Spring 2022

Breeding Sweet Corn for Vitamin A

Sam Herr

Dan Pollard

Breeding Sweet Corn for Vitamin A

Sam Herr

2022-06-13

Introduction

This tutorial aims to inform readers on how to perform a multi-kernel genomic selection. The term "multi-kernel" is meant in the sense that we will have multiple kernels with different subsets of single nucleotide polymorphisms (SNPs) that will be weighted differently with respect to the phenotype or trait. A multi-kernel model can improve prediction accuracy by separating non-biologically informative SNPs from those that have an impact of the trait of interest. Following this tutorial's steps, you, the reader, should be able to apply different types of biological information to increase the prediction accuracy of various types of genomic selection models.

Dataset

I want to start this tutorial off with a discussion of the dataset. We will use data from multiple papers, but the primary resource will be from "Transcriptome-wide association and prediction for carotenoids and tocochromanols in fresh sweet corn kernels" by Jenna Hershberger et al. Specifically, the data was collected from the Gore Lab's data repository, given here https://datacommons.cyverse.org/browse/iplant/home/shared/GoreLab/dataFromPubs/Hershberger_SweetCornRNA_2021.

In the study, the authors generated gene expression from young kernels of 308 lines of corn. This gene expression was then correlated by transcriptome-wide association studies (TWAS) to the abundance of carotenoids gathered by Baseggio et al. 2021. The authors then used gene expression and genomic data to predict various carotenoids and tocochromonals. They found that the combination of gene expression and genomic information can improve prediction accuracy. They also found previously uncorrelated genes to specific tocochromonals and carotenoid abundance.

This tutorial will focus solely on the carotenoid data from the paper, and before we go any further, let's first discuss what a carotenoid is. Carotenoids are a family of red, yellow, and orange organic pigments that include pigments with vitamin A activity, which means the compound can be converted into retinol. The pigments with vitamin A activity include beta-carotene and beta cryptoxanthin. Sufficient levels of vitamin A are essential for a healthy life as deficiency can lead to weakened immune function, blindness, and ultimately death from severe infection. There is a need for increased access to vitamin A as there are more than 127 million preschool-aged children and 7 million pregnant women who are vitamin A deficient in the world (West 2002). Other carotenoids that cannot be converted into retinol include lutein and zeaxanthin, which are important for visual function.

Here is the carotenoid data:

```
####Phenotype Data
```

```
suppressPackageStartupMessages(library(tidyverse,warn.conflicts = F))
#Read in file
vitA <- read.csv("carotenoid_transformed_BLUPs-converted.csv",1)
#Get correct column names
names(vitA) <- vitA[1,]
colnames(vitA)  <- c("Sample.ID", "Additional.ID.name", "GBS.ID",
                     "Endosperm.mutation", "Antheraxanthin",
                     "beta.Carotene", "beta.Cryptoxanthin", "Lutein",
                     "Violaxanthin", "Zeaxanthin", "Zeinoxanthin",
                     "Other.carotenes", "alpha.Xanthophylls",
                     "beta.Xanthophylls", "Total.xanthophylls",
                     "Total.carotenes", "Total.carotenoids",
                     "beta.Carotene_over_beta.cryptoxanthin",
                     "beta.Carotene_over_sum_beta.cryptoxanthin_and_zeaxanthi
n",
                     "beta.Cryptoxanthin_over_zeaxanthin",
                     "Zeinoxanthin_over_lutein",
                     "beta.Xanthophylls_over_alpha.xanthophylls",
                     "Total.carotenes_over_total.xanthophylls",
                     "X")
colnames(vitA) <- gsub("[.]", "_", colnames(vitA))

#Remove unwanted (the last) column and row
vitA <- vitA[-c(1,lengths(vitA)[1]),]
vitA <- vitA[,-length(vitA)]

#Inspect the phenotypes
head(vitA[,c(1,5,6,7,8)])
```

```
##   Sample_ID Antheraxanthin beta_Carotene beta_Cryptoxanthin Lutein
## 2         2         -0.035        -1.329             -1.175  1.573
## 3     257A9         -0.240        -0.456             -1.167  0.376
## 4       34f          0.688        -0.095              0.784  2.515
## 5    83610b          0.351        -1.062             -1.403  0.279
## 6    A10579          0.158        -1.024             -0.522  1.856
## 7 A632_susu         -0.804        -0.786             -1.491 -0.119
```

```
print(dim(vitA))
```

```
## [1] 308  23
```

```
#save as an .rds for ease of access
saveRDS(vitA, file = "pheno.rds")
```

For every Sample ID (denoting the line of corn), there are 19 phenotypes, including lutein and beta carotene. The values for each phenotype are the best unbiased linear predictor, which is the genomic proportion of the observed value. This is done through mixed modeling. If interested, read more here: https://en.wikipedia.org/wiki/Best_linear_unbiased_prediction and https://plant-breeding-genomics.extension.org/estimating-heritability-and-blups-for-traits-using-tomato-phenotypic-data/.

For each phenotypic value, we will be using genetic information in the form of bi-allelic SNPs for predictions.

Let's look at the genetic data:

```
####Genotype Data


#Open genetic information that is encoded as a Hapmap file
full_snp_mat <- as.data.frame(read.delim("Ion_11k_401_v4.hmp.txt"))


#Function to convert characters to their respective numeric value
#2 represents major homozygote, 1 represents heterozygotes, 0 represents mino
r homozygote
#NA values are set 1, following GAPIT imputation rules
convert_to_numeric <- function(snp_row){
    num = unlist(strsplit(snp_row[2],  "/"))
    snp_row[snp_row == num[1]] = 2
    snp_row[snp_row == num[2]]  = 0
    snp_row[snp_row %in% c("N","R","Y", "S","W","K","M")] = 1
  return(snp_row)
}


#apply function to every row and then transpose matrix for future ease
full_snp_mat <- as.data.frame(t(apply(full_snp_mat,  1, convert_to_numeric)))


#convert select columns to numeric for future functions
full_snp_mat[,12:length(full_snp_mat)] <- as.matrix(sapply(full_snp_mat[,12:l
ength(full_snp_mat)], as.numeric))
```

```
full_snp_mat$chrom <- as.integer(full_snp_mat$chrom)
full_snp_mat$pos <- as.integer(full_snp_mat$pos)


#Correct label column and row names
colnames(full_snp_mat) <- gsub("\\..*" ,"",colnames(full_snp_mat) , perl = T)
colnames(full_snp_mat) <- gsub("^X", "", colnames(full_snp_mat), ignore.case
=F, perl = T)
rownames(full_snp_mat) <- full_snp_mat$rs


#Save dataframe for subsetting
saveRDS(full_snp_mat, "full_SNP_M.rds")


#keeps only markers for vitamin A genotypes
vitA_marker_mat <- full_snp_mat %>% select(any_of((vitA$Sample_ID)))
vitA_marker_mat <- vitA_marker_mat[,vitA[,1]]
vitA_marker_mat <- t(vitA_marker_mat)
head(vitA_marker_mat[,c(1:5)])
##           S1_1000282 S1_2091448 S1_2307029 S1_2469093 S1_2691818
## 2                  2          2          2          2          2
## 257A9              2          2          2          2          0
## 34f                2          2          2          2          0
## 83610b             2          0          0          2          2
## A10579             2          2          2          2          0
## A632_susu          2          0          0          2          2
#Save matrix for predictions
saveRDS(vitA_marker_mat, "vitA_M.rds")
```

As we can see from the first five rows and columns of our genotype matrix, we now have a numeric representation of genetic markers for each line. The next step in this tutorial is to grasp how genomic selection works, in other words how we can use genetic information to predict phenotypes.

References:

Baseggio, M., Murray, M., Magallanes-Lundback, M., Kaczmar, N., Chamness, J., Buckler, E. S., … & Gore, M. A. (2020). Natural variation for carotenoids in fresh kernels is controlled by uncommon variants in sweet corn. The Plant Genome, 13(1), e20008.

Hershberger, J., Tanaka, R., Wood, J. C., Kaczmar, N., Wu, D., Hamilton, J. P., … & Gore, M. A. (2021). Transcriptome-wide association and prediction for carotenoids and tocochromanols in fresh sweet corn kernels.

West, K. P. (2002). Extent of vitamin A deficiency among preschool children and women of reproductive age. Journal of Nutrition, 132, 2857S–2866S.

Part 2: Theory of Genomic Selection

Sam Herr

2022-06-13

Genomic Selection

Genomic selection, also known as Genomic Prediction, is a modeling technique used in plant breeding to increase the rate of genetic gain. This provides both a time and cost-benefit compared to conventional breeding, especially for complex traits.

The basic idea of genomic selection is to use all possible genetic information (usually in the form of single nucleotide polymorphisms (SNPS)) to correlate the genome to the phenotype of interest.

Let's get some theory behind genomic selection and understand some of the most common models used.

Theory behind the models

A general random model for genomic selection is:

$Y=\mu+X\beta+\epsilon Y=\mu+X\beta+\epsilon Y=\mu+X\beta+\epsilon$,
Where YYY is an $m\times 1 m\times 1 m\times 1$ vector containing phenotypic information, $\mu\mu\mu$ is the overall mean, X is a $m\times nm\times nm\times n$ matrix, with mmm individuals and nnn genetic markers, and $\beta\beta\beta$ is a $n\times 1 n\times 1 n\times 1$ vector of random additive effect size of each marker and $\epsilon\epsilon\epsilon$ is the random error drawn from a $N(0,\sigma_e)N(0,\sigma_e)N(0,\sigma_e)$. Different models have different distributions from which $\beta\beta\beta$ is drawn, however, a common distribution would be that $\beta\sim N(0,\sigma^2_g)\beta\sim N(0,\sigma^2_g)\beta\sim N(0,\sigma^2_g)$. This form of the model is the essence of rrBLUP (Ridge Regression Best Linear Unbiased Predictor) (Meuwissen et al., 2001). The assumption of this model is that every the marker effect comes from the same distribution, while in reality, that might not be the case for some phenotypic traits. This assumption can cause markers with large effects to be underestimated.
As in many cases for plant or livestock breeding, the number of markers, m, can greatly exceed the number of individuals phenotyped, n, also known as "The Curse of Dimensionality"(Wikipedia link: https://en.wikipedia.org/wiki/Curse_of_dimensionality). This curse can be computationally expensive and cause overfitting. Thus, many scientists have turned to use a genomic selection known as GBLUP (Genomic Best Linear Unbiased Predictors), which uses an additive genomic relationship matrix instead of marker values to predict phenotypic values. The general form of GBLUP is as follows:

$Y=\mu+g+\epsilon Y=\mu+g+\epsilon Y=\mu+g+\epsilon$,
Where ggg is a random parameter drawn from $N(0,G\sigma^2_g)N(0,G\sigma^2_g)N(0,G\sigma^2_g)$. GGG denotes a $m\times mm\times mm\times m$ genomic relationship matrix, calculated by $G=WW'\gamma G=WW'\gamma G=WW'\gamma$. WWW is a $nxmnxmnxm$ matrix where $w_{ij}=x_{ij}-2p_iw_{ij}=x_{ij}-2p_iw_{ij}=x_{ij}-2p_i$, where $p_ip_ip_i$ is the allele frequency of a marker and jjj is the jth individual. $\gamma=2\sum(p_i(1-p_i))\gamma=2\sum(p_i(1-p_i))\gamma=2\sum(p_i(1-p_i))$. GBLUP has been shown to be mathematically equivalent to rrBLUP and falls victim to the same assumption that every marker effect is assumed to have the same effect size distribution. These models also

assume there is only an additive (linear) effect of the markers. While for some traits, this might be true, many traits have epistatic and dominance effects that determine the phenotype. In order to account for non-linear interactions, we can use RKHS (Reproducing Kernel Hilbert Space).

A helpful way to think about RKHS is that it is very similar to GBLUP but has the form:

$Y=\mu+g(X)+\epsilon Y=\mu+g(X)+\epsilon Y=\mu+g(X)+\epsilon$
Where $g(X)g(X)g(X)$ is an unknown function of genetic effects. This differs from GBLUP, where the assumption is ggg is a linear function of genetic effects.
If you are interest in further learning about these models, Morota et al. 2014 does a great job at explaining. www.frontiersin.org/journal/10.3389/fgene.2014.00363/abstract

Now that we have talked about different types of genomic selection models, lets give one a try.

rrBLUP

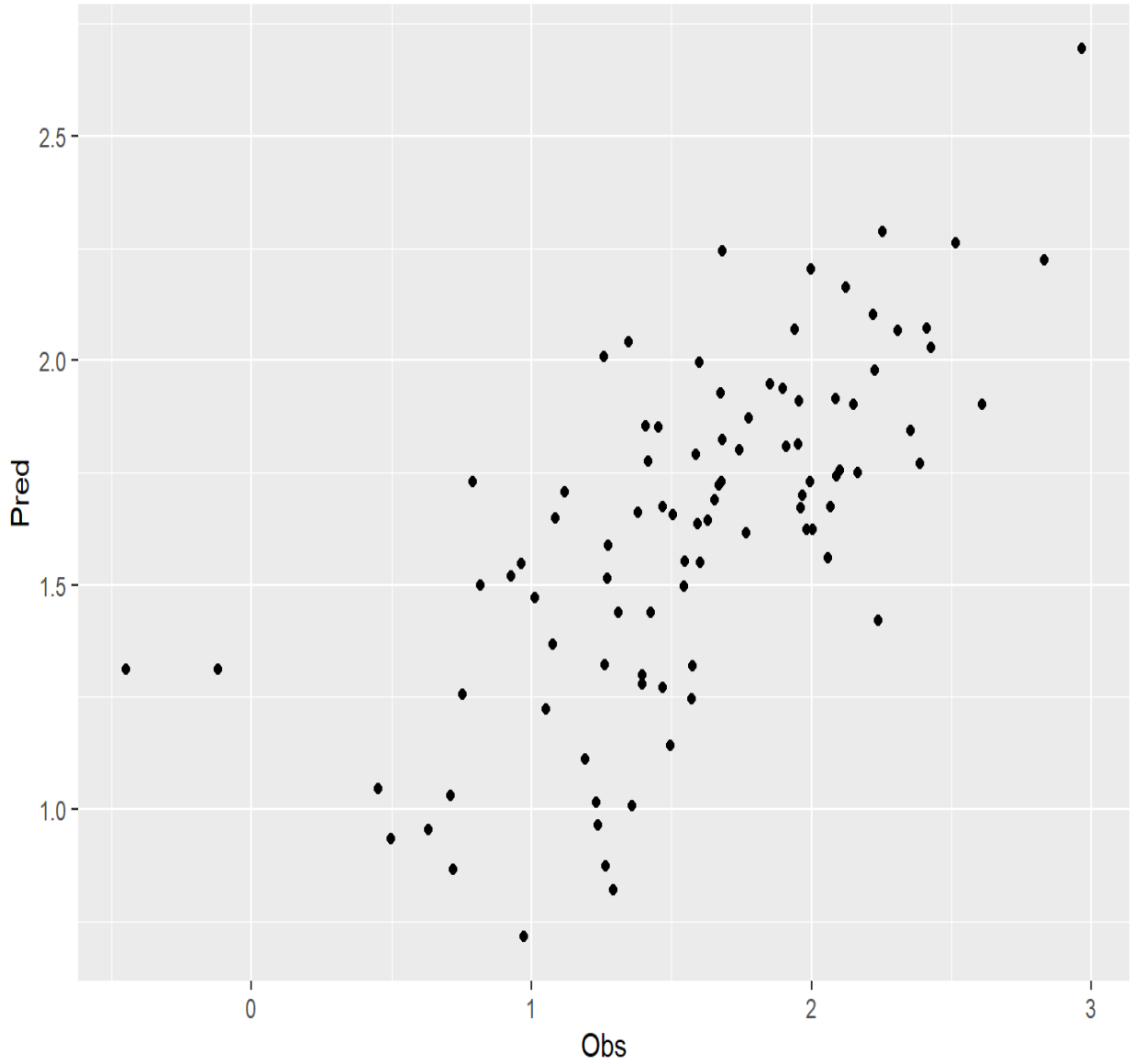
We will use the package rrBLUP for performing rrBLUP and GBLUP models.

```
#load packages and data
suppressPackageStartupMessages(library(tidyverse))
library(rrBLUP)
marker_matrix <- readRDS("vitA_M.rds")
carotenoids <- readRDS("pheno.rds")


set.seed(2022)
#scale marker matrix
marker_scaled <- scale(marker_matrix, center = TRUE, scale = FALSE)
#get phenotype values
#Picked Antheraxanthin as an example
observed <- as.numeric(carotenoids$Lutein)


##lets create a training set where will test 30% and train the model on 70%
#get a vector that matches the total number of individuals
num_individuals <- 1:length(observed)
#randomly grab individuals
test <- sample(x = num_individuals, size = round(length(observed) *0.3), repl
ace = F)


#make training and testing datasets
test_observed <- observed[test]
```

```
test_marker <- marker_scaled[test,]

train_marker <- marker_scaled[-test,]

train_observed <- observed[-test]


#run mixed.solve which performs RR-BLUP

RRBLUP <- mixed.solve(y = train_observed, Z = train_marker, K = NULL)

#Get prediction values by multipling the marker coefficients "u" by the scale
d matrix

#then add the intercept term, "beta"

predicted <- test_marker %*% RRBLUP$u + as.numeric(RRBLUP$beta)

#get correlation

cor(test_observed, predicted)
```

```
##            [,1]

## [1,] 0.6895897
```

```
#Plot to get a general sense of the prediction

#Prediction on the x-axis and Observations on the y-axis

ggplot(data.frame("Pred" = predicted, "Obs" = test_observed), aes(x = Obs, y
= Pred)) + geom_point()
```



rrBLUP performs relatively well as given by the pearson's correlation coefficient and seen in the graph but one thing to cautious about is overfitting due to the curse of dimensionality.

GBLUP

To deal with this curse, we will perform GBLUP. The number of predictors will be reduced from the number of SNPs to the number of lines, from 10,773 to 308. Before we jump into GBLUP, we first have to create a realized genomic matrix.

The equation for this version of realized genomic matrix:

$G=ZZ'2\sum_i p_i(1-p_i)G=ZZ'2\sum_i p_i(1-p_i)G=ZZ'2\sum_i p_i(1-p_i)$
Where ZZZ is a scaled genomic marker matrix, where each marker is subtracted by $2(p_i-0.5)2(p_i-0.5)2(p_i-0.5)$. $p_ip_ip_i$ is the frequency of minor allele for $ithithith$ SNP. $Z'Z'Z'$ is the transpose of Z.

```
#Function taken from GAPIT
#creates genomic matrix 1 from Van Randen 2008
get_genomic_matrix <- function(snps){

  ##removes any SNPs that contain either all the major homozygote or all the minor homozygote
  fa=colSums(snps)/(2*nrow(snps))
  index.non=fa>=1| fa<=0
  snps=snps[,!index.non]

  #get number of snps and then number of individuals
  nSNP=ncol(snps)
  nInd=nrow(snps)
  n=nInd

  ##allele frequency of second allele
  p=colSums(snps)/(2*nInd)
  P=2*(p-.5) #Difference from .5, multiple by 2
  snps=snps-1 #Change from 0/1/2 coding to -1/0/1 coding

  Z=t(snps)-P#operation on matrix and vector goes in direction of column
  #K=tcrossprod((snps), (snps))
  K=crossprod((Z), (Z)) #Thanks to Peng Zheng, Meng Huang and Jiafa Chen for finding the problem



  adj=2*sum(p*(1-p))
  K=K/adj



  return(K)
}
```

With this function for the genomic matrix we can run GBLUP similarly to rrBLUP

```
set.seed(2022)
```

```
#get phenotype values
#Picked Lutein as an example
observed <- as.numeric(carotenoids$Lutein)

##lets create a training set where will test 30% and train the model on 70%
#get a vector that matches the total number of individuals
num_individuals <- 1:length(observed)
#randomly grab individuals
test <- sample(x = num_individuals, size = round(length(observed) *0.3), repl
ace = F)

#make training and testing datasets
test_observed <- observed[test]
train_genomic_matrix <- get_genomic_matrix(marker_matrix)
train_observed <- observed
train_observed[test] <- NA

#run mixed.solve which performs GBLUP
#Notice instead of Z =, we have K =
GBLUP <- mixed.solve(y = train_observed, K = train_genomic_matrix)

#get correlation with the estimated breeding values from GBLUP
cor(test_observed, GBLUP$u[test])
## [1] 0.6895894
```

For our data set with a small amount of markers, the predictive accuracy for both GBLUP and rrBLUP perform roughly the same. In some cases researchers have reported that rrBLUP can perform better than GBLUP when there is strong linkage disequilibrium (Morota et al. 2014).

RKHS

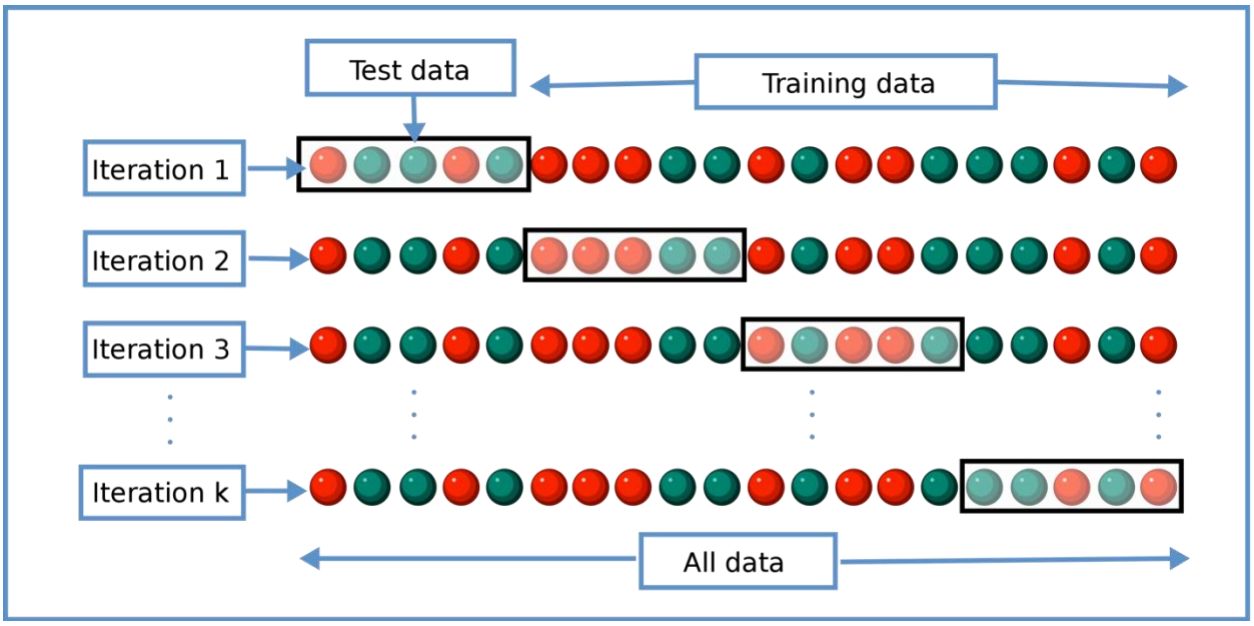
Let's introduce now RKHS modeling. For RKHS we will use the BGLR R package (Perez and de Los Campos 2014) Luckily the process is very similar to GBLUP so we only have to make minor changes

```
library(BGLR)
#format data for BGLR
#number of iterations the MC chain will do and then throw away first 8000
```

```
nIter = 12000
burnIn = 8000
#Use K when using a genomic relationship matrix
#Use X when using a snp matrix
ETA <- list(G = list( K = train_genomic_matrix, model = "RKHS"))


fm <- BGLR(y = train_observed, ETA = ETA,
                  nIter = nIter, burnIn = burnIn, verbose = F)


#get the predicted
predicted <- fm$yHat[test]
cor(test_observed, predicted)
## [1] 0.6969568
```

From the results we can see that all three models perform roughly the same when predicting the same lines of corn. But how will the model generalize to predicting many different lines of corn and how can we do this without collecting more data. Luckily with a technique known as cross validation we can randomly select different subsets of our dataset and some of the subsets to predict the other subsets. This allows to understand how well the model works in a general sense and then be able to compare different types of models more accurately

For our research, we will use five-fold cross validation. For five-fold cross validation, the data is split into five partition. Four of the five partitions will be used as a training set to build the model, the fifth partition will be used as a test set to test the model built by the other four partitions. This is repeated until each partition gets its chance at becoming the test set.



For our dataset we will be doing something a little different from a normal cross validation. This is because sweet corn has two major mutations that make the corn sweet. The lines of corn we have, either have one of the mutations or both. We want our partitions to accurately reflect the ratio of the mutations that is in the original dataset. To do this we will do a weighted cross validation so that the proportion of mutations remains constant throughout all the folds.

Here is the code.

```
#Creates the five fold cross validations for 10 replicates
#code taken from https://github.com/GoreLab/SweetCornRNA/tree/master/CODE
#Thank you Dr. Jenna Hershberger!

set.seed(2020)
#Repetitions
n.rep = 10
#folds
n.fold = 5

#Initialize matrices
df.CV.raw <- data.frame("Sample.ID" = carotenoids$Sample_ID)
mat <- matrix(NA, nr = nrow(df.CV.raw), nc = n.rep)
colnames(mat) <- paste0("Rep", formatC(1:n.rep, width = 2, flag = "0"))

#Creates the cross validation for every repetition.
```

```
#Each fold has an equal proportion of different mutation types
for ( r in 1:n.rep ) {
  #get number of each mutation type (3 total)
  n.tab <- table(carotenoids$Endosperm_mutation)
  #insert 0 for temporary table
  tmp.n.tab <- cumsum(c(0, n.tab))
  cv.num.with.names.all <- c()
  #iterate through each mutation type
  for ( i in 1:length(n.tab) ) {
    #get mutation type
    mu.type <- names(n.tab)[i]
    #select lines with mutation type
    sample.names <- carotenoids$Sample_ID[carotenoids$Endosperm_mutation == m
u.type]
    #create a vector length of mutation type
    seq.num <- (tmp.n.tab[i]+1):(tmp.n.tab[i+1])
    #modulate by n.folds to get
    cv.num <- 1 + seq.num %% n.fold
    #randomly sample to create folds
    cv.num.rand <- sample(cv.num)
    cv.num.with.names <- setNames(cv.num.rand, sample.names)
    cv.num.with.names.all <- c(cv.num.with.names.all, cv.num.with.names)
  }
  cv.num.with.names.all.ord <- cv.num.with.names.all[carotenoids$Sample_ID] #
sort
  mat[, r] <- cv.num.with.names.all.ord
}
df.CV <- cbind(df.CV.raw, mat)


write_rds(df.CV, "cv.rds")
```

Once we have our folds for each repetition we can now then perform cross-validation. Here is another function written by Dr. Jenna Hershberger that performs the cross validation. We will do 10 repetitions of five-fold cross-validation in order to get the mean and variance of the prediction accuracy.

```
#code taken from https://github.com/GoreLab/SweetCornRNA/tree/master/CODE
#Thank you Jenna Hershberger!
```

```r
#10 repetitions of five fold cross validation of BGLR functions
myFun.Cv <- function(y.all, ETA, df.CV,
                     nIter = 12000, burnIn = 8000,
                     ...) {
  # number of repetitions and folds
  n.rep <- ncol(df.CV) - 1
  n.fold <- max(df.CV[, 2])

  # cross validation: n.rep/n.fold
  pred.mat <- matrix(NA, nr = nrow(df.CV), nc = n.rep)
  rownames(pred.mat) <- df.CV[, 1]
  colnames(pred.mat) <- colnames(df.CV)[2:ncol(df.CV)]

  #iterate for each repetition
  for ( j in 1:n.rep ) {
    # make an object to save prediction result
    pred.vec <- rep(NA, length(y.all))
    #iterate through each fold
    for ( k in 1:n.fold ) {
      # cv numbers
      cv.num <- df.CV[, j + 1]

      # cross validation
      y.obs <- y.all
      y.obs[cv.num == k] <- NA # mask phenotype

      # run prediction
      fm <- BGLR(y = y.obs, ETA = ETA,
                 nIter = nIter, burnIn = burnIn, verbose = F
      )
      pred.vec[cv.num == k] <- fm$yHat[cv.num == k]
    }
```

```
    # save
    pred.mat[, j] <- pred.vec
  }

  # output
  return(pred.mat)
}
```

Lets try out the function

```
#Can use ETA and observed values from previous models
ETA <- list(G = list( K = train_genomic_matrix, model = "RKHS"))

#call function for prediction
#pred.mat stores predicted values
pred.mat <- myFun.Cv(y.all = observed, ETA = ETA, df.CV = df.CV,
                               nIter = nIter, burnIn = burnIn,verbose = F
    )

#formatting for correlation
phenotype <- data.frame("Sample.ID" = carotenoids$Sample_ID, "Observed" = car
otenoids$Lutein)
pre.mat <- as.data.frame(pred.mat)
pre.mat <- cbind("Sample.ID" = row.names(pre.mat), pre.mat)
pre.mat <- pre.mat %>% pivot_longer(cols = starts_with("Rep"), names_to = "Fo
ld", values_to = "Predicted")

#Get Pearson's and Spearmean's value for each repetition
#take mean of all lines in one repetition
pre.mat <- pre.mat %>% full_join(phenotype, by = c("Sample.ID")) %>%
  drop_na(Predicted, Observed) %>%
  mutate(Predicted = as.numeric(Predicted),
         Observed = as.numeric(Observed)) %>%
  group_by(Fold) %>%
  mutate(PearsonCor = cor(Predicted, Observed, method = "pearson"),
         SpearmanRankCor = cor(Predicted, Observed, method = "spearman")) %>%
```

```
    dplyr::select(Fold, PearsonCor:SpearmanRankCor) %>%  distinct()



meanCor <- mean(pre.mat$PearsonCor)

meanCor
```

```
## [1] 0.6962613
```

We can see that the cross validation reproduces a similar value as our models without cross-validation. This is great news as it means our models are working well! Now that we have an understanding of each model, let's discuss we should use multi-kernels.

Introduction to multi-kernel

rrBLUP, GBLUP, and RHKS all assume that every marker effect is from the same effect size distribution. To relax this assumption, researchers have used a multi-kernel model or multiBLUP to test if separating select markers into kernels can improve the prediction accuracy of the model. By creating multiple kernels with subsets of SNPs, each kernel will have a different genomic relationship matrix, thus a different variance. The form of a multi-kernel model is as follows:

$$Y=\mu+\sum g_i+\epsilon Y=\mu+\sum g_i+\epsilon Y=\mu+\sum g_i+\epsilon$$

Where $g_i g_i g_i$ denotes the ith kernel and is drawn from a $N(0,G_i\sigma_g)N(0,G_i\sigma_g)N(0,G_i\sigma_g)$.
With this model, SNP markers can have a varying impact on the prediction depending on the kernel placed in. The kernels can be created using prior biological information. An example of this would be a kernel containing markers that are associated with genes that are involved in pathways of a phenotype of interest.

The next part of this tutorial will go over how we find biologically annotated SNPs.

References:

Meuwissen, T.H.E., B.J. Hayes, and M.E. Goddard. 2001. Prediction of total genetic value using genome-wide dense marker maps. Genetics 157:1819–1829.

Morota, G., & Gianola, D. (2014). Kernel-based whole-genome prediction of complex traits: a review. Frontiers in genetics, 5, 363.

Pérez, P., & de Los Campos, G. (2014). Genome-wide regression and prediction with the BGLR statistical package. Genetics, 198(2), 483-495.

GWAS and TWAS

Sam Herr

2022-06-13

How to find biologically annotated SNPs

GWAS

How does one relate genetic information to phenotypes? The the answer to this question is highly sought after by researchers. One current technique is known as Genome-Wide Association Studies (GWAS). GWAS identifies SNPs (single nucleotide polymorphisms) that have an effect on the phenotype of interest. GWAS is used when relationships in a population is unknown, i.e. for human population or diversity panels in plants. The basic example of GWAS is a simple linear regression i.e.

$Y=\beta_i \times X_i Y=\beta_i \times X_i Y=\beta_i \times X_i$
where $\beta\beta\beta$ is a marker (SNP) effect and $X_i X_i X_i$ is a SNP. This model will be done for every single SNP and then the p-value of each marker effect will be corrected for using usually bonferroni or benjamini-hochberg corrections. To improve selection of true significant SNPS from spurious false positives, researchers use such techniques as accounting for kinship and population structure (read more about these techniques with Hoffman 2013, linked
here https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0075707)
With GWAS, we can select significant SNPs to include in the biologically annotated kernel for our multi-kernel models. One caveat about including GWAS results is that the GWAS must be done on training population or be from a different experiment. If you perform GWAS on the same population on which you perform genomic selection, you will be guilty of "insider training" and will be arrested by the feds. All jokes aside, "insider training" can lead to overfitting and your models will not predict outside genotypes as well.

Luckily for us Owens et al (2014) (https://pubmed.ncbi.nlm.nih.gov/25258377/) provides a GWAS that we can use. In order to use this data we must format it correctly.

```
suppressPackageStartupMessages(library(tidyverse))


full_snp_mat <- read_rds("full_SNP_M.rds")


#read in Owens et al 2014 GWAS results
gwas <- read.csv("gwasvitA.csv")


#fix problems with names
gwas$SNP.ID <- gsub(" ", "", gwas$SNP.ID)
```

```
gwas$Trait <- gsub(substr(gwas$Trait[2], 1,3), "beta.", gwas$Trait)

gwas$Trait <- gsub(substr(gwas$Trait[43], 7,9), "alpha.", gwas$Trait)

gwas$Trait <- gsub("/", "_over_", gwas$Trait)

gwas$Trait <- gsub("(beta.cryptoxanthin+zeaxanthin) ", "sum_beta_cryptoxanthi
n_and_zeaxanthin", gwas$Trait, fixed =T)


gwas$Trait <- gsub("Total ", "Total_", gwas$Trait, fixed =T)

gwas$Trait <- gsub("[.]", "_", gwas$Trait )


#collapse multiple traits into one SNP

gwas <- gwas %>%

  group_by(SNP.ID) %>%

  summarise(Trait = toString(Trait) )




#Select SNPs both in GWAS and our SNP dataset

gwas <- gwas[gwas$SNP.ID %in% full_snp_mat$rs,]

colnames(gwas) <- c("rs","pheno")


#save

write_rds(gwas, "owens_gwas_results.rds")
```

Looking at the dataset, we can we have the SNP id in the column “rs” and the trait in the column “pheno”. We will use this dataset to select the SNPs for the GWAS biologically annotated kernel.

```
## # A tibble: 6 x 2

##   rs            pheno

##   <chr>         <chr>

## 1 S1_140888441  "Zeinoxanthin_over_lutein "

## 2 S10_135801294 "beta_Carotene_over_sum_beta_cryptoxanthin_and_zeaxanthin,
beta~

## 3 S10_136106072 "beta_Carotene_over_sum_beta_cryptoxanthin_and_zeaxanthin,
beta~

## 4 S2_24808106   "beta_Carotene_over_sum_beta_cryptoxanthin_and_zeaxanthin"

## 5 S8_136911861  "beta__over_alpha_Xanthophylls "
```

```
## 6 S8_138888278  "beta__over_alpha_Xanthophylls "
```

TWAS

Now that we have discussed a technique that relates DNA to phenotype, lets go a step up and talk transcriptome-wide association studies (TWAS) that relates RNA to phenotype.

The model is basically the same as GWAS except we replace marker and marker effect with RNA abundance and RNA abundance effect. Instead of reporting significant SNPs, TWAS instead reports the genes that are most correlated with the phenotype of interest. Unlike GWAS, it is alright to perform TWAS and then genomic selection on the same dataset as they use two different mechanisms to predict the phenotype, RNA expression and DNA markers, respectively.

Like for GWAS, we have the results for a TWAS analysis from Hershberger et al 2022 linked here https://acsess.onlinelibrary.wiley.com/doi/10.1002/tpg2.20197.

One might ask "How do we get genetic marker information from TWAS, if TWAS only reports which genes are significant?" The quick answer would be the use of SNPs within the genes. The longer answer involves linkage disequilibrium because some of the SNPs not in the significant genes will be linked to the SNPs within the gene due to recombination. To include the linked SNPs, we will include SNPs within 250 kb of the start and end of each gene.

Let's take a look at how to do that:

```
full_snp_mat <- read_rds("full_SNP_M.rds")



#Get TWAS Genes
twas <- read.csv("noharv_B73_pathway_ap_FDR0.05.csv")


#Format Data
twas <- twas %>% select(c("RefGen_v4.Gene.ID", "trait","chr","start","end"))
twas <- as.data.frame(twas)
twas$start <- as.integer(twas$start)
twas$end <- as.integer(twas$end)


#Determines which SNPs are near a gene
check_twas_gene_snps <- function(qtls,snpset){
  snpset <- snpset %>% filter(chrom == as.integer(qtls[3]))
  snpset <- snpset[between(snpset$pos, as.integer(qtls[4])-250000, as.integer
(qtls[5])+250000), ]
  snpset <-  cbind(snpset,"pheno" = replicate(lengths(snpset)[1],qtls[2]))
```

```
  return(snpset)
}


twas <- apply(twas,1, check_twas_gene_snps, full_snp_mat)
twas <- bind_rows(twas)


#Combine the same snps for different traits
twas <- twas %>%
  group_by( rs,chrom, pos) %>%
  summarise(pheno = toString(pheno))
## `summarise()` has grouped output by 'rs', 'chrom'. You can override using
the
## `.groups` argument.
#format and save
twas <- data.frame(rs = twas$rs, pheno = twas$pheno)
row.names(twas) <- twas$rs
twas$pheno <- gsub("[.]", "_", twas$pheno  )


write_rds(twas, "twas.rds")
```

Wonderful! We have our TWAS SNPs that we can use to create another biologically annotated kernel.

The next part of tutorial will go over how to use these biologically annotated kernels to predict traits.

References:

Hershberger, J., Tanaka, R., Wood, J. C., Kaczmar, N., Wu, D., Hamilton, J. P., … & Gore, M. A. (2021). Transcriptome-wide association and prediction for carotenoids and tocochromanols in fresh sweet corn kernels.

Hoffman, G. E. (2013). Correcting for population structure and kinship using the linear mixed model: theory and extensions. PloS one, 8(10), e75707.

Owens, B. F., Lipka, A. E., Magallanes-Lundback, M., Tiede, T., Diepenbrock, C. H., Kandianis, C. B., … & Rocheford, T. (2014). A foundation for provitamin A biofortification of maize: genome-wide association and genomic prediction models of carotenoid levels. Genetics, 198(4), 1699-1716.

Prediction and Plots

Sam Herr

2022-06-13

Multi-kernel Predictions

It is time to put everything together and use two kernels for prediciton and then compare to the models to a single kernel.

We use the same techniques we used in part two but now apply them to two models. In order to do this let's introduce a couple of functions that will be used. There are two new functions needed to create the two kernels for BGLR prediction, get_phenotype_subset, which select SNPs based upon their significance in either GWAS or TWAS for a specific traits, and get_ETA_two_kernel, which creates a list needed for BGLR.

```
#Load packages for part 4
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(BGLR))
library(ggtext)
library(RColorBrewer)
options(dplyr.summarise.inform = FALSE)


#Functions:


#Subsets the Biologically annotated SNPS for the trait needed
get_phenotype_subset <- function(phenotype,full_snp_df, subset_snp_df){
    subset <- subset_snp_df[grepl(paste0("\\b",phenotype, "\\b"), subset_snp_
df$pheno),]
    sig_snps <- full_snp_df[,subset$rs]


  return(sig_snps)
}


#Returns biological annotated kernel and the other SNPs
get_ETA_two_kernel <- function(full_snp_df, subset_snp_df, getAmat ){
  #makes sure there are SNPs associated with trait
```

```
  if(length(subset_snp_df) == 0){
    return(NA)
  #If SNPs exist, create ETA needed for BGLR
  }else{
    remove <- colnames(full_snp_df) %in% colnames(subset_snp_df)
    snps_rest <- get_genomic_matrix(full_snp_df[,!remove])
    if(getAmat == T){
      subset_snp_df <- get_genomic_matrix(subset_snp_df)
    }
    ETA <-list("G1" = list(K = subset_snp_df, model = "RKHS"),"G2"= list(K =
snps_rest, model = "RKHS") )
  }
  return(ETA)
}


#code taken from https://github.com/GoreLab/SweetCornRNA/tree/master/CODE
#Thank you Jenna Hershberger!
#10 repetitions of five fold cross validation of BGLR functions
myFun.Cv <- function(y.all, ETA, df.CV,
                     nIter = 12000, burnIn = 8000,
                     ...) {
  # number of repetitions and folds
  n.rep <- ncol(df.CV) - 1
  n.fold <- max(df.CV[, 2])
  # cross validation: n.rep/n.fold
  pred.mat <- matrix(NA, nr = nrow(df.CV), nc = n.rep)
  rownames(pred.mat) <- df.CV[, 1]
  colnames(pred.mat) <- colnames(df.CV)[2:ncol(df.CV)]
  #iterate for each repetition
  for ( j in 1:n.rep ) {
    # make an object to save prediction result
    pred.vec <- rep(NA, length(y.all))
    #iterate through each fold
    for ( k in 1:n.fold ) {
      # cv numbers
```

```
      cv.num <- df.CV[, j + 1]
      # cross validation
      y.obs <- y.all
      y.obs[cv.num == k] <- NA # mask phenotype
      # run prediction
      fm <- BGLR(y = y.obs, ETA = ETA,
                 nIter = nIter, burnIn = burnIn, verbose = F
      )
      pred.vec[cv.num == k] <- fm$yHat[cv.num == k]
    }
    # save
    pred.mat[, j] <- pred.vec
  }
  # output
  return(pred.mat)
}

#Genomic relationship matrix taken from GAPIT
#Same as the one used in part 2
get_genomic_matrix <- function(snps){
  fa=colSums(snps)/(2*nrow(snps))
  index.non=fa>=1| fa<=0
  snps=snps[,!index.non]
  nSNP=ncol(snps)
  nInd=nrow(snps)
  n=nInd
  ##allele frequency of second allele
  p=colSums(snps)/(2*nInd)
  P=2*(p-.5) #Difference from .5, multiple by 2
  snps=snps-1 #Change from 0/1/2 coding to -1/0/1 coding
  Z=t(snps)-P#operation on matrix and vector goes in direction of column
  #K=tcrossprod((snps), (snps))
  K=crossprod((Z), (Z)) #Thanks to Peng Zheng, Meng Huang and Jiafa Chen for
finding the problem
  adj=2*sum(p*(1-p))
```

```
  K=K/adj
  return(K)
}
```

Now that we have the functions,lets load in the datasets needed and run the models the prediction models

```
#Load in data
marker_matrix <- readRDS("vitA_M.rds")
carotenoids <- readRDS("pheno.rds")
df.CV <- readRDS("cv.rds")
twas <- readRDS("twas.rds")
gwas <- readRDS("owens_gwas_results.rds")

#inputs for BGLR
nIter <- 12000
burnIn <- 8000

#The trait we will be using is beta_Carotene_over_beta_cryptoxanthin
trait <- colnames(carotenoids)[18]
#Get Values
y.all <- as.numeric(carotenoids$beta_Carotene_over_beta_cryptoxanthin)
#Model for singel kernel

#Create ETA for BGLR
ETA.mGRM <- list("G" =list(K= get_genomic_matrix((marker_matrix)), model = 'RKHS'))

pred.mat.mGRM <- myFun.Cv(y.all = y.all, ETA = ETA.mGRM, df.CV = df.CV,
                          nIter = nIter, burnIn = burnIn, verbose = F
                          )
#prediction results
Single_Kernel <- data.frame("Sample_ID" = df.CV[, 1], "Model" = rep("Single Kernel",lengths(pred.mat.mGRM)[1]),pred.mat.mGRM)

#Model for TWAS Kernel
ETA.TWAS <- get_ETA_two_kernel(marker_matrix,get_phenotype_subset(trait, marker_matrix, twas), getAmat = T )
```

```
pred.mat.twas_genes <- myFun.Cv(y.all = y.all, ETA = ETA.TWAS, df.CV = df.CV,
                                      nIter = nIter, burnIn = burnIn,verbose =
F
                                      )
TWAS <- data.frame("Sample_ID" = df.CV[, 1], "Model" = rep("TWAS",lengths(pre
d.mat.mGRM)[1]), pred.mat.twas_genes)


#Model for GWAS Kernel

  ETA.GWAS <- get_ETA_two_kernel(marker_matrix,get_phenotype_subset(trait, ma
rker_matrix, gwas), getAmat = T )

    pred.mat.GWAS <- myFun.Cv(y.all = y.all, ETA = ETA.GWAS, df.CV = df.CV,
                                         nIter = nIter, burnIn = burnIn,verbo
se = F
                                         )

    GWAS <- data.frame("Sample_ID" = df.CV[, 1], "Model" = rep("GWAS",lengths
(pred.mat.mGRM)[1]), pred.mat.GWAS)


#Model for combined GWAS and TWAS Kernel

  ETA.TWAS.GWAS <- get_ETA_two_kernel(marker_matrix,get_phenotype_subset(trai
t,marker_matrix,rbind(twas,gwas)),getAmat = T )

    pred.mat.TWAS.GWAS <- myFun.Cv(y.all = y.all, ETA = ETA.TWAS.GWAS, df.CV
= df.CV,
                                nIter = nIter, burnIn = burnIn,verbose = F)

   TWAS.GWAS <- data.frame("Sample.ID" = df.CV[, 1],  "Model" = rep("TWAS.GWA
S",lengths(pred.mat.mGRM)[1]),pred.mat.TWAS.GWAS)


#Lets take a look at the basic structure of our results

head(TWAS.GWAS)
```

```
##           Sample.ID     Model       Rep01       Rep02     Rep03       Rep0
4
## 2                 2 TWAS.GWAS  0.29270640  0.43671244 0.2328164 0.39232063
9
## 257A9         257A9 TWAS.GWAS  0.35370855  0.16534074 0.3385098 0.24235675
5
## 34f             34f TWAS.GWAS  0.06979688 -0.16668201 0.1217844 0.00688442
7
## 83610b       83610b TWAS.GWAS  0.49694750  0.30342505 0.3731585 0.31722186
7
```

```
## A10579       A10579 TWAS.GWAS  0.36824615  0.42500457 0.3987191 0.45219787
6
## A632_susu A632_susu TWAS.GWAS -0.01485889  0.09667152 0.1105715 0.19817674
9
##                  Rep05       Rep06        Rep07        Rep08       Rep09
## 2           0.29883718  0.41307553  0.244196450  0.267192111  0.21895718
## 257A9       0.35875534  0.25989514  0.276346883  0.213880347  0.29864482
## 34f        -0.19237979  0.06582593 -0.189725892  0.072360503 -0.17980704
## 83610b      0.39002225  0.35334824  0.389908766  0.348952785  0.43569202
## A10579      0.33480883  0.36873091  0.425079355  0.415519187  0.35526952
## A632_susu   0.03250719 -0.08943678  0.006465834 -0.004734734 -0.01518195
##                 Rep10
## 2          0.46528006
## 257A9      0.41048511
## 34f        0.04132985
## 83610b     0.31672282
## A10579     0.36928462
## A632_susu  0.04230054
```

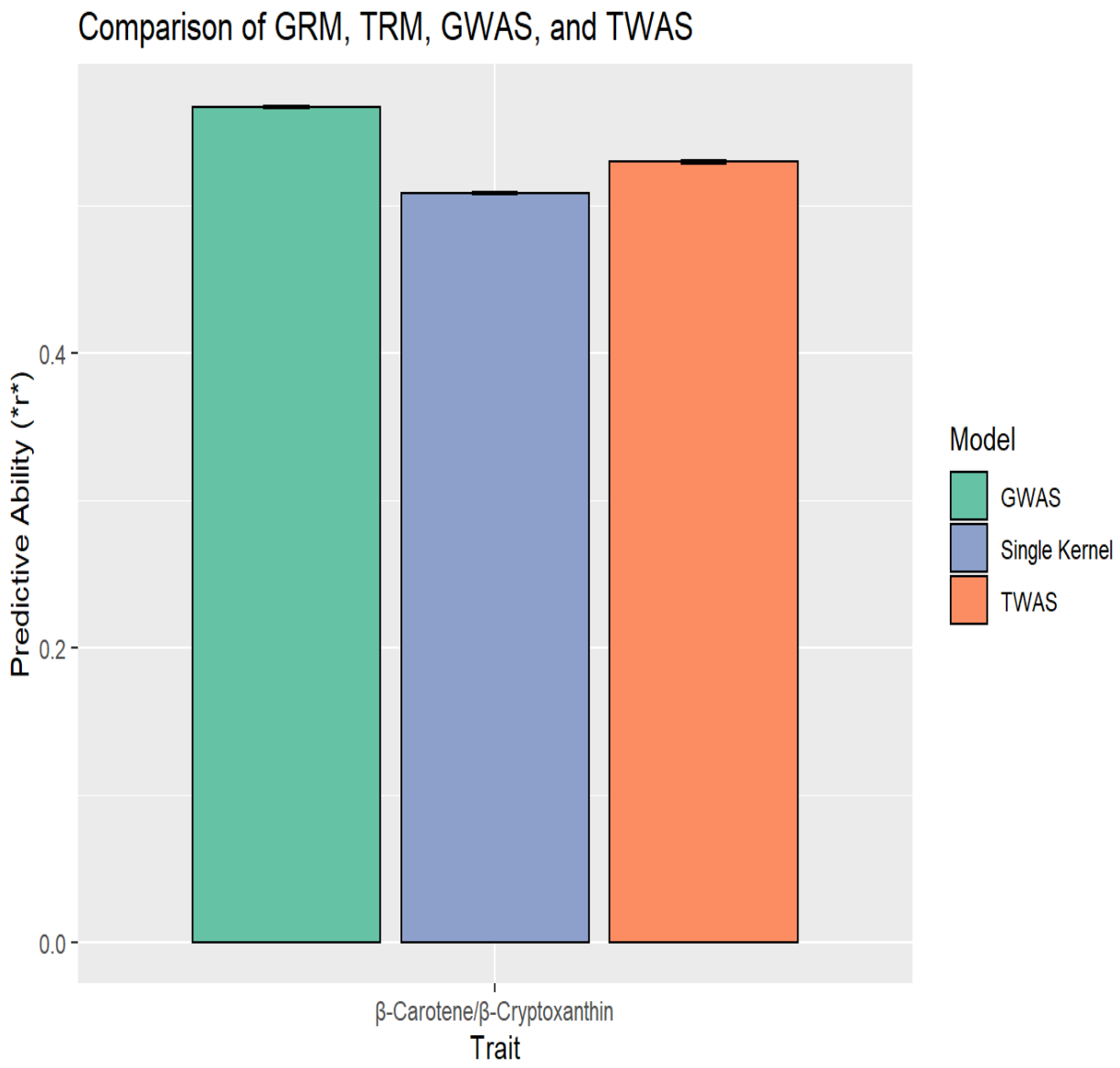
We can see that the end results are the same as before where we have a 308 by 12 matrix where each line has 10 predictions because of the 10 repetitions of the cross-validation. The Sample.ID column gives information about the line and the Model column tells us the type of model used. It is important to have these classifiers to create data visualizations.

The last step is to create a histogram to compare the predictive accuracy of different models.

```
#Function to get 95% Confidence intervals
getCI <- function(df){
  s <- sqrt(var(df))
  n <- length(df)

  return(qt(0.975,df=n-1)*s/sqrt(n))
}

#Format
blups_carot_long <- carotenoids[,c(1,18)] %>%
  pivot_longer(cols = beta_Carotene_over_beta_cryptoxanthin,
               names_to = "Trait", values_to = "Observed")
```

```
#Combine results data frames into one data frame
all_df <- list(Single_Kernel, GWAS, TWAS, TWAS.GWAS)
results <- bind_rows(all_df)

#create a long data frame for ggplot
results <- results %>%   pivot_longer(cols = starts_with("Rep"), names_to = "
Fold", values_to = "Predicted")



#Get Pearsons and spearman correlation for each repetition
results <- results %>% full_join(blups_carot_long, by = c("Sample_ID")) %>%
  drop_na(Predicted, Observed) %>%
  mutate(Predicted = as.numeric(Predicted),
         Observed = as.numeric(Observed)) %>%
  group_by(Trait, Model, Fold) %>%
  mutate(PearsonCor = cor(Predicted, Observed, method = "pearson"),
         SpearmanRankCor = cor(Predicted, Observed, method = "spearman"))



#Change name
accuracy.df <- results %>%
  mutate(Model = as.character(Model)) %>%
  mutate(Trait = as.character(Trait)) %>%
  mutate(Trait = recode_factor(Trait,
                               "beta_Carotene_over_beta_cryptoxanthin" = "\U0
3B2-Carotene/\U03B2-Cryptoxanthin"


  ))

#Get mean value for each model
summarized.accuracy.df <- accuracy.df %>%
  ungroup() %>%
```

```
  dplyr::select(-Fold) %>%
  group_by(  Trait,Model) %>%
  summarize(MeanPearson = mean(PearsonCor),
            MeanSpearman = mean(SpearmanRankCor),
            pearsonCI =  getCI(PearsonCor))


# plot
ggplot(summarized.accuracy.df , aes(x = Trait, y = MeanPearson, fill = Model)
) +
  geom_bar(stat="identity", position = "dodge2", color="black")  +
  geom_errorbar(aes(ymin=MeanPearson-pearsonCI, ymax=MeanPearson+pearsonCI),
          width=.2,      position = position_dodge(0.9)               # Width
of the error bars
  )+
  scale_fill_manual(values=c("#66C2A5", "#8DA0CB", "#FC8D62", "#E78AC3"))+


  labs(y = "Predictive Ability (*r*)")+
  ggtitle("Comparison of GRM, TRM, GWAS, and TWAS")
```



For the one trait Beta-Carotene over Beta-Cryptoxanthin, we see slight improvements when using both a GWAS and TWAS biologically-annotated kernel.

Conclusion

Yay! We made it to the end. We went from the basic of genomic selection to implementing a multiple kernels using BGLR. I hope you learned some useful things that you can carry onward with you for whatever scientific endeavors you take on. As a little note on our final results. While we see only a minimal increase in prediction accuracy using biologically informed multi-kernel models in this tutorial. The actual increase is much greater for certain traits when we use a much larger set of SNPs. The reason I chose a small number of SNPs for this dataset was for the models to run quickly. If you are thinking of doing genomic selection and know of prior information about your crop of choice, I suggest testing whether these types of models will perform better.