
Gene Expression – an Overview of Problems & Solutions: 5

Utah State University

Bioinformatics: Problems and Solutions

Summer 2006

Review

- Considering several problems & solutions with gene expression data
 - Previously:
 - 1: Data quality checks
(GIGO; use graphical checks)
 - 2: Preprocessing
(Background, Normalization, Summarization)
 - 3: Differential Expression
 - 4: Effective Visualization
 - Here: 5. Gene Profiling / Gene Selection
 - Reference: Bioinformatics and Computational Biology Solutions Using R and Bioconductor, edited by Gentleman et al.
-

Problem 3: Gene Profiling / Selection

- “Observe” gene expression in different conditions – healthy vs. diseased, e.g.
 - Use simultaneous expression “profiles” of thousands of genes
 - Look at which genes are “important” in “separating” the two conditions; i.e., what determines the conditions’ “signatures”
-

Machine Learning

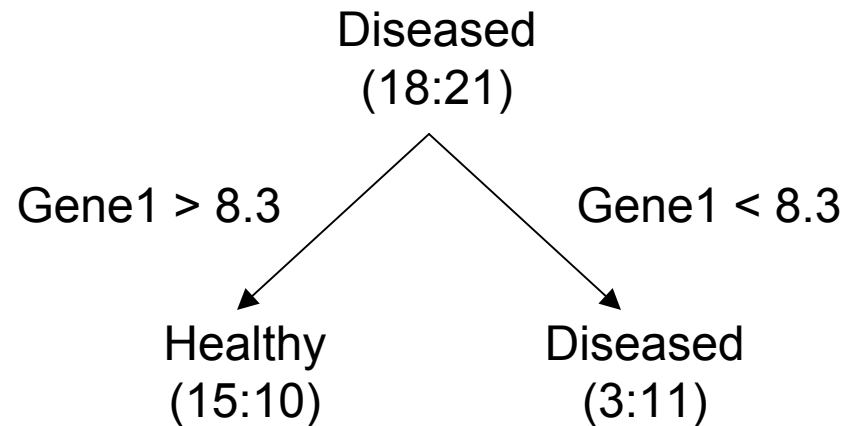
- Computational & statistical inference processes:
observed data → reusable algorithms for prediction
 - Why “machine”?
want minimal human involvement
 - Why “learning”?
develop ability to predict
 - Here, supervised learning:
use knowledge of condition type
-

Machine Learning Methods

- Neural Network
 - SVM (Support Vector Machine)
 - RPART (Recursive PArtitioning and Regression Trees)
 - CART (Classification and Regression Trees)
 - Ensembling Learning (average of many trees)
 - Boosting (Shapire et al., 1998)
 - Bagging (Breiman, 1996)
 - RandomForests (Breiman, 2001)
-

CART: Classification and Regression Trees

- Each individual (array) has data on many predictors (genes) and one response (disease state)
- Think of a tree, with splits based on levels of specific predictors
- Choose predictors and split levels to maximize “purity” in new groups; the best split at each node
- Prediction made by passing test cases down tree



CART generalized: Random Forests

- Rather than using all predictors and all individuals to make a single tree, make a forest of many (**n_{tree}**) trees, each one based on a random selection of predictors and individuals
 - Each tree is fit using a bootstrap sample of data (draw with replacement)
 - Each node is split using the best among a subset (of size **m_{try}**) of predictors randomly chosen at that node (special case using all predictors: bagging)
 - Prediction made by aggregating across the forest (majority vote or average)
-

How to measure “goodness”?

- Each tree fit on a “training” set (bootstrap sample), or the “bag”
 - The left-over cases (“out-of-bag”) can be used as a “test” set for that tree
(usually 1/3 of original data)
 - The “out-of-bag” (OOB) error rate is the % misclassification
-

What does RF give us?

- Kind of a “black box”
 - but can look at “variable importance”
- For each tree, look at the OOB data:
 - Permute values of predictor j among all OOB cases
 - Pass OOB data down the tree, save the predictions
 - For case i of OOB and predictor j , get:
 - OOB error rate with variable j permuted –
 - OOB error rate before permutation
- Average across forest to get overall variable importance for each predictor j

Why “variable importance”?

- Recall: identifying conditions' signatures
 - Sort genes by some criterion
 - Want smallest set of genes to achieve good diagnostic ability
-

Why consider RF for gene expression data?

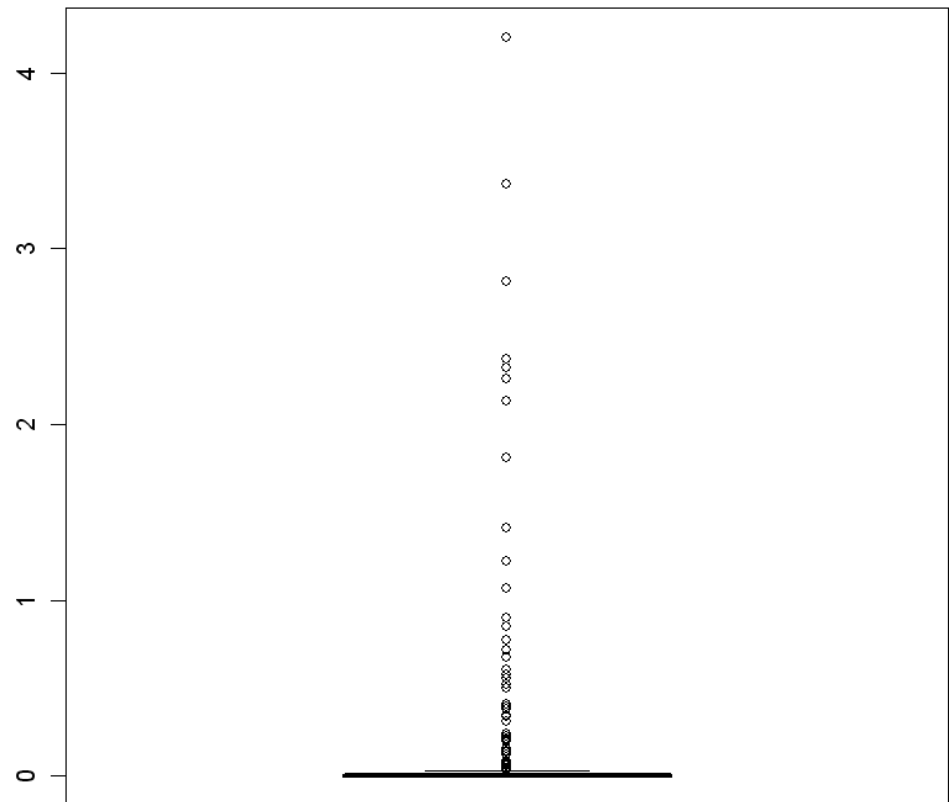
- Works well even with: many more variables than observations, many-valued categoricals, extensive missing values, badly unbalanced data
 - Low error (comparable w/ boosting and SVM)
 - Robustness (even with large “noise” in genes)
 - Does not overfit
 - Fast, and invariant to monotone transformations of the predictors
 - Free! (Fortran code by Breiman and Cutler)
 - Returns the importance of predictors (gene)
 - Little need to tune parameters
-

```
####  
#### Problem 5: Condition Signatures / Gene Selection  
####  
# Startup  
memory.limit(size=4000)  
Sys.putenv("http_proxy"="http://proxy.usu.edu:80/")  
Sys.getenv("http_proxy")  
source("http://www.stat.usu.edu/~jrstevens/get.packages.R")  
library(affy); windows(record=T)  
  
# Load data, filter, and test for DE (as done previously)  
library(ALL); data(ALL); e.mat <- 2^ALL@exprs # un-logged scale  
library(genefilter)  
ffun <- filterfun(pOverA(0.20,100), cv(0.7,10))  
small.eset <- log2(e.mat[genefilter(e.mat,ffun),])  
trt <- c(rep(0,95),rep(1,33)) # 0=B, 1=T  
design <- cbind(Intercept=1,trt=trt)  
library(limma)  
fit <- lmFit(small.eset,design); e.fit <- eBayes(fit)  
top.small <- topTable(e.fit,n=nrow(small.eset),  
                      coef=2,adjust="BH")  
gn.sig <- top.small$ID[top.small$adj.P.Val<0.05]  
gn.25 <- gn.sig[1:25]
```

```
library(randomForest)
# Define training and testing sets
set.seed(123)
t <- is.element(1:128,sample(1:128,100))
train.eset <- small.eset[,t]
train.trt <- as.factor(trt[t])
test.eset <- small.eset[,!t]
test.trt <- as.factor(trt[!t])
# Run fit
rf.fit <- randomForest(x=t(train.eset),y=train.trt,
                       xtest=t(test.eset),ytest=test.trt,
                       mtry=50)
# mtry = num.of predictors to use at each split
rf.fit
test.pred <- rf.fit$test$predicted
table(test.trt,test.pred)
#           test.pred
#test.trt  0  1
#           0 22  0
#           1  0  6
```

```
# Look at important predictors (genes) - say top 25 by importance
boxplot(rf.fit$importance)
srt.order <- sort(rf.fit$importance,dec=T,ind=T)$ix
imp.gn.25 <- rownames(rf.fit$importance)[srt.order][1:25]
# Compare with top 25 from eBayes
mean(is.element(imp.gn.25,gn.25))
# 0.8
```

So in this example, 80% of the top 25 genes from RF are among the top 25 from eBayes (with filtering)



Focus on “Variable Selection”

- Iteratively fit many forests, each time discarding predictors with low importance from previous iterations
 - Use bootstrap to assess standard error of error rates
 - Choose the forest with the smallest number of genes whose error rate is within u standard errors of the minimum error rate of all forests ($u = 0$ or 1 , typically)
 - Reference: Diaz-Uriarte and Alvarez de Andres, BMC Bioinformatics (2006) 7:3.
 - Online tool: <http://genesrf.bioinfo.cnio.es>
-

```
# An implementation that focuses on variable selection
print(date())
library(varSelRF)
rf <- varSelRF(t(small.eset), as.factor(trt))
print(date())
# This takes about 24 seconds
rf
rf.sig.gn <- rf$selected.vars
# 33238_at, 38319_at
```

Default `mtry` is sqrt of
num. of predictors

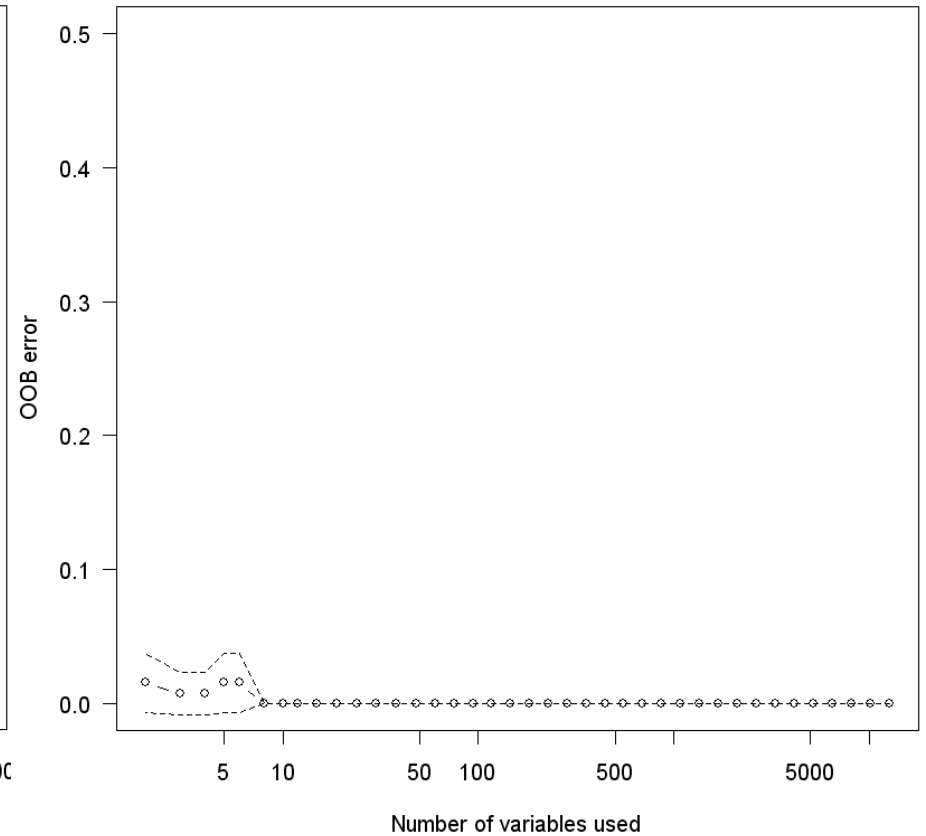
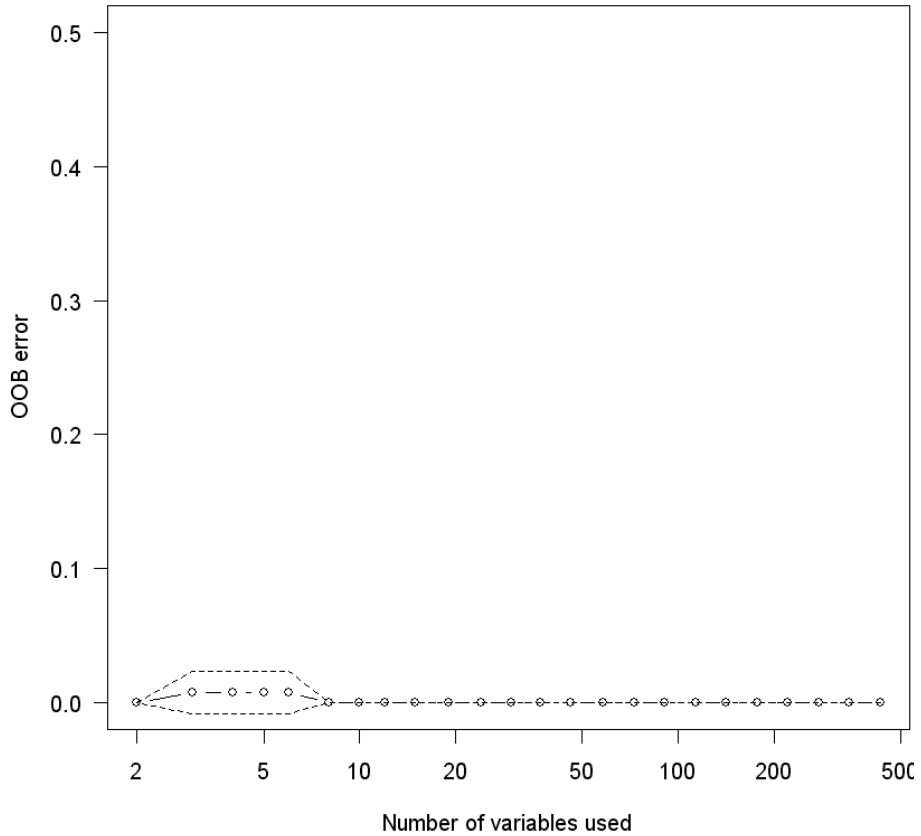
```
# What if we didn't filter first?
print(date())
eset <- ALL@exprs
rf.big <- varSelRF(t(eset), as.factor(trt))
print(date())
# This takes about 20 minutes
rf.big.sig.gn <- rf.big$selected.vars
# 2059_s_at, 33039_at, 33238_at, 37039_at,
# 37988_at, 38147_at, 38242_at, 41609_at
```

Then these genes are the “signature”

```
# Compare OOB error rate plots
```

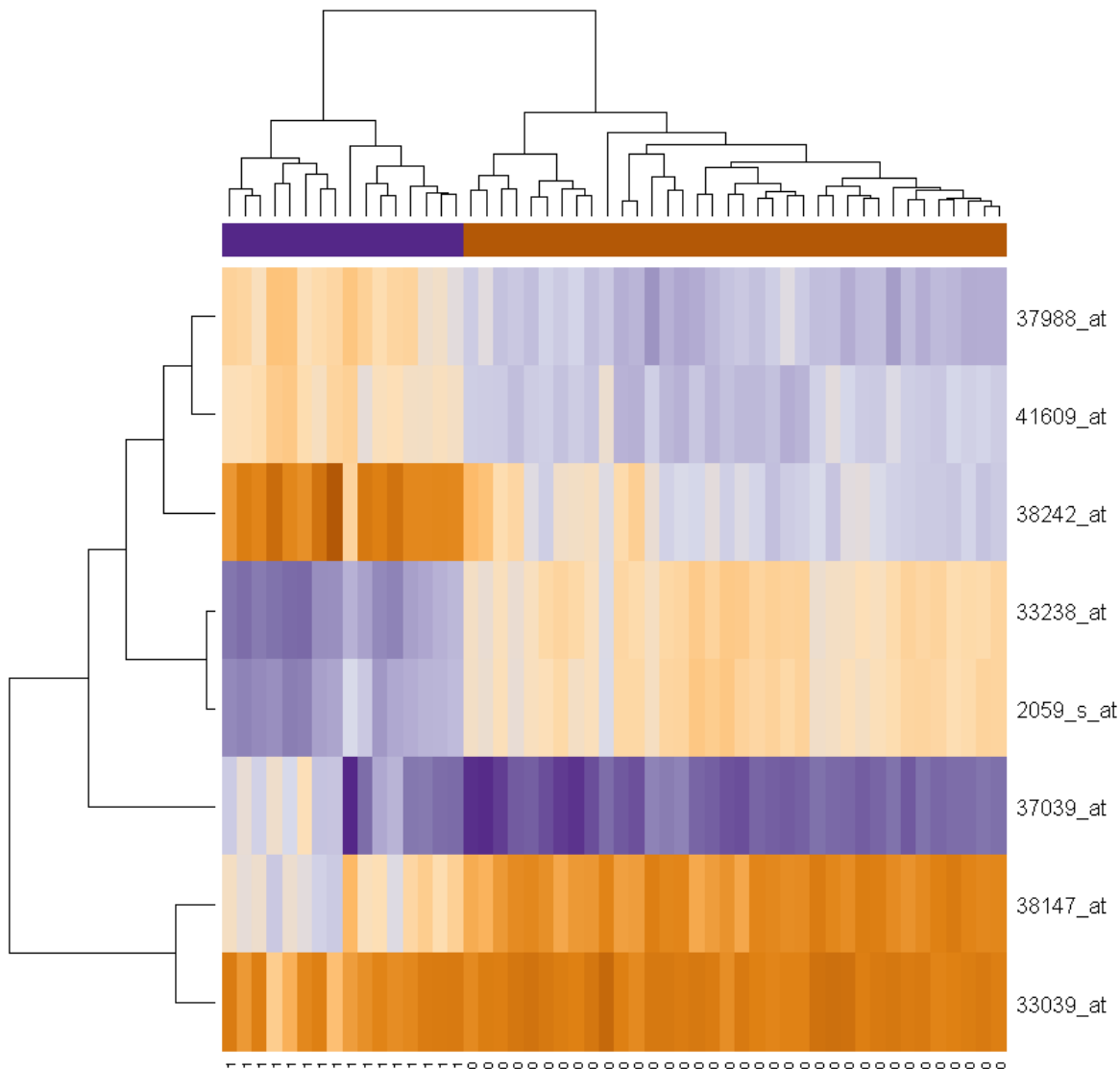
```
plot(rf)
```

```
plot(rf.big)
```



Recall Visualization Tool: Heatmap

```
# Look at a heatmap of these 8 genes
library(RColorBrewer)
set.seed(123)
n.arrays <- 52 # Look just at a few arrays
sub.arrays <- sample(1:ncol(eset),n.arrays)
rf.sig.eset <- eset[is.element(rownames(eset),rf.big.sig.gn) ,
                    sub.arrays]
hmcol <- colorRampPalette(
            brewer.pal(nrow(rf.sig.eset),"PuOr")
            )(256)
# Make group differences more obvious on plot
colnames(rf.sig.eset) <- trt[sub.arrays]
csc <- rep(hmcol[1],n.arrays)
csc[trt[sub.arrays]==1] <- hmcol[256]
# Make heatmap
heatmap(rf.sig.eset,scale="column",col=hmcol,ColSideColors=csc)
```



What is the story here?

These 8 genes (out of nearly 13,000) seem to do a “good” job of predicting disease state (0 for B cell, 1 for T cell)

There are clear blocks

Notice some genes appear to have a more “subtle” difference between disease states

Summary

- Gene profiling: look at distinguishing disease conditions based on expression profiles of genes across samples (or sample profiles across genes)
 - Various machine learning approaches exist for gene profiling
 - Random Forests is a convenient competitor
-