Protein Production Prediction

James Young

2/13/2020

## The Problem and The Value

Increasing protein titres in production has important implications for metabolic engineering, food production, and therapeutic peptide/protein production. However, manipulating protein expression levels can be very challenging because of their multi-tier regulation.

## The Approach

Using xgboost I made a predictive model for translational efficiency of eventual proteins based on their nucleotide and amino acid sequences as well as the physical characteristics of those molecules such as hydrophobicity, molecular weight, rareness of codons, RNA folding energy, and charge. The proteins were given a score (protein expression level) based on experimental work found in this article <https://www.nature.com/articles/s41467-019-13810-1#Sec23>. I tried to predict that score (protein level) using my previously described input variables.

## Prepare The Data

First I will read in the data and split the nucleotide and amino acid sequences into separate columns.

x <- read.csv("Proteins.csv")

library(data.table)  
fixed\_split <- function(text, n) {  
 data.table::tstrsplit(text, paste0("(?<=.{",n,"})"), perl=TRUE)  
}  
x1 <- cbind(as.character(x$AA))   
x1 <- as.data.frame(x1)  
x1$V1 <- as.character(x1$V1)  
n <- 1  
new\_vars <- ceiling(max(nchar(x1$V1)) / n)  
x1 <- setDT(x1)[, paste0("new\_var", seq\_len(new\_vars)) := fixed\_split(V1, n = n)][]

library(data.table)  
fixed\_split <- function(text, n) {  
 data.table::tstrsplit(text, paste0("(?<=.{",n,"})"), perl=TRUE)  
}  
x02 <- cbind(as.character(x$DNA))   
x02 <- as.data.frame(x02)  
x02$V1 <- as.character(x02$V1)  
n <- 3  
new\_vars <- ceiling(max(nchar(x02$V1)) / n)  
x02 <- setDT(x02)[, paste0("new\_var", seq\_len(new\_vars)) := fixed\_split(V1, n = n)][]

library(data.table)  
fixed\_split <- function(text, n) {  
 data.table::tstrsplit(text, paste0("(?<=.{",n,"})"), perl=TRUE)  
}  
x4 <- cbind(as.character(x$DNA))   
x4 <- as.data.frame(x4)  
x4$V1 <- as.character(x4$V1)  
n <- 1  
new\_vars <- ceiling(max(nchar(x4$V1)) / n)  
x4 <- setDT(x4)[, paste0("new\_var", seq\_len(new\_vars)) := fixed\_split(V1, n = n)][]

x2 <- cbind(x$tAI, x$nco, x$xho, x$RLI, x$Rare\_3, x$Rare\_2, x$RNAfold, x$charge, x$hydrophobicity, x$Score)

x2 <- as.data.frame(x2)  
names(x2) <- c("tAI", "nco", "xho", "RLI", "Rare3", "Rare2", "RNAfold", "Charge", "hydrophob", "Score")  
x2$nco<- as.factor(x2$nco)  
x2$xho<- as.factor(x2$xho)  
x2$RLI<- as.factor(x2$RLI)  
x2$Rare3<- as.factor(x2$Rare3)  
x2$Rare2<- as.factor(x2$Rare2)

Here is further data prep and I am adding in additional information regarding the molecular weight of the amino acid coded for as well as hydrophobicity.

x1 <- as.data.frame(x1)  
x2 <- as.data.frame(x2)  
#x02 <- as.data.frame(x02)  
x4 <- as.data.frame(x4)  
x1 = setDT(x1, keep.rownames = TRUE)[]  
x2 = setDT(x2, keep.rownames = TRUE)[]  
x4 = setDT(x4, keep.rownames = TRUE)[]  
x3 <- merge(x1, x2, by="rn", all=TRUE)  
x3 <- merge(x4, x3, by="rn", all=TRUE)  
library(dplyr)  
x22 <- c(3,4,5)  
  
x3$new\_var1.y <- as.factor(x3$new\_var1.y)  
x3$new\_var2.y <- as.factor(x3$new\_var2.y)  
x3$new\_var3.y <- as.factor(x3$new\_var3.y)  
  
x1 <- as.data.frame(x1)  
x21 = x1  
  
for (i in x22) {x21[,i] <- (recode(as.character(x21[,i]), "A" = 89 , "C" = 121, "D"= 133, "E" = 147, "F" = 165, "G" = 75, "H" = 155, "I" = 131, "K" = 146, "L" = 131, "M" = 149, "N" = 132, "P" = 115, "Q" = 146, "R" = 174, "S" = 105, "T" = 119, "V" = 117, "W" = 204, "Y" = 181, "\*" = 0))  
}  
  
x6 = x21  
x21 = x1  
  
for (i in x22) {x21[,i] <- (recode(as.character(x21[,i]), "A" = 1.8 , "C" = 2.5, "D"= -3.5, "E" = -3.5, "F" = 2.8, "G" = -0.4, "H" = -3.2, "I" = 4.5, "K" = -3.9, "L" = 3.8, "M" = 1.9, "N" = -3.5, "P" = -1.6, "Q" = -3.5, "R" = -4.5, "S" = -0.8, "T" = -0.7, "V" = 4.2, "W" = -0.9, "Y" = -1.3, "\*" = -999))  
}  
x7 = x21  
  
x3 <- merge(x6, x3, by="rn", all=TRUE)  
x3 <- merge(x7, x3, by="rn", all=TRUE)

## Warning in merge.data.frame(x7, x3, by = "rn", all = TRUE): column names 'V1.x',  
## 'new\_var1.x', 'new\_var2.x', 'new\_var3.x', 'V1.y', 'new\_var1.y', 'new\_var2.y',  
## 'new\_var3.y' are duplicated in the result

x3$rn <- NULL  
x3$V1.x <- NULL  
x3$V1.y<- NULL  
#x3$nco <- NULL  
#x3$xho <- NULL  
#x3$RNAfold <- NULL  
x3$V1.x <- NULL  
x3$V1.y<- NULL  
x3$V1.x <- NULL  
x3$V1.y<- NULL  
  
names(x3) <- c("hydro1", "hydro2", "hydro3", "MW1", "MW2", "MW3", "N1", "N2", "N3", "N4", "N5", "N6", "N7", "N8", "N9", "AA1", "AA2", "AA3", "tAI", "nco", "xho", "RLI", "Rare3", "Rare2", "RNAfold", "Charge", "hydrophob", "Score")

set.seed(12345)  
  
x2 = x3  
x2$N1 <- as.factor(x2$N1)  
x2$N2 <- as.factor(x2$N2)  
x2$N3 <- as.factor(x2$N3)  
x2$N4 <- as.factor(x2$N4)  
x2$N5 <- as.factor(x2$N5)  
x2$N6 <- as.factor(x2$N6)  
x2$N7 <- as.factor(x2$N7)  
x2$N8 <- as.factor(x2$N8)  
x2$N9 <- as.factor(x2$N9)  
  
sample <- sample.int(n = nrow(x2), size = floor(.7\*nrow(x2)), replace = F)  
train <- x2[sample, ]  
test <- x2[-sample, ]

## Split The Data 90/10

I have split the data into train/test groups to see the generalization of the model using a 90/10 split. This means we will validate on about 17,000 samples.

# Load the Matrix package  
library(Matrix)  
set.seed(123)  
x2=x3  
x2 = na.omit(x2)  
x2$V1 <- NULL  
x2$nco <- NULL  
x2$xho <- NULL  
x2$rn <- NULL  
x2$V1.y <- NULL  
x2$V1.x <- NULL  
x2$Rare2 <- NULL  
x2$Rare3 <- NULL  
x2$tAI <- NULL  
x2$RLI <- NULL  
  
  
  
library(dplyr)  
  
x2=na.omit(x2)  
x2 = as.data.frame(x2)  
  
  
x2<- na.omit(x2)  
  
  
sample <- sample.int(n = nrow(x2), size = floor(.9\*nrow(x2)), replace = F)  
train <- x2[sample, ]  
test <- x2[-sample, ]  
  
# Create sparse matrixes and perform One-Hot Encoding to create dummy variables  
dtrain <- sparse.model.matrix(Score ~ .-1, data=train)  
dtest <- sparse.model.matrix(Score ~ .-1, data=test)  
dim(dtrain)

## [1] 157358 97

dim(dtest)

## [1] 17485 97

library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

train.label <- train$Score  
test.label <- test$Score  
dtrain <- xgb.DMatrix(data = dtrain,label = train.label)   
  
dtest <- xgb.DMatrix(data = dtest,label=test.label)

#default parameters  
params <- list(booster = "gbtree", objective = "reg:linear", eta=0.1, gamma=1, max\_depth=9, min\_child\_weight=1, subsample=1, colsample\_bytree=1)

set.seed(123)  
xgb1 <- xgb.train (params = params, data = dtrain, nrounds = 300, watchlist = list(val=dtest,train=dtrain), print.every.n = 20, early.stop.round = 10, maximize = F , eval\_metric = "rmse")

## Warning: 'print.every.n' is deprecated.  
## Use 'print\_every\_n' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

## Warning: 'early.stop.round' is deprecated.  
## Use 'early\_stopping\_rounds' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

## [1] val-rmse:2.327972 train-rmse:2.334383   
## Multiple eval metrics are present. Will use train\_rmse for early stopping.  
## Will train until train\_rmse hasn't improved in 10 rounds.  
##   
## [21] val-rmse:0.735810 train-rmse:0.726741   
## [41] val-rmse:0.657825 train-rmse:0.635677   
## [61] val-rmse:0.639149 train-rmse:0.606999   
## [81] val-rmse:0.629900 train-rmse:0.589561   
## [101] val-rmse:0.624083 train-rmse:0.577195   
## [121] val-rmse:0.620115 train-rmse:0.568029   
## [141] val-rmse:0.616405 train-rmse:0.558720   
## [161] val-rmse:0.614382 train-rmse:0.552546   
## [181] val-rmse:0.611864 train-rmse:0.545186   
## [201] val-rmse:0.610497 train-rmse:0.540030   
## [221] val-rmse:0.608463 train-rmse:0.533105   
## [241] val-rmse:0.607542 train-rmse:0.527965   
## [261] val-rmse:0.606870 train-rmse:0.523745   
## [281] val-rmse:0.606192 train-rmse:0.519173   
## [300] val-rmse:0.605878 train-rmse:0.515570

## Visualizaing Important Variables

Here you can see how frequently different variables are used in the trees of the xgboost model, suggesting they are important variables. While it is common place to regard some machine learning models as hard to interpret, xgboost actually has a nice waterfall plot that can show how variables played into any given prediction. I will add that at a later date but the intstructional can be found here <https://stats.stackexchange.com/questions/342090/xgboostexplainer-intercept-and-xgboost-parameters>.

xgb.importance(colnames(dtrain, do.NULL = TRUE, prefix = "col"), model = xgb1)

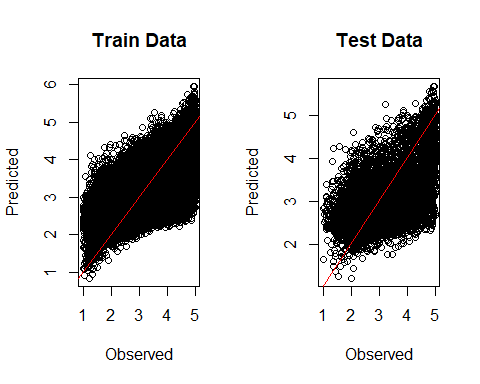
## Feature Gain Cover Frequency  
## 1: RNAfold 0.2597001829 0.2212702544 0.1728196542  
## 2: MW1 0.0800068865 0.0320978723 0.0381612320  
## 3: hydro1 0.0525327281 0.0155611544 0.0318120783  
## 4: N6G 0.0431485440 0.0203135051 0.0313644878  
## 5: N4G 0.0394144582 0.0064622075 0.0102780034  
## 6: N3G 0.0385043266 0.0171305350 0.0266399218  
## 7: hydrophob 0.0384918165 0.1283590884 0.0844122474  
## 8: MW3 0.0352090306 0.0224999462 0.0396532003  
## 9: MW2 0.0314212025 0.0305613221 0.0405815361  
## 10: N7G 0.0266790079 0.0077093650 0.0123004492  
## 11: N5G 0.0228434572 0.0062980956 0.0091838934  
## 12: Charge 0.0226615596 0.0483737966 0.0392387647  
## 13: N1G 0.0221585870 0.0060994882 0.0066475474  
## 14: hydro2 0.0214145540 0.0210679075 0.0332377368  
## 15: hydro3 0.0180944902 0.0214026805 0.0332708917  
## 16: N9C 0.0166428720 0.0156288773 0.0271869768  
## 17: N3C 0.0160975648 0.0148933010 0.0246174759  
## 18: AA1K 0.0143580789 0.0134283103 0.0029673591  
## 19: N6C 0.0130181507 0.0167262746 0.0220811299  
## 20: N9G 0.0115266489 0.0153711684 0.0172736767  
## 21: AA1N 0.0104711140 0.0109114367 0.0031662881  
## 22: N8G 0.0098933366 0.0053112553 0.0069956733  
## 23: N9T 0.0087797861 0.0149829422 0.0147704856  
## 24: N3T 0.0080878495 0.0104067372 0.0165276926  
## 25: AA3K 0.0069996319 0.0100017447 0.0048240306  
## 26: N4C 0.0066957982 0.0083876589 0.0123004492  
## 27: N7C 0.0061532841 0.0091677841 0.0139747692  
## 28: N6T 0.0060055569 0.0094738265 0.0166934668  
## 29: AA2N 0.0056331609 0.0115680505 0.0039454271  
## 30: AA2K 0.0054493337 0.0119502883 0.0047577209  
## 31: N1T 0.0052302117 0.0036737795 0.0053876631  
## 32: N8C 0.0046181753 0.0050319432 0.0073603766  
## 33: N2C 0.0045168629 0.0020271215 0.0050395372  
## 34: AA2Y 0.0044590950 0.0083726869 0.0031662881  
## 35: N1A 0.0042843074 0.0079843565 0.0078576994  
## 36: N4T 0.0042309273 0.0054649738 0.0083715996  
## 37: AA1S 0.0041197476 0.0051744742 0.0047079887  
## 38: N1C 0.0038827707 0.0107006764 0.0067304345  
## 39: N2G 0.0032901377 0.0052552633 0.0043101305  
## 40: AA2P 0.0032452510 0.0051533451 0.0052716211  
## 41: AA1Q 0.0027266946 0.0054316546 0.0042769756  
## 42: AA1H 0.0025615890 0.0060304723 0.0045422144  
## 43: AA1M 0.0025177277 0.0039310839 0.0032325978  
## 44: N7T 0.0023444068 0.0021720399 0.0073437992  
## 45: N5C 0.0022922077 0.0013810945 0.0046582564  
## 46: AA3P 0.0021368928 0.0046700924 0.0046085241  
## 47: AA1L 0.0020693930 0.0052121588 0.0041112014  
## 48: AA3Q 0.0017906773 0.0044904749 0.0053047760  
## 49: AA2Q 0.0017214741 0.0044069014 0.0044427499  
## 50: AA2L 0.0017074913 0.0032992275 0.0038956948  
## 51: AA1Y 0.0016125010 0.0082285923 0.0028678945  
## 52: AA1P 0.0016100553 0.0033509906 0.0033983721  
## 53: AA1T 0.0015830053 0.0038266270 0.0037630754  
## 54: AA2S 0.0015759296 0.0055411469 0.0039122723  
## 55: AA3H 0.0015701721 0.0052928287 0.0038625400  
## 56: AA3N 0.0015321942 0.0060288269 0.0041609336  
## 57: AA3Y 0.0015096596 0.0059234865 0.0013261940  
## 58: AA2M 0.0014777303 0.0031685589 0.0030999784  
## 59: AA2T 0.0014556762 0.0022470441 0.0042935530  
## 60: AA2H 0.0014079462 0.0047817885 0.0034646818  
## 61: AA3S 0.0013940840 0.0024727095 0.0040614691  
## 62: AA3D 0.0013832522 0.0028326493 0.0025031912  
## 63: AA1V 0.0013492952 0.0034553483 0.0025860783  
## 64: AA3L 0.0012964206 0.0026735876 0.0043930176  
## 65: AA3T 0.0012848410 0.0027072344 0.0044593273  
## 66: AA1C 0.0012342368 0.0037515285 0.0030170913  
## 67: AA3E 0.0011946210 0.0015586001 0.0018732490  
## 68: AA3M 0.0011507674 0.0039141017 0.0021716427  
## 69: AA3A 0.0010328535 0.0022009244 0.0030502462  
## 70: AA2F 0.0010318327 0.0033461637 0.0031994430  
## 71: AA1A 0.0009928364 0.0012484108 0.0020058684  
## 72: AA2D 0.0009484584 0.0022451953 0.0026523880  
## 73: AA1D 0.0009235412 0.0031812924 0.0023374169  
## 74: AA1F 0.0009155360 0.0097476367 0.0015582779  
## 75: AA3F 0.0008831647 0.0016284027 0.0031828656  
## 76: AA1E 0.0008402199 0.0034677816 0.0023871492  
## 77: AA2C 0.0008263340 0.0030604886 0.0024534589  
## 78: AA2V 0.0008034679 0.0028045813 0.0022711072  
## 79: N5T 0.0006822463 0.0014142028 0.0007957164  
## 80: AA3V 0.0006601043 0.0012967021 0.0025363460  
## 81: AA2A 0.0005851494 0.0045069037 0.0017737845  
## 82: N8T 0.0005208472 0.0002401137 0.0006299421  
## 83: AA3C 0.0004760286 0.0026535852 0.0021053330  
## 84: AA2E 0.0003100392 0.0017700553 0.0012764617  
## 85: N2T 0.0001059120 0.0001512145 0.0002652388  
## Feature Gain Cover Frequency

## Visualize The Predictions

Further below, we can see the predictions of the model for both train and test data. While there is variaition away from the true score, we can see these predictions do follow the trend line pretty well on average.

pred <- as.data.frame(predict(xgb1, dtest, type = 'response'))  
pred2 <- as.data.frame(predict(xgb1, dtrain, type = 'response'))

par(mfrow=c(1,2))  
plot(pred2$`predict(xgb1, dtrain, type = "response")` ~ train.label, xlab="Observed", ylab ="Predicted", main = "Train Data")  
abline(a=0, b=1, col="red")  
plot(pred$`predict(xgb1, dtest, type = "response")` ~ test.label, xlab="Observed", ylab ="Predicted", main = "Test Data")  
abline(a=0, b=1, col="red")



## Conclusions (Thus Far)

When attempting to increase the titre of proteins, it may be useful to consider the factors found to be important in this model. At a future date, I will interrogate the finer detail interactions of the variables, but the purpose of this project was to try to create a predictive model of protein production levels. From the RMSE of ~ 0.6 through a range of 1 to 5, I would say this model does a that fairly decently.