Using the Duke Software

Kevin R. Coombes, Jing Wang, and Keith A. Baggerly

26 March 2007

1 Problem Definition

The results that we have reported thus far have been derived from analyses that we can perform directly in R. The software that they used (posted on their website) is written in Matlab. Here, we explore various ways in which their software can be invoked using a series of batch function calls to Matlab. To run this Rnw file, you will need to have Matlab installed on your computer. You will also need to edit the file, as most of the Matlab-specific blocks currently have `eval=FALSE` set, and there are some system specific invocations.

```
> options(width = 70)
```

2 Training Data, Starting With 5-FU

2.1 Producing Files for the Software (Training)

We begin by trying to use their software to analyze the cell lines that they used for 5-FU. Their software requires an input data table of microarray quantifications, with $n_{probesets}$ by $n_{samples}$ data entries.

If this table is in an Excel file, there can be a text column giving the probeset id; the Duke script `run_binreg.m` uses the numerical output from the Matlab `xlsread` function, which discards leading columns of text. Alternatively, this table can be supplied as a comma-separated (csv) or tab-separated (tsv) text file. However, if the file suffix is anything other than xls, the Matlab function `load` is used. With `load`, there should be no text, so in particular a column of probeset ids should not be reported. In addition to the quantifications, the first row should indicate the sample classification: 0 (resistant), 1 (sensitive), or 2 (test set). We note that the mapping of numbers (0,1) to states (resistant,sensitive) was chosen arbitrarily here; there is nothing in the documentation that requires a specific direction. The contents of the classification row are discarded after the assignment is made.

For the training data, we can assemble this table by extracting the appropriate columns from the chemoPredictors object and writing them out in a csv file.

```
> library(hgu95av2)
> load(file.path("RDataObjects", "chemoPredictors.Rda"))
> load(file.path("RDataObjects", "changData.Rda"))
> load(file.path("RDataObjects", "features.Rda"))

> temp5fu <- predictors[, predictorsInfo$drugName == "5-FU"]
> tempStatus <- predictorsInfo$resp[predictorsInfo$drug ==
+     "5-FU"]
> tempValue <- rep(0, length(tempStatus))
> tempValue[tempStatus == "Sensitive"] <- 1
> tempTable <- rbind(tempValue, temp5fu)
> write.table(tempTable, file = file.path("MatlabFiles",
+     "FUTraining.csv"), row.names = FALSE, col.names = FALSE,
+     sep = ",")
> write.table(rownames(temp5fu), file = file.path("MatlabFiles",
+     "GeneIDsGood.csv"), row.names = FALSE, col.names = FALSE,
+     sep = ",")
> write.table(c("probeset id", rownames(temp5fu)), file = file.path("MatlabFiles",
+     "GeneIDsBad.csv"), row.names = FALSE, col.names = FALSE,
+     sep = ",")
> rm(temp5fu, tempStatus, tempValue, tempTable)
```

In general, however, it is dangerous to have a quantification file sitting off to the side without annotation, so we are going to export a few other files as well.

```
> filesForDuke <- function(outputPrefix, outputPath, drugName,
+     fullData, fullInfo) {
+     tempData <- fullData[, fullInfo$drugName == drugName]
+     tempStatus <- fullInfo$resp[fullInfo$drug == drugName]
+     tempValue <- rep(0, length(tempStatus))
+     tempValue[tempStatus == "Sensitive"] <- 1
+     tempQuant <- rbind(tempValue, tempData)
+     write.table(tempQuant, file = file.path(outputPath,
+         paste(outputPrefix, "Quant.csv", sep = "")),
+         row.names = FALSE, col.names = FALSE, sep = ",")
+     write.table(rownames(tempData), file = file.path(outputPath,
+         paste(outputPrefix, "GeneIDs.csv", sep = "")),
+         row.names = FALSE, col.names = FALSE, sep = ",")
+     write.table(fullInfo[fullInfo$drugName == drugName,
+         ], file = file.path(outputPath, paste(outputPrefix,
+         "SampleInfo.csv", sep = "")), row.names = TRUE,
+         col.names = TRUE, sep = ",")
+ }
```

For the training data, we then simply invoke the function.

```
> filesForDuke(outputPrefix = "FiveFUTraining", outputPath = "MatlabFiles",
+     drugName = "5-FU", fullData = predictors, fullInfo = predictorsInfo)
```

2.2 Parameter Settings and Outputs

Ok, now that we have supplied a set of raw data files, we need to clarify (a) what else we need to provide to run the Duke software in batch mode, and (b) what files we want to obtain as output.

Typically, you run the Duke software by opening Matlab with the Duke software directory somewhere in your path, and invoking `binreg`. This will pop up a GUI asking you to supply 10 modeling parameters. These parameters are:

1. **excel**: the location of the quantification file, either as an absolute path or relative to the current directory. Graphical browsing is an option. Despite the name, the software will work if a text file of all numbers and separators is supplied. Note that applying the normalization option (discussed below) will log-transform the data, so they should not be log-transformed before this step. Similarly, when normalization is used values below 1 will be thresholded (set to 1), so centering the data beforehand will also lead to problems.

2. **descFile**: The location of a Matlab .mat file containing a matrix of characters; each row of the matrix should contain the gene name (or probeset id) for the corresponding row in the excel file. Note that there should *not* be a row corresponding to the top row in the quantification file that gives the sample classifications. If there is such a row, all of the reported probesets will be off by one in lexicographical order from the one desired. Given the lists of probeset ids that we export, we can make use of one of their functions (`txt2mat2`) to convert this list to the appropriate .mat file.

3. **genes**: The number of genes to select from the training set. This is arbitrary, and is not linked to any particular inherent significance criteria (such as p-value or FDR). The genes chosen have the most extreme 2-sample pooled variance t-test values contrasting the two groups in the training set (group 0 and group 1). This selection ignores any test set data.

4. **metagenes**: The number of principal components (metagenes) used to summarize the expression matrix for the selected genes. The weightings for these vectors give the metagene scores for a particular sample; these scores will be used in a probit model for classifying the samples. The metagenes are found using a singular value decomposition (SVD) applied to the chosen expression submatrix; the values for each row (gene) are centered and scaled before this decomposition is performed. We have typically found 1 or 3 to be the numbers required for a good fit. Slight overestimates should not hurt, so using, say, 5, should be fine.

   In one version of this code, the SVD uses data from all of the samples provided, both training and test. A newer "experimental" version performs the SVD using only the training data. We will revisit this issue later.

5. **burnin**: The number of iterations to allow for the Markov chain Monte Carlo to reach the target distribution for sampling.

6. **iteration**: The number of iterations to record from the Markov chain Monte Carlo.

7. **skips**: The number of iterations to ignore between kept iterations of the Monte Carlo, so that the kept values will be roughly independent.

8. **CI**: In forming a central credible interval, this value corresponds to $\alpha/2$, the proportion to exclude at each end.

9. **crossvalidation**: A logical value indicating whether leave-one out cross-validation (LOOCV) should be used in predicting the status of a given training sample. A value of 1 indicates that cross-validation should be used, 0 that it should not. This value of this parameter is set using a checkbox in the GUI.

10. **normalization**: A logical value indicating whether the samples should be log-transformed and quantile normalized before further processing is applied. If yes, the data matrix X is first thresholded, $X(X < 1) = 1$, then log2 transformed, $X = log2(X)$, and finally quantile normalized $X = normalise(X)$. As with **crossvalidation**, 1 is yes, 0 is no, and the value is set using a checkbox in the GUI.

The parameter values are stored in a Matlab structure named `UserOptions`, and when the program is run the current version of `UserOptions` is saved in the Matlab file `preferences.mat`. Since we would like to call the processing functions in batch mode, we must specify these values in a file rather than using the GUI.

When the `binreg` function is run (with both crossvalidation and normalization set to 1), it produces a series of outputs. Specifically, it produces

1. **Figure 1**: A dotplot of the samples in "metagene" (PC) space. The two metagenes with the largest absolute weights (for separating groups 0 and 1) are used. Group 0 is in blue, group 1 is in red. This plot shows training samples only.

2. **Figure 2**: A two-panel plot showing (a) the overall weights assigned to the individual metagenes, and (b) the "expression weights" assigned to the individual genes used. Note that the weights assigned to the metagenes do not correspond to the singular values in an SVD, but rather to the size of the coefficient assigned to that PC by the predictive probit model. The gene scores are similarly weighted combinations of the scores from each of the PCs, with the weights given by the scores in (a).

3. **Figure 3**: A plot of the fitted classification probabilities and outcomes for the training set. For each sample, the "metagene score" is plotted on the x-axis, and a credible interval for the probability the sample is in group 1 on the y-axis. No cross-validation is employed here.

4. **Figure 4**: A plot of the fitted classification probabilities and outcomes for the test set. This figure is only produced if there are test cases. For each sample, the "metagene score" is plotted on the x-axis, and a credible interval for the probability the sample is in group 1 on the y-axis. No cross-validation is employed here.

5. **Figure 5**: A plot of the LOOCV fitted classification probabilities and outcomes for for the samples in the training set.

6. **Figure 11**: A heatmap showing the values of the selected genes in the expression submatrix used in the SVD. Expression values for each row (gene) are centered and scaled before plotting. If normalization was invoked, the data will also have been log-transformed. The genes are ordered according to the weights they receive from the first principal component only. Samples are largely ordered as they were in the quantification matrix supplied, save that all samples in group 0 are at the left and all samples in group 1 are at the right.

7. **imagegenelist.txt**: A table with two columns. The first column gives the position of the gene in the initially reported top gene list, and the second column gives the gene name. The order of the genes should match that in the heatmap in Figure 11.

8. **topgenecoeffs.txt**: a table with two columns. The first column is the gene coefficient (the weighted combination of PC scores), and the second column is the gene id. Unfortunately, the order of the coefficients and gene ids does not match. The function `run_binreg.m` sorts one set of values but not the other. To get them to match properly, the numerical values in the first column should be sorted, with the most positive value at the top, while leaving the column of probeset ids untouched.

9. **trainingcases.txt**: a file with 5 columns of numerical data, with rows corresponding to samples in the training set. The columns are

   (a) Sample Index.

   (b) The central probability estimate for the sample.

   (c) The lower bound for the CI.

   (d) The upper bound for the CI.

   (e) The metagene score for the sample.

   This file gives the numerical values used in Figure 3, but not the group membership. We should have this information elsewhere, however.

10. **validationcases.txt**: a file with 5 columns of numerical data, with rows corresponding to samples in the validation (test) set. This file is only produced if there are test cases. The columns are

    (a) Sample Index.

    (b) The central probability estimate for the sample.

    (c) The lower bound for the CI.

    (d) The upper bound for the CI.

    (e) The metagene score for the sample.

This file gives the numerical values used in Figure 4, but not the group membership. We should have this information elsewhere, however.

11. **normalisedX.txt**: the log-transformed and normalised data.

The numbering of the figures given above is taken from the Duke software output. The gap between "Figure 5" and "Figure 11" is not a mistake.

2.3 Invoking the Duke Software from R (Training)

Invoking the Duke software from R requires the following steps:

1. Write a Matlab m-file to perform the analysis.
2. Write a shell script to invoke Matlab and run the m-file.
3. Invoke the shell script.

Unfortunately, some of these steps are likely to be system dependent. We have not tested this thoroughly.

For the case where we are just working with training data, we can wrap most of this in a function.

```
> mFilesForDuke <- function(outputPrefix, outputPath, drugName,
+     userOptions, matlabLocation) {
+     pfx <- file.path(outputPath, outputPrefix)
+     mFileName <- paste(pfx, ".m", sep = "")
+     scriptName <- paste(pfx, ".sh", sep = "")
+     dukeLocn <- file.path("..", "PublicData", "DukeWebSite",
+         "Pittman_zip")
+     mf <- file(mFileName, "w")
+     cat("addpath('", dukeLocn, "');\n\n", file = mf,
+         sep = "")
+     cat("txt2mat2('", pfx, "GeneIDs.csv');\n", file = mf,
+         sep = "")
+     cat("movefile('descriptions.mat','", pfx, "Descriptions.mat');\n\n",
+         file = mf, sep = "")
+     for (i1 in 1:length(userOptions)) {
+         cat("UserOptions.", names(userOptions)[i1], "=",
+             userOptions[[i1]], ";\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("silent = 0;\n", file = mf)
+     cat("run_binreg(UserOptions, silent);\n\n", file = mf)
+     figNames <- c("PCA", "MetaWts", "ProbVsWt", "ProbVsWtTest",
+         "CrossVal", "Heatmap")
+     figPfx <- c("'-dpng'", "'-dpdf'", "'-depsc'")
+     figSfx <- c(".png", ".pdf", ".eps")
+     figPth <- file.path(outputPath, "Figures", outputPrefix)
+     for (i1 in c(1, 2, 3, 5)) {
+         cat("figure(", i1, ");\n", file = mf, sep = "")
+         cat("set(gcf,'Color','w');\n", file = mf)
+         for (i2 in 1:3) {
+             cat("print(", figPfx[i2], ",'", figPth, figNames[i1],
+                 figSfx[i2], "');\n", file = mf, sep = "")
+         }
+         cat("\n", file = mf)
+     }
+     cat("figure(11);\n", file = mf, sep = "")
+     cat("set(gcf,'Color','w');\n", file = mf)
+     cat("title('", drugName, ", ", userOptions$genes,
+         " Genes');\n", file = mf, sep = "")
+     cat("xlabel('Samples');\n", file = mf)
+     cat("ylabel('Genes');\n", file = mf)
+     for (i1 in 1:3) {
+         cat("print(", figPfx[i1], ",'", figPth, figNames[length(figNames)],
+             figSfx[i1], "');\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("silent = 1;\n", file = mf)
+     cat("run_binreg(UserOptions, silent);\n\n", file = mf)
+     cat("load('plotdata.mat');\n", file = mf)
+     cat("save('", pfx, "PlotData.mat','Silent','-V6');\n\n",
+         file = mf, sep = "")
+     inFiles <- c("'imagegenelist.txt'", "'topgenecoeffs.txt'",
+         "'trainingcases.txt'", "'normalisedX.txt'")
+     outFiles <- c("ImageGeneList.txt'", "TopGeneCoeffs.txt'",
+         "TrainingCases.txt'", "NormalisedX.txt'")
+     for (i1 in 1:4) {
+         cat("movefile(", inFiles[i1], ",'", pfx, outFiles[i1],
+             ");\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("quit;\n", file = mf)
+     close(mf)
+     sf <- file(scriptName, "w")
+     cat(matlabLocation, " -nojvm -nodesktop -nosplash ",
+         "< ", mFileName, " > /dev/null\n", file = sf,
+         sep = "")
+     close(sf)
+     system(paste("chmod 777 ", scriptName))
+     system(paste("./", scriptName, sep = ""))
+ }
```

The above function calls the the Matlab script `run_binreg(UserOptions, silent)` twice, once with `silent = 0` and once with `silent = 1`. Calling the routine with `silent = 0` produces all of the figures and text files mentioned above, but the output text files do not contain enough information to reconstruct the figures. For example, the final metagene weights (shown in the top half of Figure 2) are not reported. Calling the routine with `silent = 1` does not produce the figures directly, but rather stores all of the information needed to construct the figures in a Matlab structure named "`Silent`", which is saved in `plotdata.mat`. Mat-files produced by the most recent version of Matlab (version 7) are compressed and cannot be read directly into R. Mat-files in version 6 format can be read into R using the `R.matlab` package. (We noted this after writing most of this documentation, so this library is not used here.) Thus, we reload `plotdata.mat` and save it again in version 6 format.

Note that certain aspects of the function, such as the value of `matlabLocation`, the form of the shell scripts, and the system commands will need to be modified to work on other systems. The version shown here works on my G4 Mac laptop.

Now we can test the function.

```
> outPth <- "MatlabFiles"
> outPfx <- "FiveFUTraining"
> dName <- "5-FU"
> userOptions <- list(excel = paste("'", file.path(outPth,
+     outPfx), "Quant.csv'", sep = ""), descFile = paste("'",
+     file.path(outPth, outPfx), "Descriptions.mat'", sep = ""),
+     genes = 45, metagenes = 5, burnin = 1000, iteration = 100,
+     skips = 2, CI = 5, crossvalidation = 1, normalization = 1)
> matlabLocation <- "/Applications/MATLAB701/bin/matlab"
> filesForDuke(outputPrefix = outPfx, outputPath = outPth,
+     drugName = dName, fullData = predictors, fullInfo = predictorsInfo)
> mFilesForDuke(outputPrefix = outPfx, outputPath = outPth,
+     drugName = dName, userOptions = userOptions, matlabLocation = matlabLocation)
```

Nothing exploded when we ran it. Given this setup, we can now cycle through the assays for all of the drugs. For the meantime, we will not worry much about some of the parameter settings (burnin, iterations, skips, CI, crossvalidation), as these do not affect the heatmaps or gene lists produced. We focus our attention on (a) loading from the right files (excel and descFile), (b) changing the number of genes to match that used for the drug, (c) using a minimal number of metagenes (here 5), and (d) using normalized data. As a warning, this block takes about 4 minutes per drug to run on my G4 Mac laptop.

```
> outPth <- "MatlabFiles"
> userOptions <- list(excel = paste("'", file.path(outPth,
+     outPfx), "Quant.csv'", sep = ""), descFile = paste("'",
+     file.path(outPth, outPfx), "Descriptions.mat'", sep = ""),
+     genes = 45, metagenes = 5, burnin = 1000, iteration = 100,
+     skips = 2, CI = 5, crossvalidation = 1, normalization = 1)
> matlabLocation <- "/Applications/MATLAB701/bin/matlab"
> outPfxList <- c("FiveFUTraining", "AdriaTraining", "CytoxTraining",
+     "DoceTraining", "EtopoTraining", "TaxolTraining",
+     "TopoTraining")
> dNameList <- c("5-FU", "Adria", "Cytox", "Doce", "Etopo",
+     "Taxol", "Topo")
> nGeneList <- c(45, 80, 35, 50, 50, 36, 150)
> for (i1 in 2:7) {
+     outPfx <- outPfxList[i1]
+     dName <- dNameList[i1]
+     userOptions$excel <- paste("'", file.path(outPth,
+         outPfx), "Quant.csv'", sep = "")
+     userOptions$descFile <- paste("'", file.path(outPth,
+         outPfx), "Descriptions.mat'", sep = "")
+     userOptions$genes <- nGeneList[i1]
+     filesForDuke(outputPrefix = outPfx, outputPath = outPth,
+         drugName = dName, fullData = predictors, fullInfo = predictorsInfo)
+     mFilesForDuke(outputPrefix = outPfx, outputPath = outPth,
+         drugName = dName, userOptions = userOptions,
+         matlabLocation = matlabLocation)
+ }
```

2.4 Training Set Findings

2.4.1 Matching Heatmaps

Checking the heatmaps produced, we find that these match those reported in the paper *exactly* for 6 of the 7 drugs; the exception is cytoxan which does not match at all. We note that the heatmap shown in Figure 2a of the paper for cytoxan is actually the heatmap for paclitaxel; some mislabeling has taken place. These heatmaps are shown in Figure 1 (docetaxel) and Figure 2 (all the rest); the layout has been chosen to most closely match that presented in the paper.

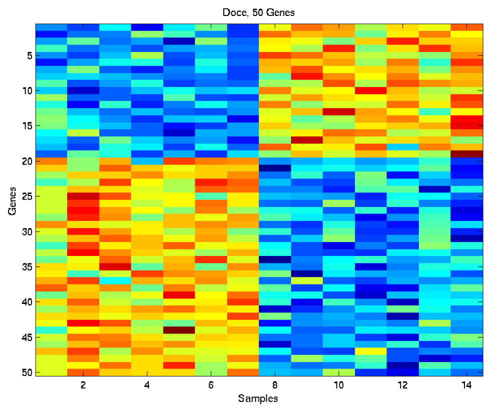


Figure 1: Heatmap for the genes selected from the Docetaxel training set. This matches the heatmap shown in Figure 1 of Potti et al.

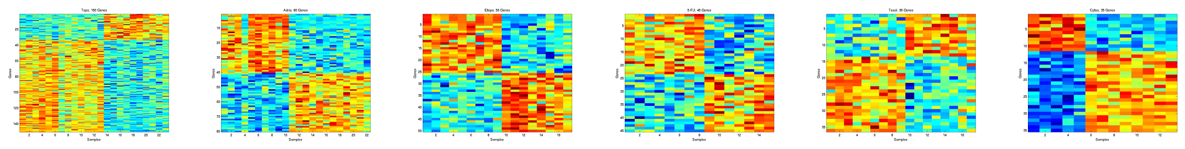


Figure 2: Heatmaps for the genes selected from the training sets for topotecan, adriamycin, etoposide, 5-fluorouracil, paclitaxel, and cytoxan. This matches the heatmap shown in Figure 2 of Potti et al. for the first four panels. The fifth panel shown here, for paclitaxel, matches the heatmap shown in the paper for cytoxan. The last panel shown here, for cytoxan, does not match anything in the paper.

2.4.2 Matching Gene Lists: 5-FU

We tried matching the gene lists returned by their software with the gene lists reported in the paper, starting with 5FU.

```
> fiveFUNames <- read.table(file.path("MatlabFiles", "FiveFUTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> fiveFUNames$V2 <- as.character(fiveFUNames$V2)
> fiveFUNames$V2 <- sub(" +$", "", fiveFUNames$V2)
> length(intersect(fiveFUNames$V2, reportedFeatures$"5"))
```

```
[1] 0
```

```
> fiveFUInds <- match(fiveFUNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[fiveFUInds - 1],
+     reportedFeatures$"5"))
```

```
[1] 45
```

```
> length(reportedFeatures$"5")
```

```
[1] 45
```

If we check the lists of genes their software outputs, these do not match those reported at all. This is because they ran the software with a version of `descriptions.mat` that included a header line; as noted in the discussion of the `descFile` option above, this introduces an off-by-one indexing error. Thus, all of the reported probeset ids are off by one from the ones desired.

After correcting for the off-by-one error, we get perfect agreement between the reported list for 5-FU and the list we produced using their software.

We tried the same thing with the other drugs.

2.4.3 Matching Gene Lists: Topotecan

```
> topoNames <- read.table(file.path("MatlabFiles", "TopoTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> topoNames$V2 <- as.character(topoNames$V2)
> topoNames$V2 <- sub(" +$", "", topoNames$V2)
> length(intersect(topoNames$V2, reportedFeatures$TOPO))
```

```
[1] 7
```

```
> topoInds <- match(topoNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[topoInds - 1],
+     reportedFeatures$TOPO))
```

```
[1] 150
```

```
> length(reportedFeatures$TOPO)
```

```
[1] 150
```

We get perfect agreement for topotecan after correcting for the off by one error.

2.4.4 Matching Gene Lists: Etoposide

```
> etopoNames <- read.table(file.path("MatlabFiles", "EtopoTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> etopoNames$V2 <- as.character(etopoNames$V2)
> etopoNames$V2 <- sub(" +$", "", etopoNames$V2)
> length(intersect(etopoNames$V2, reportedFeatures$ETOPO))
```

```
[1] 3
```

```
> etopoInds <- match(etopoNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[etopoInds - 1],
+     reportedFeatures$ETOPO))
```

```
[1] 50
```

```
> length(reportedFeatures$ETOPO)
```

```
[1] 50
```

We get perfect agreement for etoposide after correcting for the off by one error.

2.4.5 Matching Gene Lists: Adriamycin

```
> adriaNames <- read.table(file.path("MatlabFiles", "AdriaTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> adriaNames$V2 <- as.character(adriaNames$V2)
> adriaNames$V2 <- sub(" +$", "", adriaNames$V2)
> length(intersect(adriaNames$V2, reportedFeatures$ADRIA))
```

```
[1] 0
```

```
> adriaInds <- match(adriaNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[adriaInds - 1],
+     reportedFeatures$ADRIA))
```

```
[1] 75
```

```
> length(reportedFeatures$ADRIA)

[1] 80

> adriaOut <- setdiff(reportedFeatures$ADRIA, rownames(predictors)[adriaInds -
+     1])
> unlist(mget(adriaOut, hgu95av2SYMBOL))

1258_s_at 1847_s_at   1909_at 1910_s_at 2034_s_at
  "ERCC4"    "BCL2"    "BCL2"    "BCL2"  "CDKN1B"
```

For adriamycin, we match 75 of the 80 genes reported after correcting for the off by one error. The 5 outliers and their gene symbols are given above.

2.4.6 Matching Gene Lists: Paclitaxel

```
> taxolNames <- read.table(file.path("MatlabFiles", "TaxolTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> taxolNames$V2 <- as.character(taxolNames$V2)
> taxolNames$V2 <- sub(" +$", "", taxolNames$V2)
> length(intersect(taxolNames$V2, reportedFeatures$PAC))

[1] 0

> taxolInds <- match(taxolNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[taxolInds - 1],
+     reportedFeatures$PAC))

[1] 28

> length(reportedFeatures$PAC)

[1] 35

> taxolOut <- setdiff(reportedFeatures$PAC, rownames(predictors)[taxolInds -
+     1])
> unlist(mget(taxolOut, hgu95av2SYMBOL))

  1258_s_at   1802_s_at   1878_g_at    33047_at    36519_at
    "ERCC4"     "ERBB2"     "ERCC1"   "BCL2L11"     "ERCC1"
   40567_at    114_r_at
"K-ALPHA-1"      "MAPT"
```

For paclitaxel, we match all but 7 of the reported genes after accounting for the off by one error. The 7 outliers and their gene symbols are given above.

2.4.7 Matching Gene Lists: Docetaxel

```
> doceNames <- read.table(file.path("MatlabFiles", "DoceTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> doceNames$V2 <- as.character(doceNames$V2)
> doceNames$V2 <- sub(" +$", "", doceNames$V2)
> length(intersect(doceNames$V2, reportedFeatures$DOCE))

[1] 0

> doceInds <- match(doceNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[doceInds - 1],
+     reportedFeatures$DOCE))

[1] 31

> length(reportedFeatures$DOCE)

[1] 50

> doceOut <- setdiff(reportedFeatures$DOCE, rownames(predictors)[doceInds -
+     1])
> unlist(mget(doceOut, hgu95av2SYMBOL))

  1258_s_at   1751_g_at   1802_s_at   1878_g_at   1997_s_at
    "ERCC4"    "FARSLA"     "ERBB2"     "ERCC1"       "BAX"
  2085_s_at    31431_at  31432_g_at    31638_at    32099_at
   "CTNNA1"     "FCGRT"     "FCGRT"          NA     "SAFB2"
   32331_at    32523_at  32843_s_at    33047_at    33133_at
    "AK3L1"      "CLTB"       "FBL"   "BCL2L11"      "FLII"
   33214_at  33285_i_at  33371_s_at    40567_at
   "MRPS12"      "SIKE"     "RAB31" "K-ALPHA-1"
```

For docetaxel, we match 31 of the 50 reported features after accounting for the off by one error. The 19 outliers and their gene symbols are given above.

2.4.8 Matching Gene Lists: Cytoxan

```
> cytoxNames <- read.table(file.path("MatlabFiles", "CytoxTrainingTopGeneCoeffs.txt"),
+     sep = "\t")
> cytoxNames$V2 <- as.character(cytoxNames$V2)
> cytoxNames$V2 <- sub(" +$", "", cytoxNames$V2)
> length(intersect(cytoxNames$V2, reportedFeatures$CYTOX))

[1] 0
```

```
> cytoxInds <- match(cytoxNames$V2, rownames(predictors))
> length(intersect(rownames(predictors)[cytoxInds - 1],
+     reportedFeatures$CYTOX))

[1] 0

> length(reportedFeatures$CYTOX)

[1] 35
```

For cytoxan, we just cannot match their reported lists at all.

2.4.9 Matching Wrapup, and Exploring Outliers

After correcting for the off-by-one error, we get perfect agreement between the reported lists and those that we produce for 3 of the 6 genes (cytoxan just does not work). For docetaxel, there are 19 outliers. For adriamycin, there are 5 outliers. For paclitaxel there are 7 outliers. We have been unable to discern how the reported outliers can be derived from the training data. The associated p-values that we get using two-sample t-tests are not small.

For docetaxel, Chang et al provide a supplemental table of the 92 genes that they found to be most useful in discriminating responders from nonresponders in their data (the test data for Potti et al). This table is available online from the Lancet at http://image.thelancet.com/extras/01art11086webtable.pdf.

Of the 19 docetaxel outliers, 14 are in the Chang list:

```
> changReportedDoce <- c("1751_g_at", "1997_s_at", "2085_s_at",
+     "31431_at", "31432_g_at", "31638_at", "32099_at",
+     "32331_at", "32523_at", "32843_s_at", "33133_at",
+     "33214_at", "33285_i_at", "33371_s_at")
```

These 14 probesets are in one contiguous block in the table from Chang et al — they occupy positions 7:20. Note that these matches occur without offsetting the probeset ids by one. Of the 31 that we can derive from the software, 1, 36208_at, is in the Chang list before offsetting, and 1, 38831_f_at, is in the Chang list after offsetting.

We cannot match the other 5 docetaxel outliers,

```
> notInChang <- c("1258_s_at", "1802_s_at", "1878_g_at",
+     "33047_at", "40567_at")
```

but they are still quite interesting. All 5 are also in the list of 7 outliers for paclitaxel. The two others in the paclitaxel list are 36519_at, "ERCC1", and 114_r_at, "MAPT". The first of these, ERCC1, is also interrogated by 1878_g_at, which is a common outlier for both paclitaxel and docetaxel. There is one probeset, 1258_s_at or "ERCC4", which is an outlier for docetaxel, paclitaxel, and adriamycin. The ERCC family is overrepresented in the outliers.

Many of the outlier genes are explicitly named in the main text of Potti et al. For docetaxel, "both predictors were linked to expected targets for docetaxel, including BCL2, WDR7 (also known as TRAG), ERBB2 and tubulin genes". For paclitaxel, "An examination of the genes that constituted the paclitaxel predictor identified microtubule-associated protein tau (MAPT), described previously as a determinant of paclitaxel sensitivity." For adriamycin, "excision repair genes (for example, ERCC4), retinoblastoma pathway genes and BCL2 constituted the adriamycin predictor."

While we are unable to explain how they were chosen, the outliers are evidently important.

3 Training and Test: Docetaxel

In addition to running their software and getting the associated gene lists and heatmaps for the training sets, we would also like to examine the behavior of their software when we supply a test set as well. To this end, we apply the method to docetaxel, using their Novartis A set arrays for training and the Chang SOFT file quantifications for testing.

The functions that involve both training and test sets require some modification, as we typically do not have the same type of supporting information for the two sets of data, and the two datasets may have information recorded for different probesets. Because of this, we will simply provide a customized set of output for docetaxel here. We will modify the m-file writing function.

In addition to the R scripts, some modifications need to be made on the Matlab side as well. A minor modification occurs because more figures and files are produced if there are test samples. However, a more major alteration is needed. This is because we use two versions of one of their m-files, `Mbinregsvd.m`. The version of `Mbinregsvd.m` posted at `Combo1.php` does something we believe is wrong: it performs the SVD using the expression matrix for the training and test sets combined. This can allow information to leak from the test set into the training set and contaminate the results. The Duke group also provided an "experimental" version of this m-file that uses only the training data for the SVD.

To illustrate the effects, we have made the original version of `Mbinregsvd.m` available as `Mbinregsvd_old.m` and the experimental version as `Mbinregsvd.m`. We have produced a modified `run_binreg_old.m` which differs from `run_binreg.m` solely in terms of which version of `Mbinregsvd.m` it invokes.

3.1 Exporting Both Training and Testing Data

First, export the quantification data.

```
> tempDoce <- predictors[, predictorsInfo$drugName == "Doce"]
> tempDoce <- cbind(tempDoce, changSoft[rownames(tempDoce),
+     ])
> tempStatus <- predictorsInfo$resp[predictorsInfo$drug ==
+     "Doce"]
> tempStatus <- as.character(tempStatus)
> tempStatus <- c(tempStatus, as.character(changInfo$Resp))
> tempValue <- rep(0, length(tempStatus))
> tempValue[tempStatus == "Sensitive"] <- 1
> tempValue[tempStatus == "Resp"] <- 2
> tempValue[tempStatus == "NR"] <- 2
> tempTable <- rbind(tempValue, tempDoce)
> write.table(tempTable, file = file.path("MatlabFiles",
+     "DoceTrainAndTestQuant.csv"), row.names = FALSE,
+     col.names = FALSE, sep = ",")
> write.table(predictorsInfo[predictorsInfo$drug == "Doce",
+     ], file = file.path("MatlabFiles", "DoceTrainAndTestSampleInfo.csv"),
+     sep = ",")
> write.table(changInfo, file = file.path("MatlabFiles",
+     "DoceTrainAndTestSampleInfo.csv"), sep = ",", append = TRUE)
> write.table(rownames(tempDoce), file = file.path("MatlabFiles",
+     "DoceTrainAndTestGeneIDs.csv"), row.names = FALSE,
+     col.names = FALSE, sep = ",")
```

Note that the assigned classification value of 2 indicates a test sample (here, those from the Chang data); status is "blinded" in the sense that the same value is used for all test samples.

3.2 Revising the M-file

Next, we revise the script for writing m-files to deal with the case where both training and testing samples are present.

```
> mFilesForDukeTrainTest <- function(outputPrefix, outputPath,
+     drugName, userOptions, matlabLocation) {
+     pfx <- file.path(outputPath, outputPrefix)
+     mFileName <- paste(pfx, ".m", sep = "")
+     scriptName <- paste(pfx, ".sh", sep = "")
+     dukeLocn <- file.path("..", "PublicData", "DukeWebSite",
+         "Pittman_zip")
+     mf <- file(mFileName, "w")
+     cat("addpath('", dukeLocn, "');\n\n", file = mf,
+         sep = "")
+     cat("txt2mat2('", pfx, "GeneIDs.csv');\n", file = mf,
+         sep = "")
+     cat("movefile('descriptions.mat','", pfx, "Descriptions.mat');\n\n",
+         file = mf, sep = "")
+     for (i1 in 1:length(userOptions)) {
+         cat("UserOptions.", names(userOptions)[i1], "=",
+             userOptions[[i1]], ";\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("silent = 0;\n", file = mf)
+     cat("run_binreg(UserOptions, silent);\n\n", file = mf)
+     figNames <- c("PCA", "MetaWts", "ProbVsWt", "ProbVsWtTest",
+         "CrossVal", "Heatmap")
+     figPfx <- c("'-dpng'", "'-dpdf'", "'-depsc'")
+     figSfx <- c(".png", ".pdf", ".eps")
+     figPth <- file.path(outputPath, "Figures", outputPrefix)
+     for (i1 in c(1, 2, 3, 4, 5)) {
+         cat("figure(", i1, ");\n", file = mf, sep = "")
+         cat("set(gcf,'Color','w');\n", file = mf)
+         for (i2 in 1:3) {
+             cat("print(", figPfx[i2], ",'", figPth, figNames[i1],
+                 figSfx[i2], "');\n", file = mf, sep = "")
+         }
+         cat("\n", file = mf)
+     }
+     cat("figure(11);\n", file = mf, sep = "")
+     cat("set(gcf,'Color','w');\n", file = mf)
+     cat("title('", drugName, ", ", userOptions$genes,
+         " Genes');\n", file = mf, sep = "")
+     cat("xlabel('Samples');\n", file = mf)
+     cat("ylabel('Genes');\n", file = mf)
+     for (i1 in 1:3) {
+         cat("print(", figPfx[i1], ",'", figPth, figNames[length(figNames)],
+             figSfx[i1], "');\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("silent = 1;\n", file = mf)
+     cat("run_binreg(UserOptions, silent);\n\n", file = mf)
+     cat("load('plotdata.mat');\n", file = mf)
+     cat("save('", pfx, "PlotData.mat','Silent','-V6');\n\n",
+         file = mf, sep = "")
+     inFiles <- c("'imagegenelist.txt'", "'topgenecoeffs.txt'",
+         "'trainingcases.txt'", "'normalisedX.txt'", "'validationcases.txt'")
+     outFiles <- c("ImageGeneList.txt'", "TopGeneCoeffs.txt'",
+         "TrainingCases.txt'", "NormalisedX.txt'", "ValidationCases.txt'")
+     for (i1 in 1:5) {
+         cat("movefile(", inFiles[i1], ",'", pfx, outFiles[i1],
+             ");\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     pfx <- paste(pfx, "Old", sep = "")
+     outputPrefix <- paste(outputPrefix, "Old", sep = "")
+     cat("silent = 0;\n", file = mf)
+     cat("run_binreg_old(UserOptions, silent);\n\n", file = mf)
+     figNames <- c("PCA", "MetaWts", "ProbVsWt", "ProbVsWtTest",
+         "CrossVal", "Heatmap")
+     figPfx <- c("'-dpng'", "'-dpdf'", "'-depsc'")
+     figSfx <- c(".png", ".pdf", ".eps")
+     figPth <- file.path(outputPath, "Figures", outputPrefix)
+     for (i1 in c(1, 2, 3, 4, 5)) {
+         cat("figure(", i1, ");\n", file = mf, sep = "")
+         cat("set(gcf,'Color','w');\n", file = mf)
+         for (i2 in 1:3) {
+             cat("print(", figPfx[i2], ",'", figPth, figNames[i1],
+                 figSfx[i2], "');\n", file = mf, sep = "")
+         }
+         cat("\n", file = mf)
+     }
+     cat("figure(11);\n", file = mf, sep = "")
+     cat("set(gcf,'Color','w');\n", file = mf)
+     cat("title('", drugName, ", ", userOptions$genes,
+         " Genes');\n", file = mf, sep = "")
+     cat("xlabel('Samples');\n", file = mf)
+     cat("ylabel('Genes');\n", file = mf)
+     for (i1 in 1:3) {
+         cat("print(", figPfx[i1], ",'", figPth, figNames[length(figNames)],
+             figSfx[i1], "');\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("silent = 1;\n", file = mf)
+     cat("run_binreg_old(UserOptions, silent);\n\n", file = mf)
+     cat("load('plotdata.mat');\n", file = mf)
+     cat("save('", pfx, "PlotData.mat','Silent','-V6');\n\n",
+         file = mf, sep = "")
+     inFiles <- c("'imagegenelist.txt'", "'topgenecoeffs.txt'",
+         "'trainingcases.txt'", "'normalisedX.txt'", "'validationcases.txt'")
+     outFiles <- c("ImageGeneList.txt'", "TopGeneCoeffs.txt'",
+         "TrainingCases.txt'", "NormalisedX.txt'", "ValidationCases.txt'")
+     for (i1 in 1:5) {
+         cat("movefile(", inFiles[i1], ",'", pfx, outFiles[i1],
+             ");\n", file = mf, sep = "")
+     }
+     cat("\n", file = mf)
+     cat("quit;\n", file = mf)
+     close(mf)
+     sf <- file(scriptName, "w")
+     cat(matlabLocation, " -nojvm -nodesktop -nosplash ",
+         "< ", mFileName, " > /dev/null\n", file = sf,
+         sep = "")
+     close(sf)
+     system(paste("chmod 777 ", scriptName))
+     system(paste("./", scriptName, sep = ""))
+ }
```

While the above function definition is long, there are only a few qualitative changes. Specifically:

1. The loop for producing figures now includes Figure 4, showing the test sample predictions.
2. The list of text files has been expanded to include `validationcases.txt`.
3. In the second half of the code, we call `run_binreg_old` instead of `run_binreg`.
4. In the second half of the code, the prefix (`pfx`) has been expanded to `paste(pfx, 'Old', sep=■)`, so that the files produced by `run_binreg_old` will have different names.

3.3 Invoking the Script

Finally, we invoke the revised script.

```
> outPth <- "MatlabFiles"
> outPfx <- "DoceTrainAndTest"
> dName <- "Doce"
> userOptions <- list(excel = paste("'", file.path(outPth,
+     outPfx), "Quant.csv'", sep = ""), descFile = paste("'",
+     file.path(outPth, outPfx), "Descriptions.mat'", sep = ""),
+     genes = 50, metagenes = 5, burnin = 1000, iteration = 100,
+     skips = 2, CI = 5, crossvalidation = 1, normalization = 1)
> matlabLocation <- "/Applications/MATLAB701/bin/matlab"
> mFilesForDukeTrainTest(outputPrefix = outPfx, outputPath = outPth,
+     drugName = dName, userOptions = userOptions, matlabLocation = matlabLocation)
```

3.4 Findings from the Training and Testing Data

3.4.1 Comparing Gene Lists

```
> geneListTrainOnly <- read.table(file.path("MatlabFiles",
+     "DoceTrainingTopGeneCoeffs.txt"), sep = "\t")
> geneListWithTestOldAlg <- read.table(file.path("MatlabFiles",
+     "DoceTrainAndTestOldTopGeneCoeffs.txt"), sep = "\t")
> geneListWithTestNewAlg <- read.table(file.path("MatlabFiles",
```

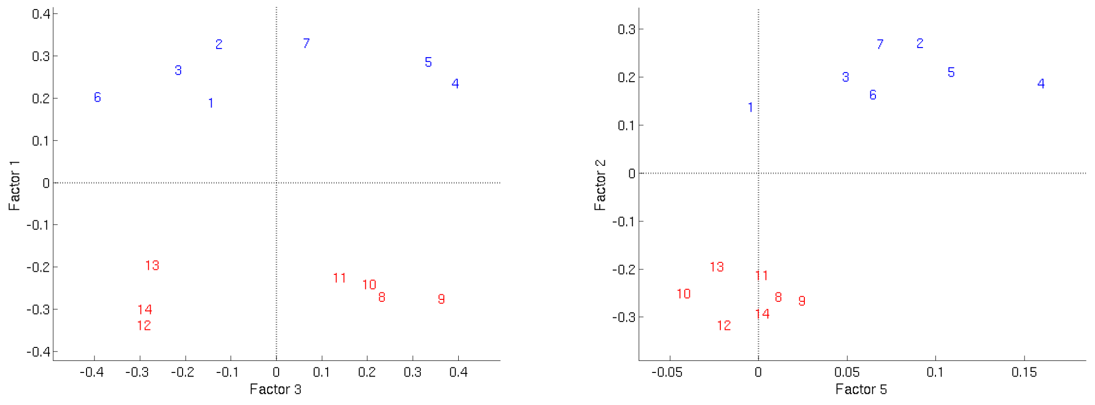


Figure 3: PCA plots for Docetaxel Training Data, showing the two principal components that are most significant in a model built to separate Group 0 (blue) from Group 1 (red). When only the training data is used in the metagene SVD, the first principal component (Factor 1, y-axis, left) is the most important factor for separating the two groups. When both training and test data are used in the SVD, this separation need not be driven by the first principal component. For docetaxel, the second principal component (Factor 2, y-axis, right) is the most important.

```
+     "DoceTrainAndTestTopGeneCoeffs.txt"), sep = "\t")
> length(intersect(geneListWithTestOldAlg$V2, geneListWithTestNewAlg$V2))

[1] 50

> length(intersect(geneListWithTestOldAlg$V2, geneListTrainOnly$V2))

[1] 47
```

The lists of genes selected when the test data are included are the same using both versions of the code. However, these lists do not precisely match that produced using the training data alone. This difference is due to the normalization step; quantile normalization is applied to all of the data supplied. Thus, when test data is supplied, normalization is acting on all of the data, training and test, rather than mapping the test values to quantiles defined by the training data.

The other differences seen using the two versions of `Mbinregsvd` are much more extreme.

3.4.2 Comparing PCA Plots

Comparing the PCA plots for the training samples, shown in Figure 3, the plots look very different, even though the sensitive and resistant groups are clearly separated in both. Using the experimental code, this difference is driven by the first factor, matching our own findings. Using the original

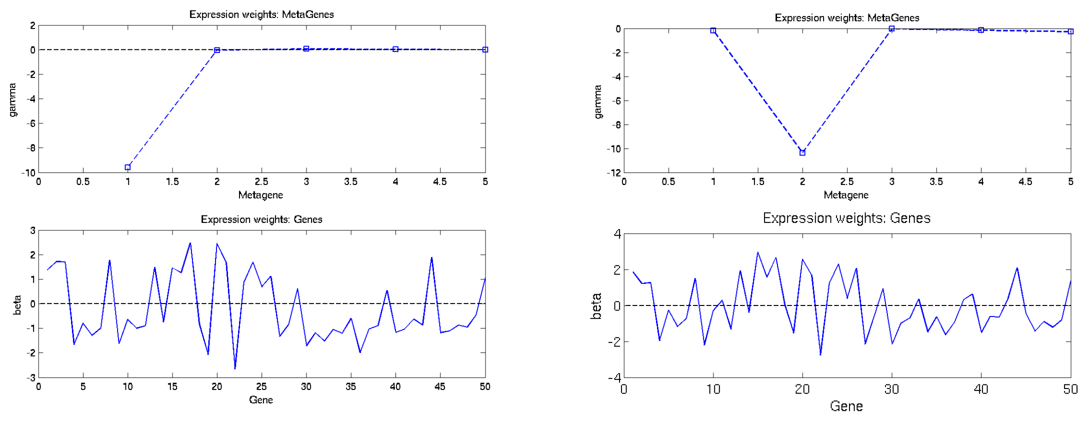


Figure 4: Metagene profile plots showing the weights assigned to each principal component (top) and to the individual genes (bottom) after the gene PCs are combined using the weights shown above. In the case of docetaxel, when just the training data is used in the SVD (left), most of the weight is given to PC1. When both training and test data are used in the SVD, most of the weight is given to PC2. The final weights assigned to the individual genes (bottom) are very similar in the two cases, but not identical. The difference is driven by “leaking” of information about major divisions in the data into several principal components.

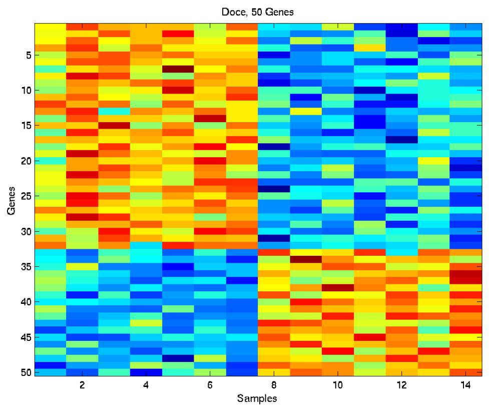
code, the first factor is actually associated with the division between training and test sets; the second is associated with the differences in the training set. In general, we expect 3 factors to be important using this approach: the two mentioned above, and a third corresponding to major divisions in the test set if such are present. The relative sizes of these 3 factors will vary from one dataset to the next.

3.4.3 Comparing Metagene Weight Profiles

Looking at the plots of which metagenes are important and the individual gene weights, shown in Figure 4, we see the abrupt change in the metagene weightings remarked on above, even though the gene weighting profiles are largely parallel. The differences that exist are due to a superposition of the 3 principal component factors discussed above.

3.4.4 Comparing Heatmaps

Looking at heatmaps of the data used to perform the SVDs, the only major visible distinction when just the training data is used is that between groups 0 and 1. When both the training and test data are used, however, several major divisions are visible. The most apparent are those between



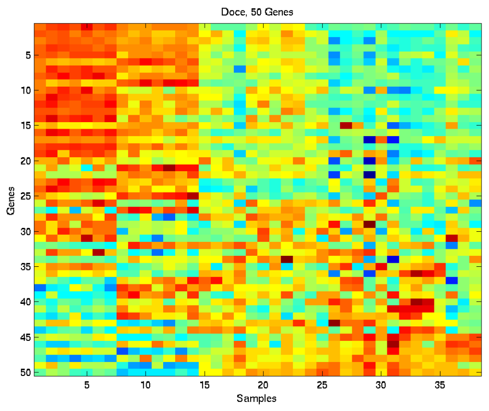


Figure 5: Heatmaps showing the data used in performing the SVDs. Using only the training data (left), there is one clear visual distinction, between group 0 and group 1. This difference is captured in the first PC. Using both the training and the test data (right), there are at least two clear distinctions: (1) between training and test, and (2) within training. Separations within the test data are more subtle, but if they exist they will be captured by the the PCs as well.

training and test, captured in PC1, and between groups 0 and 1 in the training data, captured in PC2. Other divisions are harder to see, but divisions within the test data can be captured.

3.4.5 Comparing Training Set Predictions

The plots of straight predictions for the training samples, shown in Figure 6, are largely parallel, as are the LOOCV predictions, shown in Figure 7. We do a good job of predicting the training data, and we misclassify the same samples with cross-validation.

3.4.6 Comparing Test Set Predictions

The predictions for the test cases, shown in Figure 8, however, are where the differences are most stark. Using the experimental code, *all* of the test samples are classed as group 1, matching our experience. Using the original code, the bleedover of principal component information causes the test samples to be split roughly evenly. This can lead to overly optimistic claims for predictive accuracy.

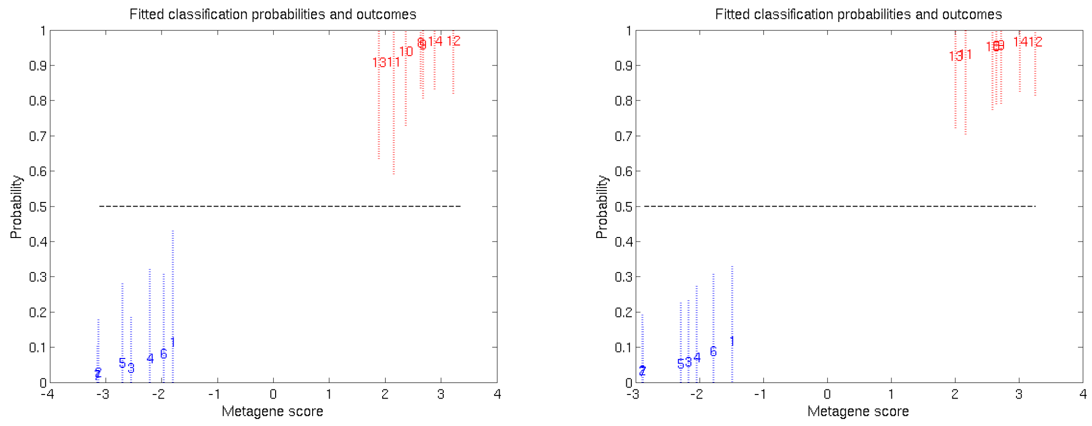


Figure 6: Predicting group membership of the samples in the training data, we do a good job of classification whether we use just the training data (left) or both the training and test (right). As the model is chosen to separate the training samples, this is not surprising.

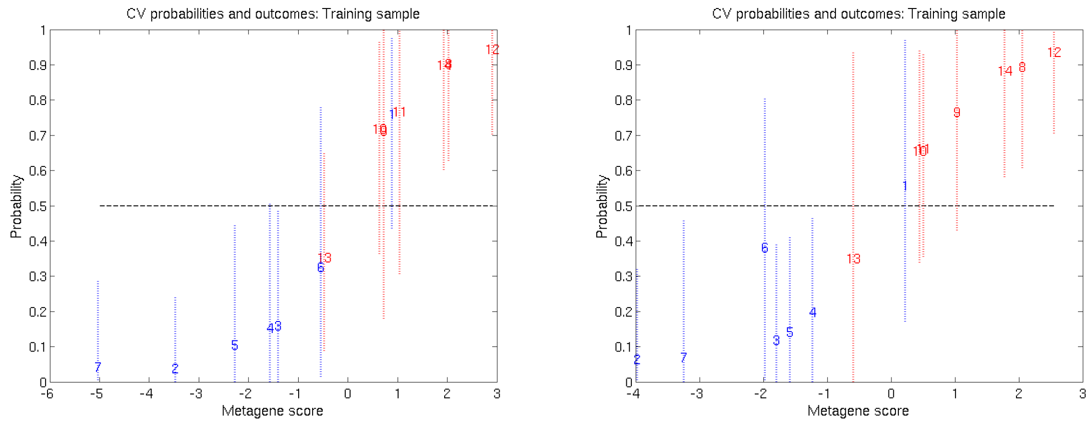


Figure 7: Predicting group membership of the samples in the training data using leave-one out cross-validation, we do a good job of classification whether we use just the training data (left) or both the training and test (right). The same training samples are misclassified. The shift in the code has not affected the training set predictions a great deal.

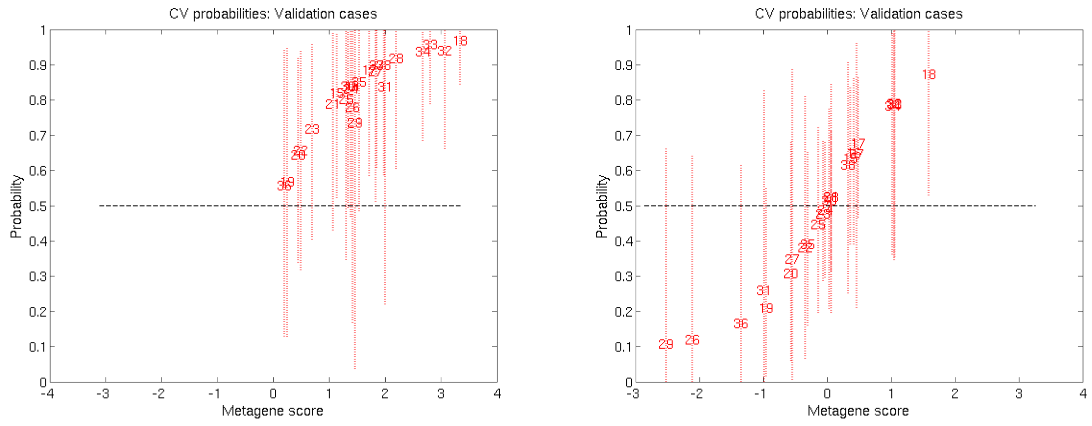


Figure 8: Predicting group membership of the samples in the test set data, we do a poor job when just the training data is used to construct the model (left), assigning all of the test data to one group. By contrast, when the SVD is performed using both training and test data, the test set predictions split nicely into two groups. Information about a split in the test data has "leaked" into the metagenes from the test set.