### Scripts for data analysis

###

### "Data science approach to PISA"

### Florence Gabriel, Jason Signolet, Martin Westwell

###

###

### External data files:

### PISA2012\_StdQ\_AUS.dat

### StdQ\_dictionary.dat

library(data.table)

library(magrittr)

library(foreach)

library(doParallel)

library(parallelMap)

library(mlr)

library(xgboost)

library(gbm)

library(dismo)

library(ggplot2)

############################

### Custom functions

############################

all\_seated <- function(DT) {

# Flag TRUE for each row where students sat all parts of the test

# Accepts a data.table of PISA student questionnaire data.

# Returns a vector of row indices excluding rows where the item response

# code was 7 ("N/A", i.e. the student wasn't set this question).

apply(DT, 1, function(a) !7 %in% a)

}

missingPISA <- function(DT, dictionary){

# Use the Pisa2012 dictionary to set missing values to NA.

# Accepts a data.table of PISA student questionnaire data (DT) and

# a data.table of a dictionary mapping item response codes to values.

# Returns the input data.table with missing responses converted to NA.

require(data.table)

variables\_input <- colnames(DT)

variables\_dict <- unique(dictionary$variable)

variables <- intersect(variables\_input, variables\_dict)

for(i in variables){

tryCatch(

{

missing <-

dictionary[variable == (i) & Label %in% c("N/A", "Missing", "Invalid"), value]

},

error = function(e)

{

return(missing <- NA)

}

)

if(is.na(missing[1])){

next

}

DT[eval(as.name(i)) %in% as.numeric(missing), eval(as.name(i)) := NA]

}

return(DT)

}

### Collarborative filtering functions

vec2matrix <- function(initial\_values, Y, MISSING, nq, ns, nf){

# optim() uses initial\_values as a vector,

# but we need them as matrices

# Accepts the current X and THETA values as a concatenated vector.

# Accepts table of data (Y)

# Accepts a vector of missing values (MISSING)

# Accepts the length of X (nq) and THETA (ns), and their widths (nf)

# as vectors.

# Returns X, THETA and DELTA as matrices

X\_length <- nq \* nf

X <- matrix(initial\_values[1:X\_length], nrow = nq)

THETA <- matrix(initial\_values[(X\_length + 1):length(initial\_values)], nrow = ns)

DELTA <- THETA %\*% t(X) - Y; DELTA[MISSING] <- 0; DELTA <- as.matrix(DELTA)

return(list(X, THETA, DELTA))

}

cost <- function(initial\_values, Y, MISSING, nq, ns, nf, lambda){

# Accepts the current X and THETA values as a concatenated vector.

# Accepts table of data (Y)

# Accepts a vector of missing values (MISSING)

# Accepts the length of X (nq) and THETA (ns), and their widths (nf)

# as vectors.

# Returns the cost as a number: i.e. sum of squared error plus sum of the coefficients

matrices <- vec2matrix(initial\_values, Y, MISSING, nq, ns, nf)

X <- matrices[[1]]

THETA <- matrices[[2]]

DELTA <- matrices[[3]]

J <- 0.5 \* (sum(DELTA^2, na.rm = T) + lambda \* (sum(X^2) + sum(THETA^2)))

}

gradient <- function(initial\_values, Y, MISSING, nq, ns, nf, lambda, alpha){

# Gradient of the cost function to perform gradient descent

# Accepts the current X and THETA values as a concatenated vector.

# Accepts table of data (Y)

# Accepts a vector of missing values (MISSING)

# Accepts the length of X (nq) and THETA (ns), and their widths (nf)

# as vectors.

# Accepts the regularisation constant (lambda) and the learning rate (alpha)

# Returns vector of X and THETA gradients

matrices <- vec2matrix(initial\_values, Y, MISSING, nq, ns, nf)

X <- matrices[[1]]

THETA <- matrices[[2]]

DELTA <- matrices[[3]]

X\_grad <- (t(DELTA) %\*% THETA) + lambda \* X

THETA\_grad <- (DELTA %\*% X) + lambda \* THETA

return(c(X\_grad, THETA\_grad))

}

col\_filt\_cv <- function(Y, MISSING){

# Used for collaborative filtering 5-fold CV.

# Accepts the table of data (Y) and a vector of missing values

# Splits the data in 5 random folds (excluding the missing values)

# and returns a list of 5 pairs of training and validation sets

val\_set <- matrix(c(sample(which(MISSING == F))), ncol = 5)

colnames(val\_set) <- paste("Set", 1:5, sep = ".")

Y\_train <- list(Y,Y,Y,Y,Y)

Y\_test <- list(Y,Y,Y,Y,Y)

for(i in 1:5){

Y\_train[[i]][val\_set[,i]] <- NA

Y\_test[[i]][-val\_set[,i]] <- NA

}

return(list(Y\_train, Y\_test))

}

############################

### Load and clean data

############################

### We are only interested in the disposition, demographic, and mathematical literacy scores

identifiers <- c("STRATUM", "SCHOOLID", "StIDStd",

"ST03Q01", "ST03Q02" # birth month, birth year

)

demographics <- c("ST04Q01", "ESCS", "ausSTATE", "ausGEOLOC\_3", "ausINDIG")

# dispositions

int\_motivation <- c("ST29Q01", "ST29Q03", "ST29Q04", "ST29Q06") # intrinsic motivation INTMAT

ext\_motivation <- c("ST29Q02", "ST29Q05", "ST29Q07", "ST29Q08") # extrinsic motivation INSTMOT

self\_concept <- c("ST42Q02", "ST42Q04", "ST42Q06", "ST42Q07", "ST42Q09") # SCMAT

self\_efficacy <- c("ST37Q01", "ST37Q02", "ST37Q03", "ST37Q04", "ST37Q05", "ST37Q06", "ST37Q07", "ST37Q08") # MATHEFF

control\_in\_school <- c("ST91Q01", "ST91Q02", "ST91Q03", "ST91Q04", "ST91Q05", "ST91Q06")

control\_in\_maths <- c("ST43Q01", "ST43Q02", "ST43Q03", "ST43Q04", "ST43Q05", "ST43Q06")

attr\_failure <- c("ST44Q01", "ST44Q03", "ST44Q04", "ST44Q05", "ST44Q07", "ST44Q08") # FAILMAT

maths\_anxiety <- c("ST42Q01", "ST42Q03", "ST42Q05", "ST42Q08", "ST42Q10") # ANXMAT

subj\_norms <- c("ST35Q01", "ST35Q02", "ST35Q03", "ST35Q04", "ST35Q05", "ST35Q06") # SUBNORM

dispositions <- c(int\_motivation, ext\_motivation, self\_concept,

self\_efficacy, control\_in\_school, control\_in\_maths, attr\_failure,

maths\_anxiety, subj\_norms)

disposition\_indices <- c("MATHEFF", "SCMAT", "ANXMAT", "INTMAT", "INSTMOT", "FAILMAT", "SUBNORM")

# mathematical literacy score plausible values

maths\_literacy <- paste0("PV", 1:5, "MATH")

# Import PISA 2012 student questionnaire data with raw item responses

stdq <- fread("Data/Raw/PISA2012\_StdQ\_AUS.dat",

select = c(identifiers, demographics, dispositions, disposition\_indices, maths\_literacy)

)

# Keep only rows where students were set the entire questionnaire

# and set missing values to NA

stdq\_dict <- fread("Data/Raw/StdQ\_dictionary.dat")

all\_seated\_test <-

all\_seated(stdq[, dispositions, with = F]) # reduces total rows from 14481 to 4752

stdq <- missingPISA(stdq[all\_seated\_test], stdq\_dict)

# Rename unclear identifier cols

setnames(stdq, c("ST03Q01", "ST03Q02"), c("BIRTHMONTH", "BIRTHYEAR"))

# Rename states

states <- c("ACT", "VIC", "NSW", "QLD", "SA", "WA", "TAS", "NT")

for(i in 1:8){

stdq[ausSTATE == i, STATE := states[i]]

}

# Rename geolocation

geolocs <- c("Urban", "Provincial", "Rural")

for(i in 1:3){

stdq[ausGEOLOC\_3 == i, GEOLOC := geolocs[i]]

}

# Rename gender

mfs <- c("Female", "Male")

for(i in 1:2){

stdq[ST04Q01 == i, GENDER := mfs[i]]

}

# Rename indig

indig <- c("Indigenous", "Non-indigenous")

stdq$INDIG <- "Indigenous"

stdq[ausINDIG == 0, INDIG := "Non-indigenous"]

# Make demographics columns into factors

for (col in c("GENDER", "STATE", "GEOLOC", "INDIG"))

set(stdq, j=col, value=as.factor(stdq[[col]]))

# Clear unused cols

stdq[, `:=`(ausSTATE = NULL,

ausGEOLOC\_3 = NULL,

ST04Q01 = NULL,

ausINDIG = NULL)]

### Model 1

stdq1 <- stdq[, c(identifiers, demographics, dispositions, maths\_literacy), with = F]

### For Model 1, the missing/invalid

### item responses need to be imputed

# Impute missing disposition data using collaborative filtering

# Centre data on 0 for training

Y <- as.matrix(stdq1[, dispositions, with = F] - 2.5)

# Define missing values

MISSING <- is.na(Y)

# Define feature matrix dimensions

ns <- nrow(Y)

nq <- ncol(Y)

# Define folds for cross-validation

Y\_5foldCV <- col\_filt\_cv(Y, MISSING)

# Tune number of hidden features (nf) across a grid from 1:7

registerDoParallel(cores = 8)

J\_counter <-

foreach(nf = 1:7, .combine = rbind) %:%

foreach(fold = 1:5, .combine = rbind, .multicombine = T, .packages = "data.table") %dopar%{

results <- optim(par = runif(nq\*nf + ns\*nf, -1, 1),

fn = cost,

gr = gradient,

Y = Y\_5foldCV[[1]][[fold]],

MISSING = is.na(Y\_5foldCV[[1]][[fold]]),

nq = nq, ns = ns, nf = nf, lambda = 1, #alpha = 0.001,

method = "L-BFGS-B",

control = list(trace = 1,

maxit = 1000))

test\_error <- cost(results$par, Y\_5foldCV[[2]][[fold]], MISSING = is.na(Y\_5foldCV[[2]][[fold]]),

nq, ns, nf, lambda = 1)

output <- data.table(nf = nf, fold = fold, Jtrain = results$value, Jtest = test\_error)

}

registerDoSEQ()

# Summarise the nf tuning

J\_summary <-

J\_counter[, .(

mean(Jtrain / (sum(!MISSING) \* 0.8)), # Jtrain is the sum total cost; we want average cost (mse)

sd(Jtrain / (sum(!MISSING) \* 0.8)) / sqrt(5),

mean(Jtest / (sum(!MISSING) \* 0.2)),

sd(Jtest / (sum(!MISSING) \* 0.2)) / sqrt(5)

), by = nf]

# Plot for Figure 1

p1 <- ggplot(J\_summary, aes(nf)) +

geom\_line(aes(y = V1), colour = "red") +

geom\_line(aes(y = V3), colour = "blue") +

geom\_errorbar(aes(ymin = V1 - V2 /2, ymax = V1 + V2 /2)) +

geom\_errorbar(aes(ymin = V3 - V4 /2, ymax = V3 + V4 /2))

# Optimum nf is 6

nf <- 6

# Learn the best coefficient values

# (cost and gradient functions defined above)

results <- optim(par = runif(nq\*nf + ns\*nf, -1, 1),

fn = cost,

gr = gradient,

Y = Y,

MISSING = MISSING,

nq = nq,

ns = ns,

nf = nf,

lambda = 1,

method = "L-BFGS-B",

control = list(trace = 1,

maxit = 1000)

)

output <- vec2matrix(results$par, Y, MISSING, nq, ns, nf)

# Estimate the missing values

X <- output[[1]]

THETA <- output[[2]]

GUESS <- THETA %\*% t(X)

Y[MISSING] <- GUESS[MISSING]

# Put the values back on the right scale

Y <- round(Y + 2.5)

# Replace raw disposition values with imputed values

stdq1 <- cbind(stdq1[, !names(stdq1) %in% dispositions, with = F], Y)

# Correct values that are out of range

for(col in dispositions) {

set(stdq1, i = which(d[[col]] < 1), j = col, value = 1)

set(stdq1, i = which(d[[col]] > 4), j = col, value = 4)

}

### Train xgboost model

# Set controllers for hyperparameter tuning

set.seed(20160522)

test\_set <- sort(sample(nrow(stdq1), 400))

train\_set <- seq(1,nrow(stdq1))[-test\_set]

# Define input and output data

task <- makeRegrTask(

id = "pisa",

data = as.data.frame(stdq1[train\_set, c(demographics, dispositions, maths\_literacy[1]), with = F]),

target = "PV1MATH"

)

# Define model and fixed hyperparameters

lrn <- makeLearner(

"regr.xgboost",

par.vals = list(

nrounds = 2000,

subsample = 0.5

)

)

# Define hyperparameter values to test

ps <- makeParamSet(

makeDiscreteParam("eta", values = c(0.005, 0.01, 0.02)),

makeDiscreteParam("max\_depth", values = c(1, 2, 5, 10, 20))

)

# Tell the optimiser to do a grid search

ctrl <- makeTuneControlGrid()

# Tell the optimiser to use 5-fold cross-validation

cv5f <- makeResampleDesc("CV", iters = 5)

# Perform tuning in parallel

parallelStartSocket(5)

results <- tuneParams(

lrn,

task = task,

resampling = cv5f,

par.set = ps,

control = ctrl,

measures = mae

)

parallelStop()

# Look at tuning results

opt <- as.data.table(results$opt.path)

# Plot Figure 3a

p3a <-

ggplot(opt, aes(

x = reorder(interaction.depth, as.numeric(as.character(interaction.depth))),

y = mae.test.mean,

group = as.numeric(as.character(shrinkage)),

linetype = reorder(shrinkage, as.numeric(as.character(shrinkage))),

shape = reorder(shrinkage, as.numeric(as.character(shrinkage)))

)) +

geom\_line() +

geom\_point() +

ylab("Cross-validation error") +

xlab("Tree depth") +

scale\_shape\_discrete(name = "Learning rate") +

scale\_linetype\_discrete(name = "Learning rate")

# Fit the model to the whole training set

# Optimised with eta = 0.005, max\_depth = 4

# Define the hyperparameter set

lrn\_model1 <- makeLearner(

"regr.xgboost",

par.vals = list(

nrounds = 2000,

print.every.n = 800,

subsample = 0.5,

eta = 0.005,

max\_depth = 4

)

)

# Do the fitting

fit\_model1 <- train(lrn\_model1, task = task, subset = train\_set)

# Make predictions from the test set

pred\_model1 <- as.data.table(predict(fit\_model1, task = task, subset = test\_set))

### Tree outputs

# Figure 4a

p4a <-

ggplot(pred\_model1, aes(truth, response)) +

geom\_point() +

geom\_abline(linetype = "dashed") +

coord\_cartesian(xlim = c(200, 800), ylim = c(200, 800))

# Error distribution of predictions on the hold-out set

error\_table[, pred\_model1[, quantile(abs(truth-response), c(.5, .9, .99))] ]

# Calculate feature importance

imp\_model1 <- xgb.importance(feature\_names = fit\_model1$features, model = fit\_model1$learner.model)

# Figure 5

p5 <- xgb.plot.importance(imp\_model1)

# Calculate partial dependency plots

pd\_model1 <- generatePartialPredictionData(fit\_model1, task, imp\_model1$Feature)

# Figure 6

p6 <- plotPartialPrediction(pd\_model1)

#### Model 2

stdq2 <- stdq[, c(identifiers, demographics, disposition\_indices, maths\_literacy), with = F]

# Only keep rows where all of the disposition indices are available

# Missing values are marked as 9999.0

for(col in names(stdq2)) {

# Returns in 4528 rows

stdq2 <- stdq2[eval(parse(text = col)) != 9999.0]

}

# Use the same students from training set 1 in set 2

students\_in\_testset1 <- stdq2[StIDStd %in% stdq1[test\_set, StIDStd], StIDStd]

test\_set\_2 <- which(stdq2$StIDStd %in% students\_in\_testset1)

train\_set\_2 <- seq(1,nrow(stdq2))[-test\_set\_2]

# Set controllers for hyperparameter tuning

task2 <- makeRegrTask(

data = stdq2[train\_set\_2, c(demographics, disposition\_indices, "PV1MATH"), with = F],

target = "PV1MATH"

)

lrn2 <- makeLearner(

"regr.gbm",

par.vals = list(

n.trees = 1500,

bag.fraction = 0.5

)

)

ps2 <- makeParamSet(

makeDiscreteParam("shrinkage", values = c(0.005, 0.01, 0.02)),

makeDiscreteParam("interaction.depth", values = c(1, 2, 5, 10, 20))

)

# Already defined for Model 1:

# ctrl <- makeTuneControlGrid()

# cv5f <- makeResampleDesc("CV", iters = 5)

# Perform tuning

parallelStartSocket(5)

results2 <- tuneParams(

lrn2,

task = task2,

resampling = cv5f,

par.set = ps2,

control = ctrl,

measures = mae

)

parallelStop()

# Look at tuning results

opt2 <- as.data.table(results2$opt.path)

# Figure 3b

p3b <-

ggplot(opt2, aes(

x = reorder(interaction.depth, as.numeric(as.character(interaction.depth))),

y = mae.test.mean,

group = as.numeric(as.character(shrinkage)),

linetype = reorder(shrinkage, as.numeric(as.character(shrinkage))),

shape = reorder(shrinkage, as.numeric(as.character(shrinkage)))

)) +

geom\_line() +

geom\_point() +

ylab("Cross-validation error") +

xlab("Tree depth") +

scale\_shape\_discrete(name = "Learning rate") +

scale\_linetype\_discrete(name = "Learning rate")

## Choose

## eta = 0.005 (could go smaller)

## max\_depth = 4

## max nrounds = 2000

## subsample = 0.5

# Fit the to the whole training set

fit\_model2 <-

gbm.step(gbm.y = "PV1MATH",

gbm.x = c(demographics, disposition\_indices),

data = as.data.frame(stdq2[train\_set\_2, ]),

family = "gaussian",

n.folds = 5,

n.trees = 1500,

max.trees = 1500,

learning.rate = 0.005,

tree.complexity = 10,

bag.fraction = 0.5,

verbose = T

)

# Make predictions from the holdout set

pred\_model2 <- data.table(

truth = stdq2[test\_set\_2, PV1MATH],

response = predict(fit\_model2, as.data.frame(stdq2[test\_set\_2, ]), n.trees = 1500)

)

# Figure 4b

p4b <-

ggplot(pred\_model2, aes(truth, response)) +

geom\_point() +

geom\_abline(slope = 1, linetype = "dashed") +

coord\_cartesian(xlim = c(200, 800), ylim = c(200, 800))

# Relative influences for Model 2

imp\_model2 <- fit\_model2$contributions

### PD plots in ggplot

# Built-in function:

# gbm.plot(fit\_model2, n.plots = 12)

pd\_model2 <-

foreach(k = 1:12) %do% {

as.data.table(gbm::plot.gbm(fit\_model2, k, return.grid = TRUE))

}

pd\_model2 <- Reduce(function(x, y) merge(x, y, by = "y", all = T), pd\_model2)

for(col in names(pd\_model2)) {

if(is.factor(pd\_model2[[col]])) {

set(pd\_model2, j = col, value = as.numeric(pd\_model2[[col]]))

}

}

pd\_model2\_long <- melt(pd\_model2, id.vars = "y")

# Factorise the variables and order by relative influence

pd\_model2\_long[, variable := factor(variable, levels = fit\_model2$contributions$var %>% as.character)]

# Figure 7

p7 <-

ggplot(pd\_model2\_long, aes(value, y)) +

geom\_point() +

facet\_wrap(~variable, scales = "free\_x") +

ylab("Mathematics literacy score")

# Interactions

int\_model2 <- gbm.interactions(fit\_model2)

int\_model2\_rankings <- int\_model2$rank.list

# Plotting range

dr <- c(-3.75, 3.907)

# Figure 8a: ANXMAT 8 vs MATHEFF 6

p8a <- gbm.perspec(fit\_model2, 8, 6, z.range = c(350, 600), x.range = dr, y.range = dr)

# Figure 8b: ANXMAT 8 vs SCMAT 7

p8b <- gbm.perspec(fit\_model2, 8, 7, z.range = c(400, 580), x.range = dr, y.range = dr)

# Figure 8c: ANXMAT 8 vs ESCS 2

p8c <- gbm.perspec(fit\_model2, 8, 2, z.range = c(400, 580), x.range = dr, y.range = dr)

# Figure 8d: MATHEFF 6 vs ESCS 2

p8d <- gbm.perspec(fit\_model2, 2, 6, z.range = c(300, 580), x.range = dr, y.range = dr)

# Figure 8e: ANXMAT 8 vs INTMAT 9

p8e <- gbm.perspec(fit\_model2, 9, 8, z.range = c(400, 580), x.range = dr, y.range = dr)

# Figure 8f: ANXMAT vs GENDER

p8f <- gbm.perspec(fit\_model2, 8, 1, z.range = c(425, 580), x.range = dr, y.range = c(0, 1))