**1. Use the below given data set**

**DataSet:https://archive.ics.uci.edu/ml/datasets/BlogFeedback**

**Problem- prediction of the number of comments in the upcoming 24 hours on those blogs, The train data was generated from different base times that may temporally overlap. Therefore, if you simply split the train into disjoint partitions, the underlying time intervals may overlap. Therefore, the you should use the provided, temporally disjoint train and test splits to ensure that the evaluation is fair.**

**a. Read the dataset and identify the right features**

**b. Clean dataset, impute missing values and perform exploratory data analysis.**

**c. Visualize the dataset and make inferences from that**

**d. Perform any 3 hypothesis tests using columns of your choice, make conclusions**

**e. Create a linear regression model to predict the number of comments in the next 24 hours (relative to basetime)**

**f. Fine tune the model and represent important features**

**g. Interpret the summary of the linear model**

**h. Report the test accuracy vs. the training accuracy**

**i. Interpret the final model coefficients**

**j. Plot the model result and compare it with assumptions of the model**

library(data.table)

install.packages("foreach")

library(foreach)

library(readr)

setwd("D:\BlogFeedback")

blogData\_train <- read\_csv("D:\BlogFeedback\blogData\_train.csv")

View(blogData\_train)

#retrieve filenames of test sets

test\_filenames = list.files(pattern = "blogData\_test")

#load and combine dataset

train = fread("blogData\_train.csv")

test = foreach(i = 1:length(test\_filenames), .combine = rbind) %do% {

temp = fread(test\_filenames[i], header = F)

}

#log-transform

train[, V281 := log(1 + V281)]

test[, V281 := log(1 + V281)]

#drop continous variables without variation

drop = c(8, 13, 28, 33, 38, 40, 43, 50, 278)

train[, (drop) := NULL]

test[, (drop) := NULL]

#write to files

write.csv(train, "BlogFeedback-Train.csv", row.names = F)

write.csv(test, "BlogFeedback-Test.csv", row.names = F)

train

test

#Basic Models

library(data.table)

library(MatrixModels)

library(e1071)

install.packages("FNN")

library(FNN)

install.packages("glmnet")

library(glmnet)

install.packages("ranger")

library(ranger)

install.packages("xgboost")

library(xgboost)

#load and combine dataset

train = fread("BlogFeedback-Train.csv")

test = fread("BlogFeedback-Test.csv")

#error measure

mse = function(y\_hat, y) {

mse = mean((y - y\_hat)^2)

return(mse)

}

#create design matrices

train\_x = model.Matrix(V281 ~ . - 1, data = train, sparse = F)

train\_x\_sparse = model.Matrix(V281 ~ . - 1, data = train, sparse = T)

train\_y = train$V281

test\_x = model.Matrix(V281 ~ . - 1, data = test, sparse = F)

test\_y = test$V281

train\_xgb = xgb.DMatrix(data = as.matrix(train\_x), label = train\_y)

test\_xgb = xgb.DMatrix(data = as.matrix(test\_x), label = test\_y)

# kNN

pred\_knn = knn.reg(train\_x, test\_x, train\_y, k = 19)$pred

mse(pred\_knn, test\_y)

# LASSO

mdl\_lasso = cv.glmnet(train\_x\_sparse, train\_y, family = "gaussian", alpha = 1)

pred\_lasso = predict(mdl\_lasso, newx = test\_x)

mse(pred\_lasso, test\_y)

# random forest

mdl\_rf = ranger(V281 ~ ., data = train, num.trees = 1000, mtry = 120, write.forest = T)

pred\_rf = predict(mdl\_rf, test)

mse(pred\_rf$predictions, test\_y)

# SVM

mdl\_svm = svm(V281 ~ V52 + V55 + V61 + V51 + V54 + V21 + V6 + V10, data = train, kernel = "radial", cost = 2, gamma = 0.25)

pred\_svm = predict(mdl\_svm, test)

mse(pred\_svm, test\_y)

# XGboost

mdl\_xgb = xgboost(data = train\_xgb, nround = 750, nthread = 4, max\_depth = 6, eta = 0.025, subsample = 0.7, gamma = 3)

pred\_xgb = predict(mdl\_xgb, test\_xgb)

mse(pred\_xgb, test\_y)

#Stacked Generalization

library(data.table)

library(foreach)

library(MatrixModels)

library(e1071)

library(FNN)

library(glmnet)

library(ranger)

library(xgboost)

#load and combine dataset

train = fread("BlogFeedback-Train.csv")

test = fread("BlogFeedback-Test.csv")

#error measure

mse = function(y\_hat, y) {

mse = mean((y - y\_hat)^2)

return(mse)

}

#create design matrices

test\_x = model.Matrix(V281 ~ . - 1, data = test, sparse = F)

test\_x\_sparse = model.Matrix(V281 ~ . - 1, data = test, sparse = T)

train\_y = train$V281

test\_y = test$V281

#divide training set into k folds

k = 5

cv\_index = 1:nrow(train)

cv\_index\_split = split(cv\_index, cut(seq\_along(cv\_index), k, labels = FALSE))

#meta features from kNN

meta\_knn\_test = rep(0, nrow(test))

meta\_knn\_train = foreach(i = 1:k, .combine = c) %do% {

split the raining set into two disjoint sets

train\_index = setdiff(1:nrow(train), cv\_index\_split[[i]])

train\_set1 = model.Matrix(V281 ~ . - 1, data = train[train\_index], sparse = T)

train\_set2 = model.Matrix(V281 ~ . - 1, data = train[cv\_index\_split[[i]]], sparse = T)

level 0 prediction

meta\_pred = knn.reg(train\_set1, train\_set2, train[train\_index]$V281, k = 19)$pred

meta\_knn\_test = meta\_knn\_test + knn.reg(train\_set1, test\_x\_sparse, train[train\_index]$V281, k = 19)$pred / k

return(meta\_pred)

}

#meta features from LASSO

meta\_glm\_test = rep(0, nrow(test))

meta\_glm\_train = foreach(i = 1:k, .combine = c) %do% {

# split the raining set into two disjoint sets

train\_index = setdiff(1:nrow(train), cv\_index\_split[[i]])

train\_set1 = model.Matrix(V281 ~ . - 1, data = train[train\_index], sparse = T)

train\_set2 = model.Matrix(V281 ~ . - 1, data = train[cv\_index\_split[[i]]], sparse = T)

level 0 prediction

temp\_glm = cv.glmnet(train\_set1, train[train\_index]$V281, family = "gaussian", alpha = 1)

meta\_pred = predict(temp\_glm, newx = train\_set2)

meta\_glm\_test = meta\_glm\_test + predict(temp\_glm, newx = test\_x\_sparse) / k

return(meta\_pred)

}

#meta features from SVM

meta\_svm\_test = rep(0, nrow(test))

meta\_svm\_train = foreach(i = 1:k, .combine = c) %do% {

#split the raining set into two disjoint sets

train\_index = setdiff(1:nrow(train), cv\_index\_split[[i]])

train\_set1 = train[train\_index]

train\_set2 = train[cv\_index\_split[[i]]]

#level 0 prediction

temp\_svm = svm(V281 ~ V52 + V55 + V61 + V51 + V54 + V21 + V6 + V10, data = train\_set1,

kernel = "radial", cost = 2, gamma = 0.25)

meta\_pred = predict(temp\_svm, train\_set2)

meta\_svm\_test = meta\_svm\_test + predict(temp\_svm, test) / k

return(meta\_pred)

}

#meta features from random forest

meta\_rf\_test = rep(0, nrow(test))

meta\_rf\_train = foreach(i = 1:k, .combine = c) %do% {

#split the raining set into two disjoint sets

train\_index = setdiff(1:nrow(train), cv\_index\_split[[i]])

train\_set1 = train[train\_index]

train\_set2 = train[cv\_index\_split[[i]]]

#level 0 prediction

temp\_rf = ranger(V281 ~ ., data = train\_set1, num.trees = 500, mtry = 120, write.forest = T)

meta\_pred = predict(temp\_rf, train\_set2)$predictions

meta\_rf\_test = meta\_rf\_test + predict(temp\_rf, test)$predictions / k

return(meta\_pred)

}

#meta features from XGBoost

meta\_xgb\_test = rep(0, nrow(test))

meta\_xgb\_train = foreach(i = 1:k, .combine = c) %do% {

#split the raining set into two disjoint sets

train\_index = setdiff(1:nrow(train), cv\_index\_split[[i]])

train\_set1 = model.Matrix(V281 ~ . - 1, data = train[train\_index], sparse = F)

train\_set2 = model.Matrix(V281 ~ . - 1, data = train[cv\_index\_split[[i]]], sparse = F)

#xgb data

train\_set1\_xgb = xgb.DMatrix(data = as.matrix(train\_set1), label = train[train\_index]$V281)

train\_set2\_xgb = xgb.DMatrix(data = as.matrix(train\_set2), label = train[cv\_index\_split[[i]]]$V281)

#level 0 prediction

temp\_xgb = xgboost(data = train\_set1\_xgb, nround = 750, nthread = 4, max\_depth = 6, eta = 0.025, subsample = 0.7, gamma = 3)

meta\_pred = predict(temp\_xgb, train\_set2\_xgb)

meta\_xgb\_test = meta\_xgb\_test + predict(temp\_xgb, test\_xgb) / k

return(meta\_pred)

}

#combine meta features

sg\_col = c("meta\_knn", "meta\_glm", "meta\_svm", "meta\_rf", "meta\_xgb", "y")

train\_sg = data.frame(meta\_knn\_train, meta\_glm\_train, meta\_svm\_train, meta\_rf\_train, meta\_xgb\_train, train\_y)

test\_sg = data.frame(meta\_knn\_test, meta\_glm\_test, meta\_svm\_test, meta\_rf\_test, meta\_xgb\_test, test\_y)

colnames(train\_sg) = sg\_col

colnames(test\_sg) = sg\_col

#ensemble with elastic-net regression

train\_sg\_sparse = model.Matrix(y ~ . - 1, data = train\_sg, sparse = T)

test\_sg\_sparse = model.Matrix(y ~ . - 1, data = test\_sg, sparse = T)

mdl\_glm = cv.glmnet(train\_sg\_sparse, train\_y, family = "gaussian", alpha = 0.2)

pred\_glm = predict(mdl\_glm, newx = test\_sg\_sparse, s = "lambda.min")

mse(pred\_glm, test\_y)

[1] 0.3840147

library(data.table)

library(foreach)

library(MatrixModels)

library(xgboost)

library(ranger)

#load and combine dataset

train = fread("BlogFeedback-Train.csv")

test = fread("BlogFeedback-Test.csv")

#error measure

mse = function(y\_hat, y) { mse = mean((y - y\_hat)^2)

return(mse)}

#create design matrices

train\_x = model.Matrix(V281 ~ . - 1, data = train, sparse = F)

train\_y = train$V281

test\_x = model.Matrix(V281 ~ . - 1, data = test, sparse = F)

test\_y = test$V281

train\_xgb = xgb.DMatrix(data = as.matrix(train\_x), label = train\_y)

test\_xgb = xgb.DMatrix(data = as.matrix(test\_x), label = test\_y)

#number of models

n = 5

#fit XGBoost

pred\_xgb = foreach(i = 1:n, .combine = cbind) %do% { mdl\_xgb = xgboost(data = train\_xgb, nround = 750, nthread = 4, max\_depth = 6, eta = 0.025, subsample = 0.7, gamma = 3)

return(predict(mdl\_xgb, test\_xgb))}

#fit random forest

pred\_rf = foreach(i = 1:n, .combine = cbind) %do% { mdl\_rf = ranger(V281 ~ ., data = train, num.trees = 1000, mtry = 120, write.forest = T)

return(predict(mdl\_rf, test)$predictions)}

#weighted average

mse(rowMeans(pred\_rf) \* 0.25 + rowMeans(pred\_xgb) \* 0.75, test\_y)