**STAT 440**

**BIG DATA**

**MODULE 2 REPORT**

Earth-similarity of exoplanets

Anson Hsu

**301208466**

Daniel South

**301258075**

Youwei Yan

**301283162**

**Introduction**

This report was made to analyze variables concerning stars and planets with the interest of finding planets which might be similar to Earth. The response variable we were tasked to estimate is Earth similarity index or ESI. ESI ranges from 0 to 1 with 1 being Earth and 0 being completely unlike earth. The data set contains information primarily on the star nearest the planet in question and each planet has its own file on the flux measurement by time from the nearest star as said planet revolves around it. This presented a bit of a problem because we figured we would need more information on the planet itself to consider its similarity to earth. By using the light curves data we were able to create an estimate on planet radius which could be an important variable to the equation of determining ESI. Realizing this we created additional data such as planet radius from the lightcurve data to include and use in order to train a model for predicting ESI. From there we split our training set into our own train and test set in order to analyze our own models for accuracy before making submissions. After trying various models and comparing their performance, we decided to use an ensemble of Random Forest, GBM, and XGBoost model as our final model for prediction.

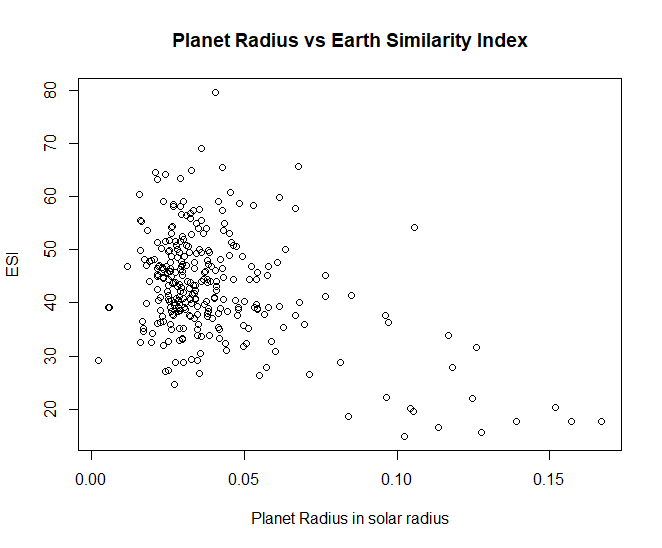
**Data Description**

There are 16 explanatory variables in this data set along with the response variable ESI. However, among these explanatory variables, only the variable ‘period’ describes a planet’s own property while other variables describe the properties of the star which the planet is orbiting around. Besides the training and testing data set, we were also provided with the lightcurve data. The lightcurve data contain the intensity of light emitted by different stars over 500 days. Since we wanted to gain more knowledge about the planets, we developed an algorithm to estimate the radius of the planets using the lightcurves.

**Lightcurve Analysis**

To make use of the lightcurves, we first found the lightcurves with star names appeared in both the lightcurve data and the combined training and testing data. We also kept track of the radius of the star and the orbital period of the planets which orbit around those stars. Then we developed a two-stage algorithm to estimate the radius of the planets.

Since we were given the orbital period of the planets, we expected periodic drops with the period equivalent to the orbital period in the light intensity, and the depth of the drops can be used to calculate the radius of the planets using the radius of the stars. However, due to reasons like small intensity drops, multiple planets orbiting around the same star, fluctuations in the actual period time, etc, it was difficult to detect these drops in many lightcurves. As a result, the first part of our algorithm detected the drops in the light curves which were more likely to be the true drops resulted by the planets, and then we could estimate the radius of these planets. The second part of the algorithm estimated the radius of the planets which could not be determined by the first part by slicing the lightcurves and finding the minimum value of the sliced pieces as the depth of the drops. The algorithm also output the status of how the radii were computed, i.e, which part of the algorithm, by assigning different levels to a categorical variable ‘status’ beside the computed radii. Figure 1 shows the relationship between the ESI and the radius of the planets estimated by the algorithm.



*(Figure 1)*

**Data Preprocess**

In both the training and testing data set, there were missing values appeared in 6 explanatory variables. In the beginning, we used KNN to impute the missing values but later we found that this imputation method was outperformed by simply taking the column median to replace the missing values. We added the radius of planets and their estimation status into the dataset as new variables, and we imputed the missing values in the planets’ radii with a bagged tree model. The missing values in the categorical variable ‘status’ were replaced with a new level. To generate variables that might be useful for predicting the ESI, we made transformations using few variables based on certain equations that might be used during the calculation of the ESI. For example, we took the cubic of the planets’ radii. We also transformed the categorical variable ‘planet’ into an integer variable because its initial levels were determined by the rank of the positions in the stellar system. In addition, we generated a new variable by dividing the transformed ‘planet’ by the variable ‘planets’ which indicates the total number of planets in the stellar systems.

**Model Construction**

We tried many different models using the preprocessed data to predict the ESI, and the tree-based models, which were Random Forest, GBM, and XGBoost, showed the best performance. To further improve the performance of these models, we tuned the hyperparameters of the models by first defining a search range in the hyperparameter space. Then, we performed 200 random searches in the space. For each search, we obtain a set of hyperparameters and we evaluated them by a 5-fold cross-validation using the pre-processed training data. After 200 searches, we used the set of hyperparameters which gave the best test mean MAE. Finally, we trained the models using the entire training set and used the model to predict the ESI of the testing data. We estimated our prediction performance by repeating a 3-fold cross-validation 10 times. We also built an ensemble of the three models, but we found its estimated performance could be occasionally surpassed by the single XGBoost model. We suspected that this was caused by the random search mechanism we used to find the hyperparameters. In the end, we decided to use the ensemble model to make predictions due to its more stable performance.

**Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table1: Model Performance** | | | |
|  | **Estimated**  **Performance** | **Public Leaderboard** | **Private Leaderboard** |
| **Random Forest** | 5.43 | 6.26 | 5.34 |
| **GBM** | 5.50 | 6.34 | 5.36 |
| **XGBoost** | 5.38 | 6.19 | 5.29 |
| **Ensemble** | 5.38 | 6.18 | 5.27 |

Table 1 shows the performance of the models we constructed in terms of MAE. We noticed that there was a relatively large gap between our estimated MAE and the MAE on the public leaderboard. We speculated that the gap was a result of the dissimilar distribution of the data used by the public leaderboard since it only used 30% of the testing data to calculate the MAE. The performance of the data on the rest of the 70% testing data was consistent with our estimation, the ensemble model usually had the best and most stable performance. In addition, by comparing the submission results, we also noticed that using explanatory variables generated from the lighcurve data had led to an approximately 0.4 reduction in MAE on the private leaderboard.

**Conclusion**

In conclusion, the ensemble model obtained an MAE of 5.27 on the public leaderboard and gave us the first rank in this module. Through this module, we learned that performing cross-validation can provide a good criterion for evaluating model performance and prevent overfitting. Also, the success in using the lightcurve data taught us that gaining useful information and using it for model training can significantly improve the model performance.

CODE A

library(randomForest)

library(TSA)

library(dplyr)

rm(list=ls())

set.seed(123)

setwd("D:/sfu/stat 440/module2")

s1 = read.table("train-2.txt",sep=" ",header = T)

s2 = read.table("test-2.txt",sep=" ",header = T)

sav = s1[,c("star","period","radius")]

sav2 = s2[,c("star","period","radius")]

# lightcurve dir

setwd("D:/sfu/stat 440/module2/lightcurves")

write.csv(sav,"sav.csv",row.names = F, col.names = T)

write.csv(sav2,"sav2.csv",row.names = F, col.names = T)

filenames = list.files(pattern="\*.txt", full.names=FALSE)

colname = sub(".txt","",filenames)

filenames = as.data.frame(colname)

colnames(filenames)[1] = "star"

sav = read.csv('sav.csv',header=T)

sav2 = read.csv('sav2.csv',header = T)

sav = rbind(sav,sav2)

# extract useful variables for calculation

mat = inner\_join(sav,filenames,by = "star")

radius = data.frame(matrix(nrow=nrow(mat),ncol=5))

radius[,1] = mat$star

radius[,2:3] = c(mat$period,mat$radius)

colnames(radius) = c("star","period","s\_radius","status","p\_radius")

radius = radius[!duplicated(radius),]

# first part of algorithm

detection = function(t,fl,p\_day,rad, s\_radius, verbose = 0) {

fl = na.roughfix(fl)

days = max(t)

day\_per\_int = days/length(t)

nint = length(t)

# number of intervals in a orbital period

p\_int = round(p\_day/day\_per\_int)

p\_int\_00 = floor(p\_int/100)

# find the indices of flux which can be used to calculate planet radius

if(p\_day>5){

N = floor(days/p\_day)\*10

# indices of lowest flux

index <- order(fl)[1:N]

index = unique(floor(index/100))

index = sort(index)

} else {

status = "D"

p\_radius = NA

return(list(status,p\_radius))

}

len = length(index)

status = "S"

p\_radius = NA

for (i in 1:(len-1)) {

# stop when p\_radius is determined

if(status == "D"){

break

}

# stop when no pattern detected in the first 25% data

if ((i/len)>0.25){

if (verbose==1){

print(status)

}

break

}

j = i

if ((index[j+1]-index[j])==1){

j = j+1

}

curr = index[j]

n = 0

depth = 0

# check if next index is in the range of current index +- orbital period

while((TRUE%in%(index[(j+1):len]%in%(seq(curr+p\_int\_00-2,curr+p\_int\_00+2))))&((j+1)<len)){

n = n+1

if(index[j]==0) {

# find the min around that index when index is 0

depth = depth + min(fl[seq(1,99,by=1)])

}else{

# find the min around that index

depth = depth + min(fl[seq(index[j]\*100-20,index[j]\*100+99,by=1)])

}

# move current index to the next index satisfied the condition

j=j+min(which(index[(j+1):len]%in%(seq(curr+p\_int\_00-2,curr+p\_int\_00+2))==TRUE))

curr = index[j]

# if next index is index+1 make the current index to the next index (might be problematic)

if (((j<len)&(index[j+1]-index[j])==1)){

j = j+1

curr = index[j]

}

# print(j)

}

# if the pattern is found in 40% of the times it should theoretically appear

if(n >= (floor(0.4\*(days/p\_day)))) {

if(verbose == 1){

print("Detected")

}

status = "D"

# calculate the planet's radius using the average max depth found

max\_depth = (1-depth/n)

p\_radius = s\_radius\*sqrt(max\_depth)

} else{

status = "N"

}

}

return(list(status,p\_radius))

}

for(i in 1:nrow(radius)){

if(!is.na(radius[i,2])){

f = read.table(paste0(radius[i,"star"],".txt"),sep = " ",header = T,row.names = NULL)

radius[i,"status"] = detection(t = f$time, fl = f$flux, p\_day = radius[i,"period"], s\_radius = radius[i,"s\_radius"],verbose = 0)[[1]]

radius[i,"p\_radius"] = detection(t = f$time, fl = f$flux, p\_day = radius[i,"period"], s\_radius = radius[i,"s\_radius"])[[2]]

}

print(paste("/////////////////////",i,"//////////////////////"))

}

sum(!is.na(radius$p\_radius))

# second part of algorithm

# slice the light curve data with regard to the orbital period

# and compute the radius with the min of the sliced lightcurves

sup = function(t , fl, p\_day , s\_radius){

fl = na.roughfix(fl)

days = max(t)

day\_per\_int = days/length(t)

nint = length(t)

p\_int = round(p\_day/day\_per\_int)

p\_int\_00 = floor(p\_int/100)

n = floor(nint/(p\_int\*2))

Min = 0

if(p\_day>280) {

# too lazy to fix the outofbound error

# so just add a condition that computes na

n = 1

}

for(i in 1:n) {

Min = Min+min(fl[(1+(i-1)\*p\_int\*2):(i\*p\_int\*2)])

if(p\_day>280) {

print(paste0("range: ",(1+(i-1)\*p\_int\*2),"/",i\*p\_int\*2))

print(Min)

}

}

Min = Min/n

max\_depth = (1-Min)

p\_radius = s\_radius\*sqrt(max\_depth)

print(p\_radius)

if(p\_day>280) {

p\_radius = s\_radius\*sqrt(1-min(fl))

}

return(p\_radius)

}

# compute the radius for which the prior for loop outputs NA

for(i in 1:nrow(radius)){

if((!is.na(radius[i,"period"]))&(is.na(radius[i,"p\_radius"]))){

f = read.table(paste0(radius[i,"star"],".txt"),sep = " ",header = T,row.names = NULL)

radius[i,"p\_radius"] = sup(t = f$time, fl = f$flux, p\_day = radius[i,"period"],s\_radius = radius[i,"s\_radius"])

}

print(paste("/////////////////////",i,"//////////////////////"))

}

radius = subset(radius, select = -s\_radius)

setwd("D:/sfu/stat 440/module2")

write.csv(radius,"rad2.csv",row.names = F, col.names = T)

CODE B

library(randomForest)

library(xgboost)

library(mlr)

library(dplyr)

### run code\_A.R before running this script ###

rm(list=ls())

set.seed(123)

setwd("D:/sfu/stat 440/module2")

samples = read.table("train-2.txt", sep=' ', row.names = NULL, header = TRUE)

co = cor(sample[,is.numeric(samples)], method = "spearman")

corrplot::corrplot(co, type = "upper")

index = sample(nrow(samples),floor(0.75\*nrow(samples)))

train = samples[index,]

test = samples[-index,]

data\_processing = function(train,test,option){

if("ESI"%in%colnames(test)) {

test\_y = test[,"ESI"]

test = test[,!colnames(test)%in%"ESI"]

}

if("ESI"%in%colnames(train)) {

train\_y = train[,"ESI"]

train = train[,!colnames(train)%in%"ESI"]

}

tot = rbind(train,test)

tot$temp = as.numeric(tot$temp)

tot$planets = as.numeric(tot$planets)

tot\_planet = subset(tot, select = planet)

tot = subset(tot, select = -planet)

tot\_star = subset(tot, select = star)

tot = subset(tot, select = -star)

n\_train = nrow(train)

n\_test = nrow(test)

n\_tot = n\_train+n\_test

# imputation

tot = na.roughfix(tot)

tot$planets = as.integer(tot$planets)

tot = cbind(tot\_star,tot)

# add p\_radius and status variable

rad = read.csv("rad2.csv",header = T)

tot = left\_join(tot,rad, by = c("star","period"))

tot = subset(tot,select = -star)

tot$status = addNA(tot$status)

levels(tot$status)[is.na(levels(tot$status))] = "A"

# impute missings in p\_radius

prepro = caret::preProcess(tot,method = "bagImpute")

tot = predict(prepro,tot)

# add possibly useful features

tot$tf = sqrt(((0.009158-tot$p\_radius)/(0.009158+tot$p\_radius))^2)

tot$es = tot$p\_radius^3

tot$sh = tot$period/tot$p\_radius

tot = createDummyFeatures(tot)

# recreat planet and add planet/planets

tot$planet = recode(tot\_planet$planet,"b" = 1,'c' = 2,'d'=3,'e'=4,'f'=5,'g'=6,'h'=7)

tot$pp = tot$planet/tot$planets

# fix rows with planet>planets

rrow = which(tot$pp>1)

tot[rrow,"planets"] = tot[rrow,"planet"]

tot[rrow,"pp"] = 1

train = cbind(tot[1:n\_train,],"ESI" = train\_y)

if(option == 0){

test = cbind(tot[(n\_train+1):n\_tot, ], "ESI" = test\_y)

tot = rbind(train,test)

return(tot)

} else if(option ==1) {

test = tot[(n\_train+1):n\_tot, ]

return(list(train,test))

}

}

tot = data\_processing(train = train, test = test,option = 0)

# regression and data

tsk = makeRegrTask(data = tot, target = "ESI")

# split data into train and test

h = makeResampleDesc("Holdout")

ho = makeResampleInstance(h,tsk)

tsk.train = subsetTask(tsk,ho$train.inds[[1]])

tsk.test = subsetTask(tsk,ho$test.inds[[1]])

# use all cpus during training

library(parallel)

library(parallelMap)

parallelStartSocket(cpus = detectCores())

# number of iterations used for hyperparameters tuning

tc = makeTuneControlRandom(maxit = 200)

# resampling strategy for evaluating model performance

rdesc = makeResampleDesc("RepCV", reps = 10, folds = 3)

#------------------ randomForest ------------------

# build model

rf\_lrn = makeLearner(cl ="regr.randomForest", par.vals = list())

# define the search range of hyperparameters

rf\_ps = makeParamSet( makeIntegerParam("ntree",150,600),makeIntegerParam("nodesize",lower = 3,upper = 15),

makeIntegerParam("se.ntree",lower = 50 ,upper = 300), makeIntegerParam("se.boot",lower = 50,upper = 300),

makeIntegerParam("mtry",lower = 2,upper = 18),makeLogicalParam("importance",default = FALSE))

# search for the best hyperparameters

rf\_tr = tuneParams(rf\_lrn,tsk.train,cv5,mae,rf\_ps,tc)

# specify the hyperparmeters for the model

rf\_lrn = setHyperPars(rf\_lrn,par.vals = rf\_tr$x)

detach(package:caret)

# evaluate performance use CV

r = resample(rf\_lrn, tsk, resampling = rdesc, show.info = T, models = FALSE,measures = mae)

#-------------------------------------

#------------------ gbm ------------------

gbm\_lrn = makeLearner(cl = "regr.gbm", par.vals = list())

gbm\_ps = makeParamSet( makeNumericParam("shrinkage",lower = 0.0001, upper= 0.01),makeNumericParam("bag.fraction",lower = 0.5,upper = 1),

makeIntegerParam("n.trees",lower = 50,upper = 500), makeIntegerParam("interaction.depth",lower = 1,upper = 10),

makeIntegerParam("n.minobsinnode",lower = 5,upper = 30))

gbm\_tr = tuneParams(gbm\_lrn,tsk.train,cv5,mae,gbm\_ps,tc)

gbm\_lrn = setHyperPars(gbm\_lrn,par.vals = gbm\_tr$x)

gbm\_mod = train(gbm\_lrn, tsk.train)

gbm\_pred = predict(gbm\_mod, tsk.test)

performance(gbm\_pred, measures = mae)

r = resample(gbm\_lrn, tsk, resampling = rdesc, show.info = T, models = FALSE,measures = mae)

#------------------------------------

#------------------ xgboost ------------------

xgb\_train = as.matrix(tot[1:nrow(train),])

xgb\_test = as.matrix(tot[(nrow(train)+1):nrow(samples),])

dtrain = xgb.DMatrix(data = subset(xgb\_train,select = -ESI),label = subset(xgb\_train,select = ESI))

dtest = xgb.DMatrix(data = subset(xgb\_test,select = -ESI),label= subset(xgb\_test,select = ESI))

params <- list(booster = "gbtree",

objective = "reg:linear", eta=0.007, gamma=0, max\_depth=5, min\_child\_weight=1, subsample=1, colsample\_bytree=1,nthread = 8)

xgbcv <- xgb.cv( params = params, data =dtrain, nrounds = 800, nfold = 5, showsd = T, stratified = T,

print\_every\_n = 10, early\_stop\_round = 20, maximize = F,metrics = 'mae')

# tuning

xgb\_lrn = makeLearner(cl = "regr.xgboost",predict.type = "response")

xgb\_lrn$par.vals = list(objective="reg:linear", eval\_metric="error", nrounds=800, eta=0.007,verbose=0)

xgb\_ps = makeParamSet( makeIntegerParam("max\_depth",lower = 7,upper = 14),

makeNumericParam("min\_child\_weight",lower = 1,upper = 9), makeNumericParam("subsample",lower = 0.5,upper = 1),

makeNumericParam("colsample\_bytree",lower = 0.5,upper = 1))

xgb\_tr = tuneParams(xgb\_lrn,tsk.train,cv5,mae,xgb\_ps,tc)

xgb\_lrn = setHyperPars(xgb\_lrn,par.vals = xgb\_tr$x)

xgb\_mod = train(xgb\_lrn, tsk.train)

xgb\_pred = predict(xgb\_mod, tsk.test)

performance(xgb\_pred, measures = mae)

r = resample(xgb\_lrn, tsk, resampling = rdesc, show.info = T, models = FALSE,measures = mae)

#------------------------------------------------

#------------------ ensemble --------------------

m = makeStackedLearner(base.learners = list(rf\_lrn,xgb\_lrn,gbm\_lrn),

predict.type = "response", method = 'hill.climb')

#------------------------------------------------

#------------------ submimssion -----------------

sub = read.table("test-2.txt", sep=' ', row.names = NULL, header = TRUE)

sub = subset(sub,select = -Id)

s = samples

sub = data\_processing(train=s,test = sub,option = 1)

sub = sub[[2]]

make\_prediction = function(lrn,tsk,sub\_data,subname) {

mod = train(lrn,tsk)

pred = predict(mod,newdata = sub\_data)

rdesc = makeResampleDesc("RepCV", reps = 10, folds = 3)

r = resample(lrn, tsk, resampling = rdesc, show.info = T, models = FALSE,measures = mae)

submission = read.csv("sample-2.csv",header = T)

submission$ESI = pred$data$response

write.csv(submission,file = subname,row.names = F, col.names = T)

}

make\_prediction(lrn = rf\_lrn,tsk = tsk,sub\_data = sub,subname = "rf\_-planet.csv")

make\_prediction(lrn = xgb\_lrn,tsk = tsk,sub\_data = sub,subname = "xgb\_-planet.csv")

make\_prediction(lrn = gbm\_lrn,tsk = tsk,sub\_data = sub,subname = "gbm\_-planet.csv")

make\_prediction(m,tsk = tsk,sub\_data = sub,subname = "ens\_-planet.csv")