Genome to Field Genomic Prediction

January 18, 2020

1. Obtain phenotypic and genetic datasets from G2F resources Since hybrid crop performance has low correlation with inbred per se data and hybrids have more data points, hybrid data are used for modeling.

a. Obtain hybrid and inbred GBS codes

[1]:
```
setwd('C:/Shengqiang/Inari/')
gbsHybridCodes = read.csv('g2f_2017_gbs_hybrid_codes.csv', stringsAsFactor=F)
missingGBSIndex = gbsHybridCodes[,'Female.GBS'] =='#N/A' |␣
↪gbsHybridCodes[,'Male.GBS'] =='#N/A'
gbsHybridCodes = subset(gbsHybridCodes, !missingGBSIndex)
cat(nrow(gbsHybridCodes), 'hybrids with GBS data; ', sum(missingGBSIndex),␣
↪"hybrids are missing GBS data: ",  "\n")
femaleGbsCode  = unique(unlist(gbsHybridCodes[,'Female.GBS']))
maleGbsCode = unique(unlist(gbsHybridCodes[,'Male.GBS']))
inbredGbsCode = unique(c(femaleGbsCode, maleGbsCode))
cat('Inbred GBS codes: ', length(femaleGbsCode), 'Female, ',␣
↪length(maleGbsCode), 'Male, ', length(inbredGbsCode), 'Total')
save(gbsHybridCodes, file='gbsHybridCodes.Rdata')
```

```
2149 hybrids with GBS data;  52 hybrids are missing GBS data:
Inbred GBS codes:  829 Female,  60 Male,  845 Total
```

b. Genotypic data:

1. Read and process imputed inbred GBS data;
2. Then obtain hybrid genotype score by combining female and male inbred GBS data.

[2]:
```
if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
    BiocManager::install(version = "3.10")
}
if (!requireNamespace("rhdf5", quietly = TRUE))
    BiocManager::install(c("rhdf5"))
library(rhdf5)
# Read imputed inbred GBS data from h5 file
genotype <- h5read("g2f_2017_ZeaGBSv27_Imputed_AGPv4.h5",
                   paste("/Genotypes/", inbredGbsCode[1], "/calls", sep=""))

genotypeAll = matrix (NA, length(inbredGbsCode), length(genotype))
```

```
for (iCode in 1:length(inbredGbsCode)){
 genotype <- h5read("g2f_2017_ZeaGBSv27_Imputed_AGPv4.h5",
                    paste('/Genotypes/',inbredGbsCode[iCode], "/calls", sep=''))
 genotypeAll[iCode, ] =  genotype
}
rm(genotype)
# End of read imputed inbred GBS data from h5 file
cat('Total number of GBS markers: ', ncol(genotypeAll), '\n')
cat('Obtained', nrow(genotypeAll), 'inbred lines with GBS markers')
saveRDS(genotypeAll, file='genotypeAllFull.RDS')
```

```
Total number of GBS markers:  945574
Obtained 845 inbred lines with GBS markers
```

A quick look at the number of genotypes at each marker. Most of the markers have only two genotypes, which is expected since those are inbred lines. Around 46% of markers have more than 3 genotypes. This means that there are more than two alleles at those marker position.

[3]:
```
compute_genotype_num = function (a) length(unique(a))
genotypeNum = apply(genotypeAll, 2, compute_genotype_num)
markerStatistic = table(genotypeNum)
t(data.frame(markerStatistic))
cat ('Proportion of markers with more than 3 genotypes: ',␣
 ↪sum(markerStatistic[-c(1:2)])/sum(markerStatistic))
```

genotypeNum	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Freq	366352	139845	207494	178275	31084	14177	5673	1742	806	55	34	16	9	7

```
Proportion of markers with more than 3 genotypes:  0.464667
```

For this analysis, we are going to use markers with only have two genotypes (this means that they are all bi-allelic markers). Based on Meuwissen's rule of thumb, accurate genomic selection will require 10 x Ne x L markers, where Ne is effective population size (should be much less than 845 inbred codes used in this experiement), and L (18 Morgan for maize) is genome length in Morgan. Since Ne should be much less than 845 for this data, we need less than 10 x 845 x 18 = 152,100 markers for genomic prediction. With 366 K bi-allelic markers, we have dense enough markers for genomic prediction.

[4]:
```
# Only keep markers with only two genotypes
markerToKeep = genotypeNum==2
genotypeAll = genotypeAll[, markerToKeep]
saveRDS(genotypeAll, file='genotypeAllBiAllele.RDS')
cat('Total number of GBS markers with just two genotypes: ', ncol(genotypeAll))
```

```
Total number of GBS markers with just two genotypes:  366352
```

Remove markers with less than 0.05 in terms of minor allele frequency.

[5]:
```
compute_minor_allele_freq=function(a) min(table(a)/length(a))
minorAlleleFreq = apply(genotypeAll, 2, compute_minor_allele_freq)
```

```
MAFThreshold = 0.05 # Minor allele frequency threshold
genotypeAll = genotypeAll[, minorAlleleFreq >= MAFThreshold]
saveRDS(genotypeAll, file='genotypeAllProcessed.RDS') # genotypeAll=␣
↪readRDS('genotypeAllProcessed.RDS')
cat('Total number of GBS markers after further filtering by minor allele␣
↪frequency: ', ncol(genotypeAll))
```

```
Total number of GBS markers after further filtering by minor allele frequency:
232631
```

Change genotype table to genotype score. Genotype score for major allle is coded as 0 so that when genotype score matrix is save as sparse matrix, more memory space can be saved.

[6]:
```
minor_allele_genotype = function(a) names(sort(table(a)))[1]
minorAlleleGenotype = apply(genotypeAll, 2, minor_allele_genotype)
genotypeScore = matrix(NA, nrow(genotypeAll), ncol(genotypeAll))
genotypeScore[ t(t(genotypeAll) == minorAlleleGenotype ) ] = 1
genotypeScore[ t(t(genotypeAll) != minorAlleleGenotype ) ] = 0
# Change genotype score matrix to sparse matrix
require(Matrix)
genotypeScore = Matrix(genotypeScore, sparse = TRUE)
rownames(genotypeScore) = inbredGbsCode
saveRDS(genotypeScore, file='genotypeScore.RDS')
genotypeScore[1:5,1:10]
```

```
Loading required package: Matrix
```

```
5 x 10 sparse Matrix of class "dgCMatrix"

PI539921:250031401 . . . 1 1 . 1 1 1 .
PI601170:250032540 . . . . . . 1 1 1 .
LH198:100000467    . . . . . . . . . .
PI537097:250033872 . . . . . . . . . .
LH132:250007440    . . . . . . . . . .
```

Obtain hybrid genotype score data by simply averaging female and male genotype score data.

[7]:
```
# Free up memory space by removing intermediate objects that no longer needed
rm( list = ls()[!ls()%in%c('genotypeScore', 'gbsHybridCodes')])
gc()
hybridGenotypeScore = (genotypeScore[gbsHybridCodes[, 'Female.GBS'], ]  +␣
↪genotypeScore[gbsHybridCodes[, 'Male.GBS'], ]) /2
rownames(hybridGenotypeScore) = gbsHybridCodes$Pedigree
saveRDS(hybridGenotypeScore, file= 'hybridGenotypeScore.RDS')
rm(genotypeScore)
gc()
```

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	1608530	86.0	2482911	132.7	2925215	156.3
Vcells	52541915	400.9	568261744	4335.5	976475057	7450.0

	used	(Mb)	gc trigger	(Mb)	max used	(Mb)
Ncells	1612550	86.2	2482911	132.7	2925215	156.3
Vcells	195983987	1495.3	867085382	6615.4	976475057	7450.0

c. Download clean phenotypic data after outliers are removed from 2014 to 2017

Choose two reprentative traits, yield and moisture, for genomic prediction analysis for the following reasons: 1. Both are important for corn production. 2. One trait (moisture) has high heritability and the other (yield) has relative low heritability.

[8]:
```
dataLink = 'https://de.cyverse.org/anon-files//iplant/home/shared/commons_repo/
 ↪curated/GenomesToFields_2014_2017_v1/'
phen14 = read.csv(file = paste(dataLink, 'G2F_Planting_Season_2014_v4/a.
 ↪_2014_hybrid_phenotypic_data/g2f_2014_hybrid_data_clean.csv', sep=''))
phen15 = read.csv(paste(dataLink,'G2F_Planting_Season_2015_v2/a.
 ↪_2015_hybrid_phenotypic_data/g2f_2015_hybrid_data_clean.csv', sep=''))
phen16 = read.csv(paste(dataLink,'G2F_Planting_Season_2016_v2/a.
 ↪_2016_hybrid_phenotypic_data/g2f_2016_hybrid_data_clean.csv', sep=''))
phen17 = read.csv(paste(dataLink,'G2F_Planting_Season_2017_v1/a.
 ↪_2017_hybrid_phenotypic_data/g2f_2017_hybrid_data_clean.csv', sep=''))
```

Combined phenotypic data into one phenotype object

[9]:
```
colnames(phen14)[1] ='Year'
colnames(phen15)[1] ='Year'
colnames(phen16)[1] ='Year'
colnames(phen17)[1] ='Year'
phenotype = rbind(phen14, phen15, phen16, phen17)
phenotype = subset(phenotype, select=c('Year', 'Field.Location',  'Pedigree',␣
 ↪'Replicate', 'Block', 'Plot', 'Range',     'Rows.Plot',  'Anthesis..date.',␣
 ↪'Silking..date.', 'Pollen.DAP..days.', 'Silk.DAP..days.', 'Plant.Height..cm.
 ↪',
                           'Ear.Height..cm.', 'Stand.Count..plants.', 'Root.
 ↪Lodging..plants.', 'Stalk.Lodging..plants.', 'Grain.Moisture....', 'Test.
 ↪Weight..lbs.bu.' , 'Grain.Yield..bu.A.'))
saveRDS(phenotype, file='phenotype.RDS')
```

2. Partition the dataset to build training and testing sets There are two partitions in separating training and testing sets. 1. The partition is done by leaving out one year data as testing data. For example, training data: 2014, 2015, and 2016 Testing data: 2017. 2. Random sample 20% of hybrid pedigrees from testing data and further remove those pedigrees' phenotypic data in the training years from training data sets

Such partitions allow us to test the models under three different situtations (newEnvOldPed, oldEnvNewPed, and newEnvNewPed) shown in the following table

	training_environment (oldEnv)	testing_environment (newEnv)
pedigrees in training (oldPed)	oldEnvOldPed	newEnvOldPed
pedigrees NOT in training (newPed)	oldEnvNewPed	newEnvNewPed

[10]:
```
traitList = c('Grain.Moisture....', 'Grain.Yield..bu.A.')
yearList = c('2014', '2015', '2016', '2017')
# Year combination (single or three year) for phenotypic BLUP analysis
yearCombination =list(
                      c('2015', '2016', '2017'), # leave out 2014
                      c('2014', '2016', '2017'), # Leave out 2015
                      c('2014', '2015', '2017'), # Leave out 2016
                      c('2014', '2015', '2016'), # leave out 2017
                      c('2014'),
                      c('2015'),
                      c('2016'),
                      c('2017')
                      )
```

Phenotype BLUP are first analyzed with one year or three year of raw phenotypic data

Phenotypic analysis: Pedgrees, locations, blocks are treated as random effects, Blocks are nested within locations, and Year as fixed effect.

[11]:
```
compute_BLUP  = function(phenotype, trait, yearList){
    # Input phenotype, trait to analysis, which years data to analysis
    # Ouput BLUP result for pedigrees
    require(lme4)
    phenotype = subset(phenotype, Year%in%yearList)
    phenotype$TRAIT = phenotype[, trait]
    phenotype$Pedigree = as.factor(as.vector(phenotype$Pedigree))
    phenotype$Field.Location = as.factor(as.vector(phenotype$Field.Location))
    phenotype$Block = as.factor(as.vector(phenotype$Block))
    if (length(unique(phenotype$Year))==1) {
        model  = lmer(TRAIT ~ 1 + (1|Pedigree) + (1|Field.Location/Block),
                      data = subset(phenotype, !is.na(TRAIT))) # Block nested␣
 ↪within location
    } else {
        phenotype$Year = as.factor(as.vector(phenotype$Year))
        model  =  lmer(TRAIT ~ 1 +  (1|Pedigree) + (Year|Field.Location/Block),␣
 ↪data = subset(phenotype, !is.na(TRAIT))) # Block nested within location     ␣
 ↪
    }
    #
   return (ranef(model)$Pedigree)
}

blupAll = NULL
```

```
for (iTrait in traitList){
    for (iYear in 1:length(yearCombination)){
       blup = compute_BLUP(phenotype, iTrait, yearCombination[[iYear]])
       blup = data.frame(Pedigree = rownames(blup), Trait =iTrait, Value =␣
 ↪blup[, 1], Year_combination= paste(yearCombination[[iYear]], collapse=" "))
       blup$Pedigree = as.vector(blup$Pedigree)
       blupAll = rbind(blupAll, blup)
    }
}

saveRDS(blupAll, file='blupAll.RDS')
```

```
Loading required package: lme4
Warning message:
"package 'lme4' was built under R version 3.6.2"
```

[12]:
```
# Training and testing combination for genomic prediction
trainingTestingCombination= data.frame(Train =c(
                           '2015 2016 2017',
                           '2014 2016 2017',
                           '2014 2015 2017',
                           '2014 2015 2016'),
                                  Test = c('2014', '2015', '2016', '2017')
                                 )

trainTestTrait = NULL
for (iYear in c('2014', '2015', '2016', '2017')){
    for (iTrait in traitList){
        tmp= subset(trainingTestingCombination, Test==iYear)
        tmp$Trait = iTrait
        trainTestTrait = rbind(trainTestTrait, tmp)
    }
}
print('Training, testing data sets, and trait combination for modeling')
trainTestTrait
```

```
[1] "Training, testing data sets, and trait combination for modeling"
```

	Train	Test	Trait
1	2015 2016 2017	2014	Grain.Moisture....
2	2015 2016 2017	2014	Grain.Yield..bu.A.
21	2014 2016 2017	2015	Grain.Moisture....
22	2014 2016 2017	2015	Grain.Yield..bu.A.
3	2014 2015 2017	2016	Grain.Moisture....
31	2014 2015 2017	2016	Grain.Yield..bu.A.
4	2014 2015 2016	2017	Grain.Moisture....
41	2014 2015 2016	2017	Grain.Yield..bu.A.

[13]:
```
remove_partial_line_in_testingdata_from_training = function(testingYear, trait){
    # Input testing year
    # Output one fifth pedigrees that are from testing year. These pedigrees␣
 ↪will be removed from training data
    set.seed(123) # set the same seed so that we can get back the same splitted␣
 ↪samples for training and testing
    pedigree = subset(blupAll, Year_combination == testingYear & Trait ==␣
 ↪trait)$Pedigree
    pedigree = sample(pedigree, ceiling(0.2*length(pedigree)) , replace=FALSE)
    pedigree = pedigree [pedigree%in%gbsHybridCodes$Pedigree]
    return(pedigree)
}

# Obtain the list of pedigree to be removed from each training and testing␣
 ↪combinations
lineRemoveFromTraining = list()
for (iYear in yearList){
    lineRemoveFromTraining[[iYear]] = list()
    for (iTrait in traitList){
        lineRemoveFromTraining[[iYear]][[iTrait]] =␣
 ↪remove_partial_line_in_testingdata_from_training (iYear, iTrait)
    }
}

training_phenotype_preparation = function(trainRow) {
  tmp = trainTestTrait[trainRow, ]
  blup = subset(blupAll, Year_combination == as.vector(tmp$Train) & Trait== as.
 ↪vector(tmp$Trait))
  blup = subset(blup, !
 ↪Pedigree%in%lineRemoveFromTraining[[tmp$Test]][[tmp$Trait]])
  blup = subset(blup, Pedigree%in%gbsHybridCodes$Pedigree)
  return (blup)
}
```

3. Genomic prediction models:

a. Penalized linear regression (glmnet) Phenotype is model as the linear summmation of marker effects. During training, marker coefficient were constrained between lasso and ridge regression penalty.

b. Kernel regression:
   1. G-BLUP: a simple linear kernel (marker relationship matrix) from markers
   2. Gaussian kernel: a Euclidean distance based Gaussian kernel, the model can potentially model epistasis

[14]:
```
generate_five_fold_cross_validation_list=function(y, foldNo=5){
  # Input sample number, and fold number of cross-valiation.
  # Default is five-fold cross-validation during training to pick the best␣
↪parameters for the model
  # Output sample number list for cross-validation
  randomList = sample(1:length(y), replace=F)
  crossValidationList =rep(list(NULL),foldNo)
  for (i in 1:foldNo){
      temp=rep(FALSE, 5)
      temp[i] =TRUE
      crossValidationList[[i]] = randomList[temp]
  }
  return (crossValidationList)
}
```

Penalized linear regression function using glmnet package

[15]:
```
cross_validation_glmnet=function(x, y, foldNo=5){
  # Input: genotype data x for training, phenotype data y for training
  # Ouput: prediction for all hybrid with genotype data
  # Approach: Cross-validation approach with training data is first used to␣
↪find the best parameters.
  # Then model is built with all training data with the best parameters for␣
↪prediction.
  require(glmnet)
  alphaSeq=c(0.01, 0.1, 0.2*(1:5))
  lambdaSeq=c(seq(0.01,1, 0.01),1:100)
  alphaLambdaCorrelationNet = NULL
  crossValidationList = generate_five_fold_cross_validation_list(y, foldNo)

  for (iAlpha in alphaSeq){
    lambdaCorrelation=0
    for (j in 1:foldNo){  # j=1 ; # Five-fold nested cross-validation.
     validationRandom = crossValidationList[[j]]
     fit=glmnet(x[-validationRandom,],y[-validationRandom], alpha=iAlpha) #␣
↪fit=glmnet(trainGeno,trainPheno, alpha=1) ;␣
↪fit2=glmnet(trainGeno,trainPheno, alpha=0.5)
     predictFit=predict(fit,newx=x[validationRandom,], s=lambdaSeq)
     options(warn=-1) # Ignore warnings when correlation is not available
     tempCorrel=cor(predictFit, y[validationRandom], use="complete")
     options(warn=0) # Turn warning back
     tempCorrel[is.na(tempCorrel)] = -1
     lambdaCorrelation=lambdaCorrelation+tempCorrel
    }
    lambdaCorrelation=lambdaCorrelation/5
    alphaLambdaCorrelationNet=cbind(alphaLambdaCorrelationNet,lambdaCorrelation)
  }
```

```
    alphaMax=matrix(rep(alphaSeq, length(lambdaSeq)),length(lambdaSeq),byrow=T)
    lambdaMax=matrix(rep(lambdaSeq,␣
↪length(alphaSeq)),,length(lambdaSeq),byrow=F)

    alphaMax=alphaMax[alphaLambdaCorrelationNet==max(alphaLambdaCorrelationNet)]
  ␣
↪lambdaMax=lambdaMax[alphaLambdaCorrelationNet==max(alphaLambdaCorrelationNet)]
    # Retrain with whole data set
    fit=glmnet(x,y,alpha=alphaMax)
    # Generate prediction for all hybrids with genotype data
    predictFit=predict(fit,newx=hybridGenotypeScore, s=lambdaMax)
    return(predictFit)
}
```

Kernel regression using mixed effect model from rrBLUP package

[16]:
```
kernel_predict = function(kernelMatrix, blup){
    # Input kernel matrix
    # Ouput prediction for all breeding values
    blup2 = rep(NA, length(gbsHybridCodes$Pedigree))
    names(blup2) = gbsHybridCodes$Pedigree
    blup2[as.vector(blup$Pedigree)] = blup[, 'Value']
    require(rrBLUP)
    ans <- mixed.solve(blup2,K = kernelMatrix)
    predict = ans$u
    names(predict) = gbsHybridCodes$Pedigree
    return(predict)
}
```

[17]:
```
train_predict_to_list = function(blup, predict){
  # Combine phenotypic data in training and predition in one list
  trainPredict = list()
  trainPredict[['train_blup']] = blup
  trainPredict[['predict']] = predict
  return(trainPredict)
}
```

Genomic prediction with three algorithms

Python instead of R is used to build the kernel matrix for regression. My PC doesn't have enough memory to build kernel matrix in R with large number of markers. Please see the attached python notebook for detail.

[18]:
```
glmnetPredictAll = list()
GBLUPPredictAll = list()
euKernelPredictAll =list()
for (iRow in 1:nrow(trainTestTrait)){
```

```
  # Obtain phenotypic data for training
  blup =training_phenotype_preparation (iRow)

  # Glmnet modeling
  predict = cross_validation_glmnet(hybridGenotypeScore[as.
↪vector(blup$Pedigree), ], blup[,"Value"], foldNo=5)
  glmnetPredictAll[[iRow]] = train_predict_to_list(blup, predict)

  # G-BLUP modeling
  if ('kernelMatrix'%in% ls()) rm(kernelMatrix)
  load('kernelMatrix_G.gzip') # Linear kernel matrix
  predict = kernel_predict(kernelMatrix, blup)
  GBLUPPredictAll[[iRow]] = train_predict_to_list(blup, predict)

  # Euclidean distance based Gaussian kernel regression modeling
  if ('kernelMatrix'%in% ls()) rm(kernelMatrix)
  load('kernelMatrix_eu_dist.gzip') # Euclidean distance based Gaussian kernel␣
↪matrix
  predict = kernel_predict(kernelMatrix, blup)
  euKernelPredictAll[[iRow]] = train_predict_to_list(blup, predict)
}
save(glmnetPredictAll, GBLUPPredictAll, euKernelPredictAll, file␣
↪='genomePredictionResult.Rdata')
```

Compute predictive correlation and overlapping porportion from selection

[19]:
```
compute_correlation_overlap_result=function(trainRow, predictionResult){ #␣
↪trainRow =1;
    prediction = predictionResult[[trainRow]]
    temp = trainTestTrait[trainRow, ] # traing, test year, and trait
    # All BLUP data in training year combination
    blupTrain = subset(blupAll, Year_combination%in%temp$Train & Trait␣
↪==temp$Trait)
    # BLUP data in testing year
    blupTest = subset(blupAll, Year_combination%in%temp$Test & Trait␣
↪==temp$Trait)

    blup_OldEnvNewPedigree = subset(blupTrain, !(Pedigree%in%␣
↪prediction$train_blup$Pedigree))
    blup_OldEnvOldPedigree = prediction$train_blup
    blup_NewEnvNewPedigree = subset(blupTest, !(Pedigree%in%␣
↪prediction$train_blup$Pedigree))
    blup_NewEnvOldPedigree = subset(blupTest, (Pedigree%in%␣
↪prediction$train_blup$Pedigree))

    # Rank the value and obtain top certain proportion of pedigrees
    get_top_pedigree=function(data, valueColumn, trait, topRate = 0.3){
```

```
       if (trait =='Grain.Yield..bu.A.')   data = data[order(-data[,␣
↪valueColumn]),]
       if (trait =='Grain.Moisture....')   data = data[order(data[,␣
↪valueColumn]),]
       data$RANK = (1:nrow(data))/nrow(data)
       topPedigree = data[data$RANK <=topRate, ]$Pedigree
       return (topPedigree)
    }

    obs_pred_compare =function(observe, predict){ # predict =␣
↪prediction$predict; observe = blup_OldEnvNewPedigree
         # Input observation and prediction
         # Ouput the their correlation and overlapping proportion
         predict = prediction$predict
         if (length(dim(prediction$predict)) ==2) predict=predict[,1]
         predict= data.frame(Pedigree=names(predict), predict=predict)
         obsPredict = merge(observe, predict, by='Pedigree', all.x=T)
         obsPredict = subset(obsPredict, (!is.na(Value)) &  (!is.na(predict)))
         correlation = with(obsPredict, cor(Value, predict, use='complete'))
         # Compute overlapping proportion of pedigrees
         topObs = get_top_pedigree(obsPredict, "Value", temp$Trait)
         topPredict =  get_top_pedigree(obsPredict, "predict", temp$Trait)
         overlapProportion = length(intersect(topObs, topPredict))/
↪length(topObs)
         return( round(c(correlation, overlapProportion),3))
    }
    oldEnv_OldPed_training = obs_pred_compare(blup_OldEnvOldPedigree,␣
↪prediction$predict)
    oldEnv_NewPed_test = obs_pred_compare(blup_OldEnvNewPedigree,␣
↪prediction$predict)
    newEnv_OldPed_test = obs_pred_compare(blup_NewEnvOldPedigree,␣
↪prediction$predict)
    newEnv_NewPed_test = obs_pred_compare(blup_NewEnvNewPedigree,␣
↪prediction$predict)

    correlationOverlapResult = data.frame(train_year =temp$Train, test_year =␣
↪temp$Test, trait = temp$Trait,
       oldEnvOldPed_cor = oldEnv_OldPed_training[1],
       oldEnvNewPed_cor = oldEnv_NewPed_test[1] ,
       newEnvOldPed_cor = newEnv_OldPed_test[1],
       newEnvNewPed_cor = newEnv_NewPed_test[1],
       oldEnvOldPed_overlap = oldEnv_OldPed_training[2],
       oldEnvNewPed_overlap =oldEnv_NewPed_test[2]   ,
       newEnvOldPed_overlap = newEnv_OldPed_test[2] ,
        newEnvNewPed_overlap = newEnv_NewPed_test[2]
    )
```

```
  return (correlationOverlapResult)
}
```

[20]:
```
GBLUPResult = NULL
glmnetResult = NULL
euKernelResult = NULL
for (i in 1:8){
  GBLUPResult =rbind(GBLUPResult, compute_correlation_overlap_result(i,␣
 ↪GBLUPPredictAll))
  glmnetResult =rbind(glmnetResult, compute_correlation_overlap_result(i,␣
 ↪glmnetPredictAll))
  euKernelResult =rbind(euKernelResult, compute_correlation_overlap_result(i,␣
 ↪euKernelPredictAll))
}
```

[21]:
```
result_summary=function(predictionResult, statistics ='cor', traitName = "",␣
 ↪method=""){
    if (traitName!="") predictionResult = subset(predictionResult, trait ==␣
 ↪traitName)
    if (statistics =='cor') {
        resultCol = -grep('_overlap', colnames(glmnetResult))
     } else {
      resultCol = -grep('_cor', colnames(glmnetResult))
     }
    predictionResult = predictionResult[, resultCol]
    colnames(predictionResult) = gsub(paste('_', statistics, sep=''), '',␣
 ↪colnames(predictionResult))
    predictionResult = predictionResult[order(predictionResult$trait), ]

    average = round(apply(predictionResult[, -c(1:3)] , 2, mean),3)
    average['train_year'] =''
    average['test_year'] ='Average'
    average['trait'] = traitName
    average = average[colnames(predictionResult)]
    average= data.frame(t(average))
    result = rbind(predictionResult, average)
    if (method !='') result =data.frame(method, result)
    return(result)
}
summary_by_method = function(statistics){
 # Combine results from different methods and traits
 resultAll =NULL
 for (iTrait in traitList){
  resultTrait=rbind(result_summary(glmnetResult, statistics, iTrait, 'glmnet'),
  result_summary(GBLUPResult, statistics, iTrait, 'G-BLUP'),
  result_summary(euKernelResult, statistics, iTrait, 'Gaussian Kernel'))
```

```
   resultAll=rbind( resultAll,  resultTrait)
  }
  return(resultAll)
}

prediciveAbilityResult =  summary_by_method('cor')
selectionReliabilityResult = summary_by_method( 'overlap')
```

4. Model comparison The following table shows the predictive correlation for different models, averaged from four training-and-testing-year combinations.

For yield, Gaussian kernel regression gives the highest prediction accuracy under all the following three situations 1. predict new pedigree in new environment, newEnvNewPed (21% better than G-BLUP and glmnet) 2. predict old pedigree in new envionment, newEnvoldPed (14% better than G-BLUP and glmnet) 3. predict new pedigree in old environment, oldEnvNewPed (14% better than G-BLUP and glmnet)

For moisture, the difference among three methods are small.

Overall, glmnet and G-BLUP have very similar prediction accuracy for both yield and moisture

[22]:
```
# Averaged predictive correlation across 4 years
predictionAbilityAverage = subset(prediciveAbilityResult, test_year␣
 ↪=='Average', select=-(oldEnvOldPed))
predictionAbilityAverage [, -grep('year', colnames(predictionAbilityAverage)) ]
```

	method	trait	oldEnvNewPed	newEnvOldPed	newEnvNewPed
11	glmnet	Grain.Moisture....	0.885	0.876	0.817
111	G-BLUP	Grain.Moisture....	0.884	0.876	0.814
112	Gaussian Kernel	Grain.Moisture....	0.887	0.872	0.806
14	glmnet	Grain.Yield..bu.A.	0.621	0.627	0.388
113	G-BLUP	Grain.Yield..bu.A.	0.62	0.643	0.396
121	Gaussian Kernel	Grain.Yield..bu.A.	0.706	0.733	0.481

The following table shows the proportion of common entries between observed and predicted phenotype at top 30% selection intensity with different models, averaged from four training-and-testing-year combinations.

For yield, Gaussian kernel regression gives the highest overlapping proportion under all three situations: 1. predict new pedigree in new environment, newEnvNewPed 2. predict old pedigree in new envionment, newEnvoldPed
3. predict new pedigree in old environment, oldEnvNewPed

For moisture, the difference among three methods are small.

[23]:
```
# Average overlapping proportion at top 30% selection
selectionAbilityAverage = subset(selectionReliabilityResult,␣
 ↪test_year=='Average', select=-(oldEnvOldPed))
selectionAbilityAverage [, -grep('year', colnames(selectionAbilityAverage)) ]
```

	method	trait	oldEnvNewPed	newEnvOldPed	newEnvNewPed
11	glmnet	Grain.Moisture....	0.772	0.8	0.783
111	G-BLUP	Grain.Moisture....	0.74	0.797	0.786
112	Gaussian Kernel	Grain.Moisture....	0.743	0.802	0.765
14	glmnet	Grain.Yield..bu.A.	0.577	0.548	0.501
113	G-BLUP	Grain.Yield..bu.A.	0.59	0.563	0.497
121	Gaussian Kernel	Grain.Yield..bu.A.	0.663	0.612	0.542

5. Conclusion: Among three algorithms, I would recommend Gaussian kernel regression approach over G-BLUP and glmnet, given its highest prediction accuracy and overlapping selection proportion for yield.

Kernel Matrix

January 18, 2020

Compute kernel matrix based on genotype score.

1. G-Matrix (Linear kernel)
2. Euclidean distance kernel

Import genotype data (a R object) to python

[1]:
```
import numpy as np
import rpy2.robjects as robjects
from rpy2.robjects import r
from rpy2.robjects.numpy2ri import numpy2ri

robjects.r('''
        load('C:/Shengqiang/Inari/hybridGenotypeScore.Rdata')
        ''')
hybridGenotypeScore = robjects.globalenv['hybridGenotypeScore']
np.shape(hybridGenotypeScore)
```

[1]:
```
(2149, 232631)
```

Normalize the genotype data for each marker

[2]:
```
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.preprocessing import scale
scaledGenotype = scale(hybridGenotypeScore, axis=0, with_mean=True,␣
↪with_std=True, copy=True)
scaledGenotype[0:5, 0:5]
```

[2]:
```
array([[-0.60080131, -0.28266499, -0.25992471,  2.07422975,  2.07422975],
       [-0.60080131, -0.28266499, -0.25992471, -0.43140248, -0.43140248],
       [-0.60080131, -0.28266499, -0.25992471, -0.43140248, -0.43140248],
       [-0.60080131, -0.28266499, -0.25992471, -0.43140248, -0.43140248],
       [-0.60080131, -0.28266499, -0.25992471, -0.43140248, -0.43140248]])
```

Compute linear kernel (G-Matrix) and save it as a R object

[3]:
```
p = scaledGenotype.shape[1] # Number of markers
linearKernel = np.dot(scaledGenotype,scaledGenotype.T)/p
linearKernel = np.array(linearKernel, dtype="float64") # <- convert to double␣
↪precision numeric since R doesn't have unsigned ints
```

```
ro = numpy2ri(linearKernel)
r.assign("kernelMatrix", ro)
r("save(kernelMatrix, file='C:/Shengqiang/Inari/kernelMatrix_G.gzip',␣
 ↪compress=TRUE)")
```

[3]: rpy2.rinterface.NULL

Compute Euclidean distance kernel and save it as a R object

[4]:
```
from sklearn.metrics.pairwise import euclidean_distances
euclideanDistKernel = ((euclidean_distances(scaledGenotype))**2) /p
h = 0.5
euclideanDistKernel = np.exp( - h*euclideanDistKernel)
euclideanDistKernel = np.array(euclideanDistKernel, dtype="float64") # <-␣
 ↪convert to double precision numeric since R doesn't have unsigned ints
ro = numpy2ri(euclideanDistKernel)
r.assign("kernelMatrix", ro)
r("save(kernelMatrix, file='C:/Shengqiang/Inari/kernelMatrix_eu_dist.gzip',␣
 ↪compress=TRUE)")
```

[4]: rpy2.rinterface.NULL