,type = "response")
pred_test_gbm = matrix(unlist(pred_test_gbm), ncol = 5)
```

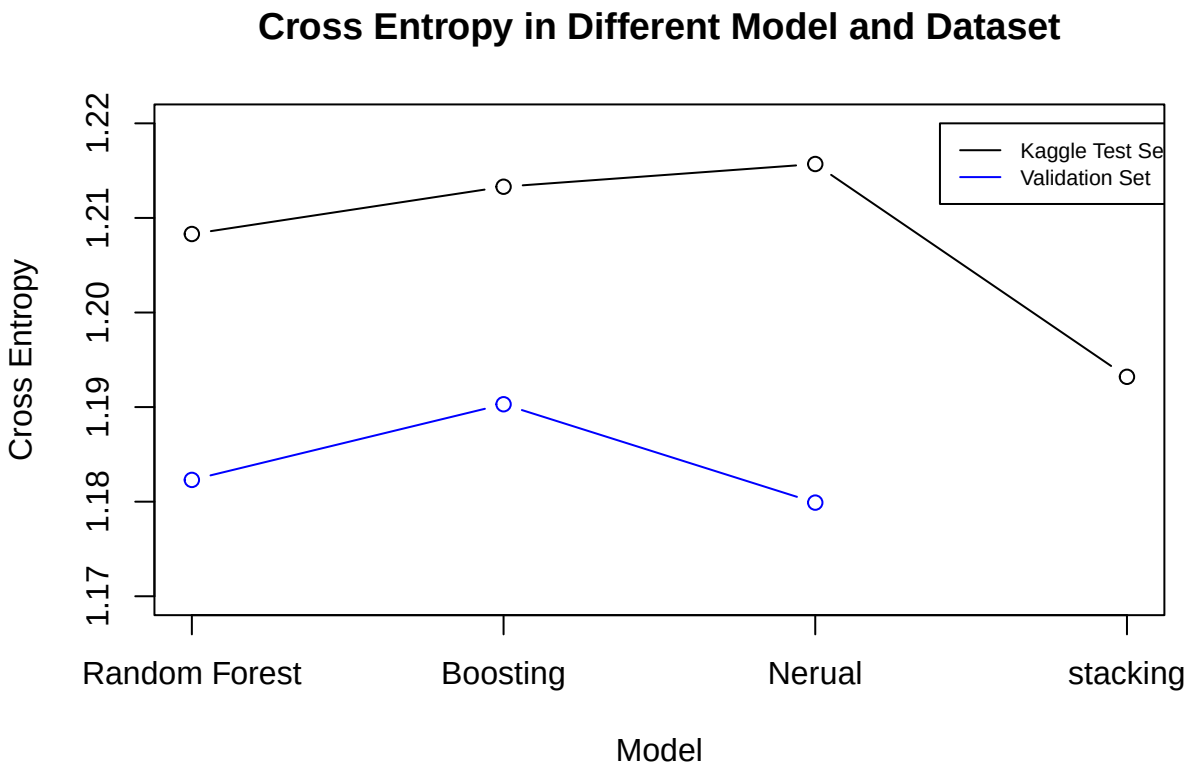
Combining the results and predicting the testing dataset using stacking model

```
test_stack = as.data.frame(cbind(pred_test_rf,pred_test_nerual,pred_test_gbm))
colnames(test_stack) <- c('rf1','rf2','rf3','rf4','rf5',
                          'gbm1','gbm2','gbm3','gbm4','gbm5',
                                          'n1','n2','n3','n4','n5')
pred = predict(multinom_model,test_stack,type = "probs")
```

Conclusion

The performance of different models in validation dataset and test set from the Kaggle submission

```
validation_cross = c(1.1823,1.1903,1.1799,NA)
kaggle_cross = c(1.2083,1.2133,1.2157,1.1932)
names = c('Random Forest','Boosting','Nerual','stacking')
plot(kaggle_cross, ylim = c(1.17,1.22),type = 'b',xaxt = "n",
     xlab = "Model",ylab = "Cross Entropy",
     main = "Cross Entropy in Different Model and Dataset")
axis(1, at=c(1,2,3,4), labels=names)
points(validation_cross, col = 'blue',type = 'b')
legend(3.4,1.22,legend=c("Kaggle Test Set", "Validation Set"),
       col=c("black","blue"), lty=1:1,cex = 0.7 )
```


Cross Entropy in Different Model and Dataset
Cross Entropy
1.17
1.18
1.19
1.20
1.21
1.22
Kaggle Test Se
Validation Set
Random Forest
Boosting
Nerual
stacking
Model