Report Two: Extreme Computing
Project-based High Performance Distributed and Parallel Systems
APCOMP 290R

Group CPU

November 12, 2015

Partners:	Xiaowen CHANG
	Feifei PENG
	Zelong QIU
	Qing ZHAO
Instructor:	Sean Davis

1 Machine Learning Algorithms on the NCI-60 Drug Data

1.1 Background

We would like to identify pharmacogenomic correlations between compounds and associated drug screen data. In the experiment, each cell line has been treated with many different compounds. Our objective is to use the filtered variants to predict drug GI50.

The challenge here is that we have 1225 variants (predictors or features) while we only have 61 responses for each drug. As a result, feature selection becomes the key of conduct analysis for this problem.

1.2 Original model (Random Forest Regression)

The original model used Random Forest Algorithm to do feature selection and then do a linear regression to observe the association between compounds and the drug. Since we have 181 drugs, the origin model conducted 181 linear regression models and among which find those models whose variance explained is greater than 0.7. Among all models, there are two with variance explained greater than 0.7: "NSC1390" and "NSC755985".

The advantage of Random Forest Algorithm is that it has conducted feature selection in the algorithm, and the features it selected are the compounds we have thus are easy to be interpreted. It also has some limitations. Since there are too many features, even after feature selection, we still have many features. So when conducting linear regression, it is easy to get in the problem of overfitting. As a result, we will do feature selection using different method but continue doing linear regression after feature selection.

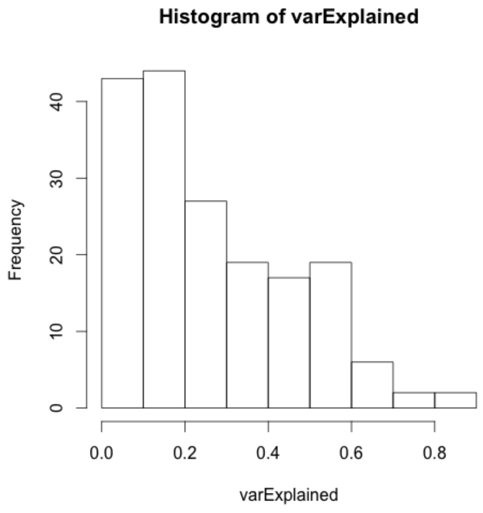

Figure 1: Random Forest: Histogram of varExplained

We can see from the histogram that the distribution is right-skewed with most values small. This is to say, Random Forest Algorithm is not accurate enough. This also explains why we only got two variance explained greater than 0.7.

1.3 Stepwise regression: Backward Elimination Using AIC

Backwards elimination involves starting with all candidate variables, testing the deletion of each variable using a chosen model comparison criterion, deleting the variables according to the criterion. Here we choose to use AIC (Akaike Information Criterion).

The Backward Elimination starts with full model. The problem with this algorithm is there are 1225 predictors, which is far more than the number of responses. As a result, it is hard to conduct regression. The way we solve this problem is by looking at the correlation matrix of the predictors (compounds) and deleting the features with high correlations. Finally we got 30 predictors, with which we will use to conduct stepwise regression.

This algorithm finally gave us 7 models whose variance explained greater than 0.7.

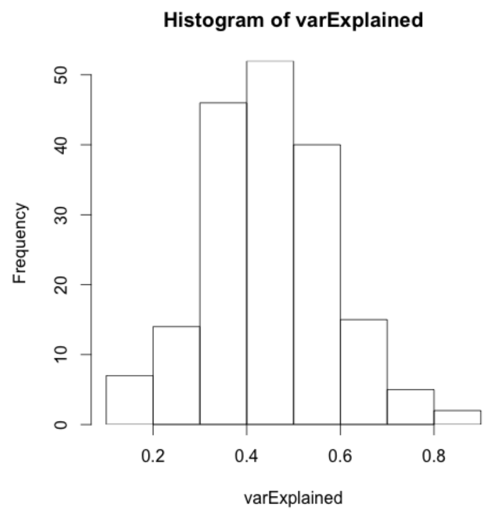

Figure 2: Stepwise Regression (Highly Correlated Predictors Removed): Histogram of varExplained

From the histogram above, we can see that most of the distribution fall between 0.3 and 0.6. Also the distribution is more centered than the previous one. Accuracy of the model has been greatly improved. However, the 7 models we got does not include the 2 models given by Random Forest Algorithm previously. The two models give us totally different results, this may due to the fact that too many features were filtered out according to correlation matrix. Using correlation matrix to filter features is a highly robust method and may leads to some issues that need to deal with later.

One way to improve is that instead of using Backwards Elimination, we use Forward Selection which start with no predictors in the model and then add in the predictor (if any) that improves the model the most. However, the drawback is that each addition of a new variable may render one or more of the already included variables non-significant. Another way to improve is that instead of using correlation matrix to filter features, we use some fine-grained method to select the features and then keep the Backwards Elimination method to do stepwise linear regression. This leads to what we are going to do next.

1.4 Principal Component Regression (PCR)

In PCR, instead of regressing the dependent variable on the explanatory variables directly (in our case, compound), the principal components of the compounds are used as regressors. Instead of using correlation matrix, we do Principal Component Analysis first to get top p principal components and use these principal components as regressors to stepwise regression for each drug.

The first issue to address is how many principal components should we get. A popular way to do this is to look at scree plot, which visualizes the Eigen values of each component. The scree plot is below:

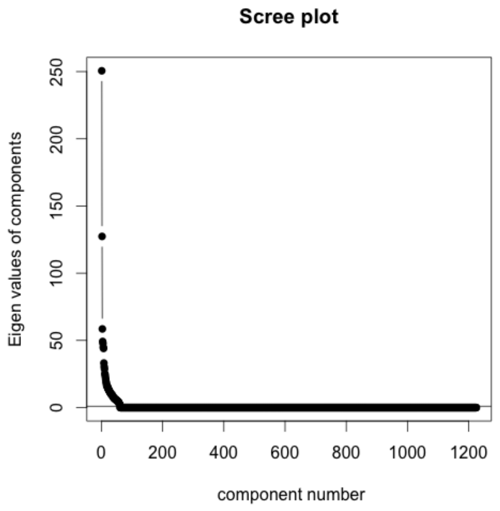

Figure 3: Scree Plot

One popular way is to choose components with Eigen value greater than 1. However, we have more than 60 components whose Eigen value greater than 1, which is greater than number of responses (61 in this case). Another popular way is to retain those components in the steep curve before the first point that starts the flat line trend. Our plot gets its turning point at around 10, so we pick up components with Eigen value greater than 10. This gives us 30 components. So we continue to conduct Backward Elimination stepwise regression with these 30 components. The following plot is the histogram of the variance explained of our output models.

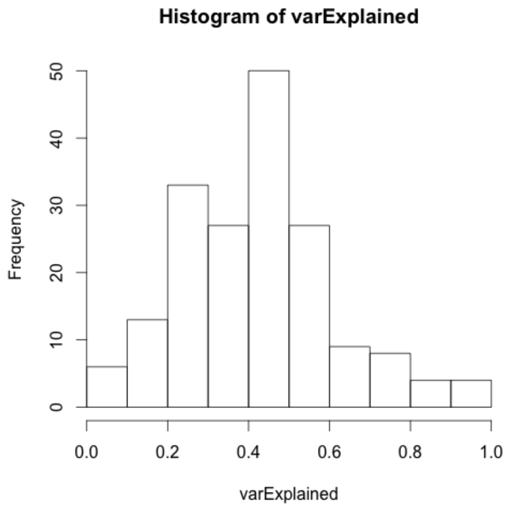

Figure 4: PCA Variance Explained

After doing stepwise regression, we get 16 models whose variance explained is greater than 0.7. Also, these 16 models includes the 2 models given by Random Forest Regression and some of the models given by Stepwise Regression with highly correlated predictors removed. From the plot and the models got, this algorithm is the most comprehensive and accurate method. The drawback of this algorithms is that the models we got are difficult to interpret. The reason is that the principal components used as predictors here are the linear combinations of the original features (compounds). And also the coefficients of the linear combination can be either positive or negative. This makes it hard for use to interpret the association between drugs and compounds. But this algorithm does an excellent job in drug prediction. For the future study, we can improve by using Nonnegative Matrix Factorization (NMF) instead of PCA to make our model more interpretable.

2 Complexity of the Central Dogma

The Central Dogma of Biology describe a process of how genetic information, DNA which contain the information to make all proteins, and RNA which is the messenger carrying the information, to make proteins which are the final product. The process of converting DNA instructions into proteins is called gene expression. The gene expression have three levels and two steps which are usually described as “DNA makes RNA and RNA makes protein”.
In living organisms, there are three major classes of sequential information-carrying biopolymers, namely DNA, RNA and protein. There are nine kinds of transfers between every pair of these three biopolymers and the Central Dogma categorize these nine transfers into three groups, general transfers, special transfers and unknown transfers. The general transfers are the transfers that most likely to occur in most normal cells and these transfers describe the flow of information from DNA to protein. The “DNA makes RNA” part describe DNA information could be copied into message RNA and the process called transcription. The “RNA makes protein” describe RNA synthesis protein which is called translation. Also, DNA could be copied to DNA which is called replication. There are a lot of complexity regarding there three level in the Central Dogma of biology.
DNA replication happens before a cell divide into two. The replication process allow each child cell to contain a mutual complement sequence of chromosome and thus results in two identical DNA sequences. There are at least two kind of complexity involved in the DNA replication process, one called origin recognition complexity and another called pre-replication complexity. The origin recognition complexity is the complexity to identify the first particular point at which to initiate the replication process. The pre-replication complexity involved the actually replication process. Since the DNA is double-stranded structure, it involves complex precess to unzip the double-stranded DNA into two single parts.
DNA transcription process happened we the information contained in DNA is copied in the form of mRNA. During the transcription process, a DNA sequence is read by an RNA and then produce a completely opposite RNA single-stranded sequence. The generated RNA sequence would then served as a template for the further protein generation. The second level of complexity in Central Dogma is involved the whole DNA transcription process.
The third level of complexity involved the third step of Central Dogma of biology which is translation. The messenger RNA, the one produced from DNA transcription, would be decoded during the translation process. The ribosome reads the sequential information carried by the mRNA and generate complete opposite tRNA anticodon, which carries the specific animo acid and chain them together to form the specific protein implied.

3 Extreme Biocomputing

In this section, we are going to investigate on the topic of Extreme Biocomputing. Specifically, we will discuss techniques from three different perspectives: algorithm, cloud parallel computing, and hardware Accelerators (GPUs) to leverage modern technologies to deal with extremely high volumns of genomics data.

3.1 Pattern recognition in high-throughput genes/proteins data

In recent years, large amounts of high dimensional data have been generated from high-throughput expression techniques, such as gene expression data using microarray or deep sequencing, and metabolomics and proteomics data using Liquid Chromatography ? Mass Spectrometry (LC-MS). Mining the hidden patterns inside these data leads to an enhanced understanding of functional genomics, gene regulatory networks, etc. However, the complexity of biological networks and the huge number of genes pose great challenges to analyze the big mass of data.
Clustering techniques has usually been applied as a first step in the data mining process to analyze hidden structures and reveal interesting patterns in the data. One way of better modelling nonlinear relationships in high dimensions with less computationally expensive cost is a new nonlinear clustering method named K-profiles clustering. Researchers incorporated statistical inference into the algorithm to remove the impact of noise genes due to their common existence in real world microarray data. The algorithm is efficient due to the quality of the Distance based on Conditional Ordered List (DCOL). The algorithm outperformed our previous General Dependency Hierarchical Clustering (GDHC) algorithm and the traditional k-means clustering algorithm in our simulation studies. It generated meaningful results in real data analysis. It can be used in the analysis of high-throughput data to detect novel patterns based on nonlinear dependencies.
Compared to similar methods such as GDHC which is very computation intensive, the new K-profile clustering is much more efficient. Another key advantage of the K-profile clustering method is that by building statistical

inference into the iterations, noise genes that don?t belong to any cluster will not interfere with the cluster profile estimation, and they are naturally left out of the final results.

3.2 Genome Analysis in a Dynamically Scaled Hybrid Cloud

Nowadays biological laboratories often purchase large high-performance computing (HPC) facilities, or spend significant proportions of their budget hiring compute resources from an academic or public cloud provider. However, in neither case is their purchasing as efficient as possible: it is common for HPC facilities to spend large proportions of their time idle, or for external cloud purchasing to be sized in an ad-hoc manner, either manually or with minimal automation. Applying intelligent algorithms to determine what cloud resources to hire, and how to distribute resources between different analysis jobs, has the potential to improve resource usage efficiency and thus represent a significant saving to biological laboratories and institutions. Researchers explore the benefits of automatically determining the degree of parallelism used to perform genetic mutation calling in a hybrid cloud environment. The proposed algorithms can automatically control both the hiring of hybrid cloud resources and the selection of the degree of parallelism employed in analysis tasks executed against that cloud. Using the Broad Institute?s Genome Analysis Toolkit as a case study, reserchers then conduct profile-driven simulation studies to characterise the circumstances in which the algorithms are beneficial or deleterious compared to simple, conventional baseline algorithms. The finding is there are a wide range of cloud workload scenarios where the algorithms outperform the baselines, and thereby argue that automatic control of cloud scaling and task parallelism, using techniques like those proposed, are likely to be beneficially applicable to real-world biocomputing.

3.3 Hardware Accelerators

GPU and CBE provide multi-core platforms where applications with sufficient scope for parallelization can be ported. From the programmability perspective, GPUs are easier for implementation than CBEs, which require the user to write elaborate synchronization software. Furthermore, due to the inherent architectural properties of a GPU, it achieves comparable or higher speedups than a CBE-based platform for compute-intensive applications. On the flip side, the GPU-based solution spends a significant fraction of its time in CPU-GPU communication through PCI-X. General purpose multi-core platforms provide the best overall speedup and also provide the maximum ease of porting of code, while the effort required for CBE is maximum. Due to their reconfigurable nature, FPGA platforms provide the scope for implementing parallel architectures specifically optimized to solve SA or phylogenetic inference. The advantage of NoC-based multi-core architectures is that both computation and intrachip communication can be tailor-made for the application, providing orders of magnitude performance improvement. The field of bioinformatics and computational biology is host to applications that combine a number of desirable attributes ? emerging importance, high computational complexity, inherent data-parallelism, and well-defined communication and computation patterns ? soliciting the design, development and application of novel hardware accelerators. While all the hardware models studied so far have shown significant levels of improvement over single.

```
# We can grab the data directly from github.
#tmp <- read.csv("https://raw.githubusercontent.com/seandavi/NCI60_SuperLearner/master/Data/VariantTableByGene.csv")
library('RCurl')
x <- getURL("https://raw.githubusercontent.com/seandavi/NCI60_SuperLearner/master/Data/VariantTableByGene.csv")
tmp <- read.csv(text = x)
VariantTable = as.matrix(tmp[,-1])
rownames(VariantTable) = tmp[,1]

## Load drug/agent GI50 data
#DrugDat <- read.csv("https://raw.githubusercontent.com/seandavi/NCI60_SuperLearner/master/Data/AOD_IOA_GI50.csv")
x <- getURL("https://raw.githubusercontent.com/seandavi/NCI60_SuperLearner/master/Data/AOD_IOA_GI50.csv")
DrugDat <- read.csv(text = x)

# find overlap in cell lines
setdiff(rownames(VariantTable), unique(DrugDat$CELL))
setdiff(unique(DrugDat$CELL), rownames(VariantTable))

# fix names in exome data (to match the names in the drug data)
rownames(VariantTable) <- sub("A549_ATCC", "A549/ATCC", rownames(VariantTable))
rownames(VariantTable) <- sub("COLO-205", "COLO 205", rownames(VariantTable))
rownames(VariantTable) <- sub("DU145", "DU-145", rownames(VariantTable))
rownames(VariantTable) <- sub("HCC2998", "HCC-2998", rownames(VariantTable))
rownames(VariantTable) <- sub("HL-60", "HL-60(TB)", rownames(VariantTable))
rownames(VariantTable) <- sub("Hs_578T", "HS 578T", rownames(VariantTable))
rownames(VariantTable) <- sub("IGR-OV1", "IGROV1", rownames(VariantTable))
rownames(VariantTable) <- sub("K562", "K-562", rownames(VariantTable))
rownames(VariantTable) <- sub("LOX_IMVI", "LOX IMVI", rownames(VariantTable))
rownames(VariantTable) <- sub("MDA-MB-231", "MDA-MB-231/ATCC", rownames(VariantTable))
rownames(VariantTable) <- sub("NCI-ADR-RES", "NCI/ADR-RES", rownames(VariantTable))
rownames(VariantTable) <- sub("RXF-393", "RXF 393", rownames(VariantTable))

# restrict drug data to overlap with exome data
DrugDat <- DrugDat[which(DrugDat$CELL %in% rownames(VariantTable)), ]

# Change our drug data into a more usable format
GI50Wide <- reshape(DrugDat[, c("NSC", "CELL", "NLOGGI50")], direction = "wide", timevar = "NSC", idvar = "CELL")
colnames(GI50Wide) <- sub("NLOGGI50.", "NSC", colnames(GI50Wide))

# use cell line name as rowname
rownames(GI50Wide) <- GI50Wide[, 1]
# remove cell line name, and line up cell lines with Variant Table
GI50Wide <- GI50Wide[rownames(VariantTable), -1]
all.equal(rownames(VariantTable), rownames(GI50Wide))

## filter VariantTable -- why?
CountVar <- colSums(VariantTable)
VariantTable_sub <- as.data.frame(VariantTable[, CountVar > 4])

## clean variable names in variant table
colnames(VariantTable_sub) <- make.names(colnames(VariantTable_sub)) # some genes have "-" in the name, but this creates problems for regression
models/formulas, replace with "."

# random forest
library(randomForest)
rfPredict = function(i) {
  require(randomForest)
  idx = !is.na(GI50Wide[,i])
  rf  = randomForest(x = VariantTable_sub[idx, ], y = GI50Wide[idx, i])
  return(rf)
}

res = lapply(seq_along(GI50Wide),rfPredict)
varExplained = sapply(res,function(x) {return(abs(x$rsq[length(x$rsq)]))})
hist(varExplained)
# and find
which(varExplained>0.7)

# what is the compound name?
colnames(GI50Wide)[3]
# Which gene is most important in the prediction?
varImpPlot(res[[3]])

###########################################
# Linear Regression                       #
# Correlation Matrix; Backward Selection   #
###########################################

# load the library
library(mlbench)
library(caret)
library(MASS)
#compute the correlation matrix
correlationMatrix <- cor(VariantTable_sub)
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.3)

rclpredict = function(i) {
  idx = !is.na(GI50Wide[,i])
  y = GI50Wide[idx, i]
  x = VariantTable_sub[idx,-highlyCorrelated]
  rclfit  = lm(y~.,data=x)
  step<-step(rclfit,direction="backward")
  return(step)
}

cor_reg = lapply(seq_along(GI50Wide),rclpredict)
# Let's use R square explained as a measure of "goodness" of fit.
varExplained = sapply(cor_reg,function(x) {return(summary(x)$r.squared)})
hist(varExplained)
# and find
which(varExplained>0.7)

# what is the compound name?
colnames(GI50Wide)[6]
# output anova
af <- anova(cor_reg [[6]])
afss <- af$"Sum Sq"

print(afss/sum(afss))
# Which gene is most important in the prediction?
var_exp <- cbind(af,PctExp=afss/sum(afss))
print(var_exp['PctExp'])
sorted_var = sort(var_exp$PctExp)
name <- row.names(var_exp)[order(var_exp$PctExp)]
barplot(sorted_var, main="Variance Percentage Explained", horiz=T,
        names.arg=name, cex.names=0.5, las=2)


############################################
# PCA                                      #
############################################

# Look at the eigen value and Scree plot to select the number of components/factors
library(psych)
## run scree plot
eigen<-scree(VariantTable_sub, factors=FALSE)
## print eigenvalues
eigen$pcv
length(which(eigen$pcv>10)) # output 31
# get the first 60 principal components
pca<-principal(VariantTable_sub, nfactors=30)

# use the principal components as predictor to do linear regression
pcapredict = function(i) {
  idx = !is.na(GI50Wide[,i])
  y = GI50Wide[idx, i]
  x = as.data.frame(pca$scores[idx,1:30])
  pcafit  = lm(y~.,data=x)
  step<-step(pcafit,direction="backward")
  return(step)
}

pca_reg = lapply(seq_along(GI50Wide),pcapredict)
# Let's use R square explained as a measure of "goodness" of fit.
varExplained = sapply(pca_reg,function(x) {return(summary(x)$r.squared)})
hist(varExplained)
# and find
which(varExplained>0.7)

# what is the compound name?
colnames(GI50Wide)[3]
# output anova
af <- anova(pca_reg[[3]])
afss <- af$"Sum Sq"
print(afss/sum(afss))
# Which gene is most important in the prediction?
var_exp <- cbind(af,PctExp=afss/sum(afss))
print(var_exp['PctExp'])
sorted_var = sort(var_exp$PctExp)
name <- row.names(var_exp)[order(var_exp$PctExp)]
barplot(sorted_var, main="Variance Percentage Explained", horiz=T,
        names.arg=name, cex.names=0.5, las=2)
pca$loadings[,13:14]
```