Random Cell Lines and the Chang Data

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

12 March 2007

1 Problem Definition

One assumption of the Duke procedure is that the choice of cell lines affects the goodness of the signature. This means that using their chosen cell lines, we should achieve much better separation of the clinical data than we would if we were to use randomly chosen cell line profiles. This assumption can be empirically tested, and in this report we attempt this in the case of docetaxel, using the Chang breast cancer data from GEO and the lists by Potti and colleagues of the sensitive and resistant cell lines.

In our approach, we simply simulate the separation process several times, where each time we draw 7 "resistant" and 7 "sensitive" cell lines (matching their numbers) at random and without replacement from the set of 59 cell lines in the A replicate set. The final separation between the test groups (defined below) is then measured for each random draw, and the starting grouping is positioned within this set.

2 Load the existing data

Start with the individual data from Novartis and the data on 10 drugs from the DTP.

```
> prep <- file.path("Tangled", "prepareData.R")
> Stangle(file.path("RNowebSource", "prepareData.Rnw"),
+     output = prep)

Writing to file Tangled/prepareData.R

> source(prep)
> rm(prep)
```

Use this function to load cached data, if it exists, and produce it from scratch otherwise.

```
> getCached <- function(rda, r) {
+     rfile <- file.path("Tangled", paste(r, "R", sep = "."))
+     rdafile <- file.path("RDataObjects", paste(rda, "Rda",
```

```
+         sep = "."))
+     if (file.exists(rdafile)) {
+         cat("loading from cache\n")
+         load(rdafile, .GlobalEnv)
+     }
+     else {
+         Stangle(file.path("RNowebSource", paste(r, "Rnw",
+             sep = ".")), output = rfile)
+         source(rfile)
+     }
+ }
```

See how easy it is to use?

```
> getCached("chemoPredictors", "predCellLines")

loading from cache

> getCached("doceGI50", "gi50ValuesOverlap")

loading from cache

> getCached("features", "wobblingFeatures")

loading from cache

> getCached("pcaModels", "pca")

loading from cache

> getCached("changData", "prepareChang")

loading from cache

> getCached("predict", "predict")

loading from cache
```

We also load the relevant OOMPA libraries.

```
> library(ClassComparison)
> library(ClassDiscovery)
```

3 Processing the Simulated Data

Now we define the parameters needed to process the data; these will later become the input parameters to a function that we run repeatedly. (The two "colors" objects are not used as function parameters, but are included here to see that they match correctly with the known status of the samples.)

```
> trainStatus <- pottiCells
> trainColors <- rep("red", length(trainStatus))
> trainColors[trainStatus == "Sensitive"] <- "green"
> nGenes <- 50
> nPC <- 5
> validationSet <- changSoftNL
> validStatus <- changInfo[, "Response"]
> validColors <- changColors
```

Now, we draw a set of 14 cell lines at random from the Novartis A set and see what the revised projections look like.

```
> novA <- novartisNL[, novartisInfo[, "ID"] == "A"]
> N <- length(trainStatus)
> randLn <- novA[, sample(ncol(novA), N)]
```

We have now generated a new, randomly chosen, training set. Now we use two-sample t-tests to select the features (genes) used to produce the predictive model.

```
> t.test.results <- MultiTtest(randLn, trainStatus)
> topNRand <- order(t.test.results@p.values)[1:nGenes]
> topTrainRand <- randLn[topNRand, ]
> topValidRand <- validationSet[topNRand, ]
```

Next, we use the selected gene-features to compute principal components, which is what will actually be used in the probit model. Note that the PCs are computed using the training set, and the validation set is then projected into the same PC space.

```
> pcaTrainRand <- SamplePCA(topTrainRand, splitter = trainStatus)
> projValidRand <- predict(pcaTrainRand, newdata = topValidRand)
> trainPCRand <- pcaTrainRand@scores[, 1:nPC]
> colnames(trainPCRand) <- paste("PC", 1:nPC, sep = "")
> trainPCRand <- data.frame(status = trainStatus, trainPCRand)
> validPCRand <- as.data.frame(projValidRand[, 1:nPC])
> colnames(validPCRand) <- paste("PC", 1:nPC, sep = "")
```

Now we can produce a new plot of the principal component space (Figure 1).

Next, we build a probit model using the principal components as predictors.

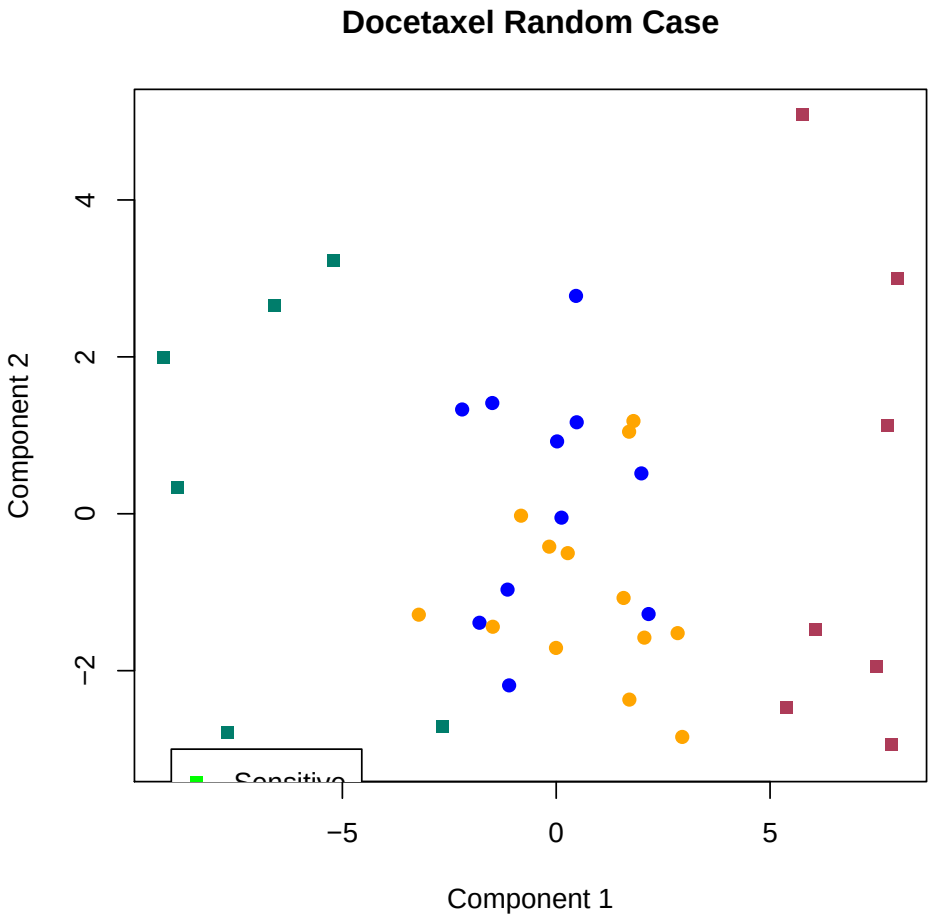

Figure 1: Principal components plot.

```
> randModel <- glm(status ~ ., family = binomial(link = probit),
+     data = trainPCRand)
> randModelStep <- step(randModel, trace = 0)
```

Here is a summary of the terms included in the model.

```
> summary(randModelStep)

Call:
glm(formula = status ~ PC1, family = binomial(link = probit),
    data = trainPCRand)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.241e-05  -2.107e-08   0.000e+00   2.107e-08   1.407e-05

Coefficients:
            Estimate Std. Error  z value Pr(>|z|)
(Intercept)    2.141   7281.878 0.000294        1
PC1           -1.586   1770.929   -0.001        1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.9408e+01  on 13  degrees of freedom
Residual deviance: 3.5486e-10  on 12  degrees of freedom
AIC: 4

Number of Fisher Scoring iterations: 24
```

Now we can use the model to make predictions.

```
> trainRandPredictions <- predict(randModelStep, type = "response")
> validRandPredictions <- predict(randModelStep, newdata = validPCRand,
+     type = "response")
> temp1 <- table(trainStatus, trainRandPredictions > 0.5)
> assignedLevels <- factor(validRandPredictions > 0.5,
+     levels = c(FALSE, TRUE))
> temp2 <- table(validStatus, assignedLevels)
> if (temp1["Resistant", "FALSE"] > 5) {
+     dimnames(temp2)[[2]] <- c("Resistant", "Sensitive")
+ } else {
+     dimnames(temp2)[[2]] <- c("Sensitive", "Resistant")
+ }
> propnNRResistant <- temp2["NR", "Resistant"]/sum(temp2["NR",
```

```
+     ])
> propnRespSensitive <- temp2["Resp", "Sensitive"]/sum(temp2["Resp",
+     ])
> propnsRand <- c(propnNRResistant, propnRespSensitive)
```

We now wrap this code into a function that we can use repeatedly.

```
> loopy <- function(trainStatus, nGenes, nPC, validationSet) {
+     novA <- novartisNL[, novartisInfo[, "ID"] == "A"]
+     N <- length(trainStatus)
+     randLn <- novA[, sample(ncol(novA), N)]
+     t.test.results <- MultiTtest(randLn, trainStatus)
+     topNRand <- order(t.test.results@p.values)[1:nGenes]
+     topTrainRand <- randLn[topNRand, ]
+     topValidRand <- validationSet[topNRand, ]
+     pcaTrainRand <- SamplePCA(topTrainRand, splitter = trainStatus)
+     projValidRand <- predict(pcaTrainRand, newdata = topValidRand)
+     trainPCRand <- pcaTrainRand@scores[, 1:nPC]
+     colnames(trainPCRand) <- paste("PC", 1:nPC, sep = "")
+     trainPCRand <- data.frame(status = trainStatus, trainPCRand)
+     validPCRand <- as.data.frame(projValidRand[, 1:nPC])
+     colnames(validPCRand) <- paste("PC", 1:nPC, sep = "")
+     randModel <- glm(status ~ ., family = binomial(link = probit),
+         data = trainPCRand)
+     randModelStep <- step(randModel, trace = 0)
+     trainRandPredictions <- predict(randModelStep, type = "response")
+     validRandPredictions <- predict(randModelStep, newdata = validPCRand,
+         type = "response")
+     temp1 <- table(trainStatus, trainRandPredictions >
+         0.5)
+     assignedLevels <- factor(validRandPredictions > 0.5,
+         levels = c(FALSE, TRUE))
+     temp2 <- table(validStatus, assignedLevels)
+     if (temp1["Resistant", "FALSE"] > 5) {
+         dimnames(temp2)[[2]] <- c("Resistant", "Sensitive")
+     }
+     else {
+         dimnames(temp2)[[2]] <- c("Sensitive", "Resistant")
+     }
+     propnNRResistant <- temp2["NR", "Resistant"]/sum(temp2["NR",
+         ])
+     propnRespSensitive <- temp2["Resp", "Sensitive"]/sum(temp2["Resp",
+         ])
```

```
+     propnsRand <- c(propnNRResistant, propnRespSensitive)
+     return(propnsRand)
+ }
```

Now we simulate 200 draws of random "sensitive" and "resistant" cell lines, using each one to make predictions on the Chang breast cancer data.

```
> nSims <- 200
> propnMat <- matrix(NA, nSims, 2)
> for (i in 1:nSims) {
+     propnMat[i, ] <- loopy(trainStatus, nGenes = 50,
+         nPC = 5, validationSet = validationSet)
+ }
```

We use this function to get the performance of models we have produced previously (in `predict.Rnw`).

```
> vPerf <- function(vOAS, vStat) {
+     ps <- factor(vOAS$psens, levels = c("Resistant",
+         "Sensitive"))
+     tab <- table(ps, vStat)
+     spec <- tab[1, 1]/sum(tab[, 1])
+     sens <- tab[2, 2]/sum(tab[, 2])
+     list(sens = sens, spec = spec)
+ }
```

Finally, we plot the results of the simulation (Figure 2). Because there are only 24 samples in the validation set, we have "jittered" the proportions slightly to make all 200 simulation results visible.

The average of the two proportions can be interpreted either as the expected accuracy (assuming the sensitive and resistant groups are the same size) or as the area under an ROC curve (AUC) on which we have only observed one point. (Here "sensitivity" refers to ability to correctly predict which patients will respond to the treatment, and "specificity" refers to our ability to correctly predict patients who will not respond.) The following computation shows that the cell lines selected by Potti and colleagues are in the middle of the distribution, using the AUC estimate as a measure of the quality of the predictions.

```
> x <- apply(propnMat, 1, sum)/2
> aucPAS <- sum(unlist(nullPAS))/2
> aucPMS <- sum(unlist(nullPMS))/2
> aucPRS <- sum(unlist(nullPRS))/2
> mean(x > aucPAS)

[1] 0.74
```

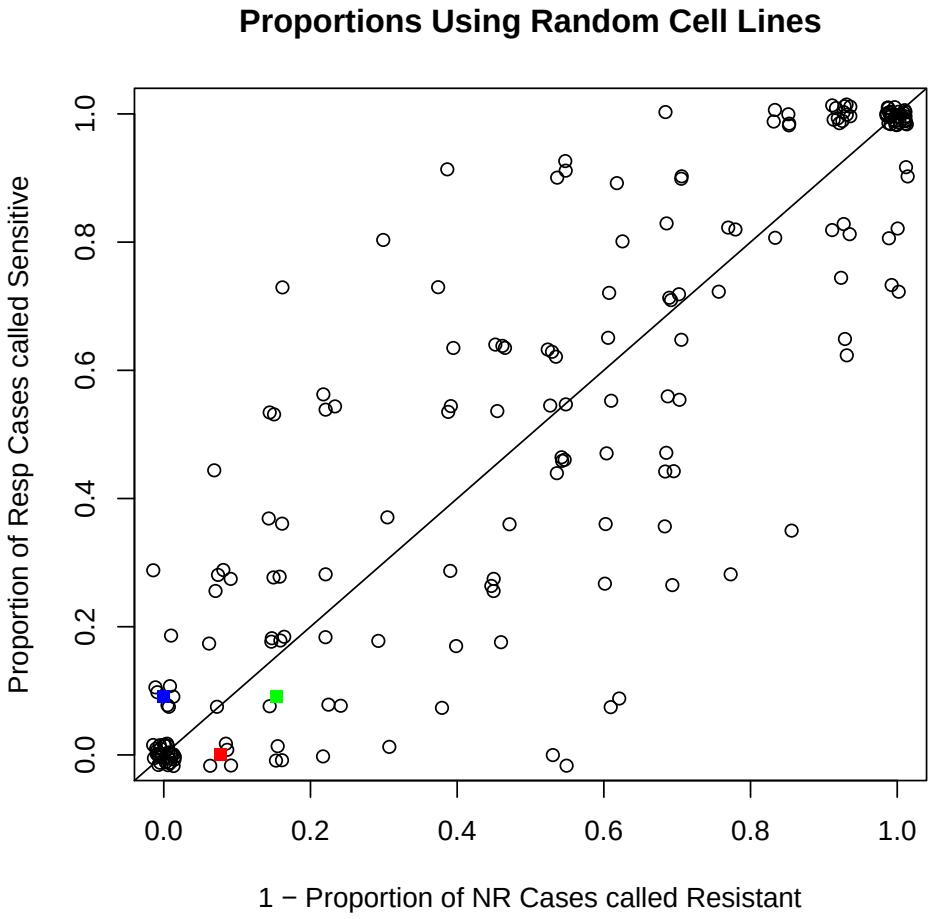

Figure 2: Prediction results using random cell lines to represent "sensitive" or "resistant" cases. The red square marks the accuracy using the cell lines chosen by Potti and colleagues.

```
> mean(x > aucPMS)

[1] 0.73

> mean(x > aucPRS)

[1] 0.225
```

4 Our cell lines

We now repeat the simulation for our choice of cell lines.

Now we simulate 200 draws of random "sensitive" and "resistant" cell lines, using each one to make predictions on the Chang breast cancer data.

```
> trainStatus <- ourCells
> trainColors <- rep("red", length(trainStatus))
> trainColors[trainStatus == "Sensitive"] <- "green"
> nSims <- 200
> ourMat <- matrix(NA, nSims, 2)
> for (i in 1:nSims) {
+     ourMat[i, ] <- loopy(trainStatus, nGenes = 50, nPC = 5,
+         validationSet = validationSet)
+ }
```

Finally, we plot the results of the simulation (Figure 3). Because there are only 24 samples in the validation set, we have "jittered" the proportions slightly to make all 200 simulation results visible.

The average of the two proportions can be interpreted either as the expected accuracy (assuming the sensitive and resistant groups are the same size) or as the area under an ROC curve (AUC) on which we have only observed one point. The following computation shows that the cell lines selected by Potti and colleagues are in the middle of the distribution, using the AUC estimate as a measure of the quality of the predictions.

```
> x <- apply(ourMat, 1, sum)/2
> aucOAS <- sum(unlist(nullOAS))/2
> aucOMS <- sum(unlist(nullOMS))/2
> mean(x > aucOAS)

[1] 0.175

> mean(x > aucOMS)

[1] 0.845
```

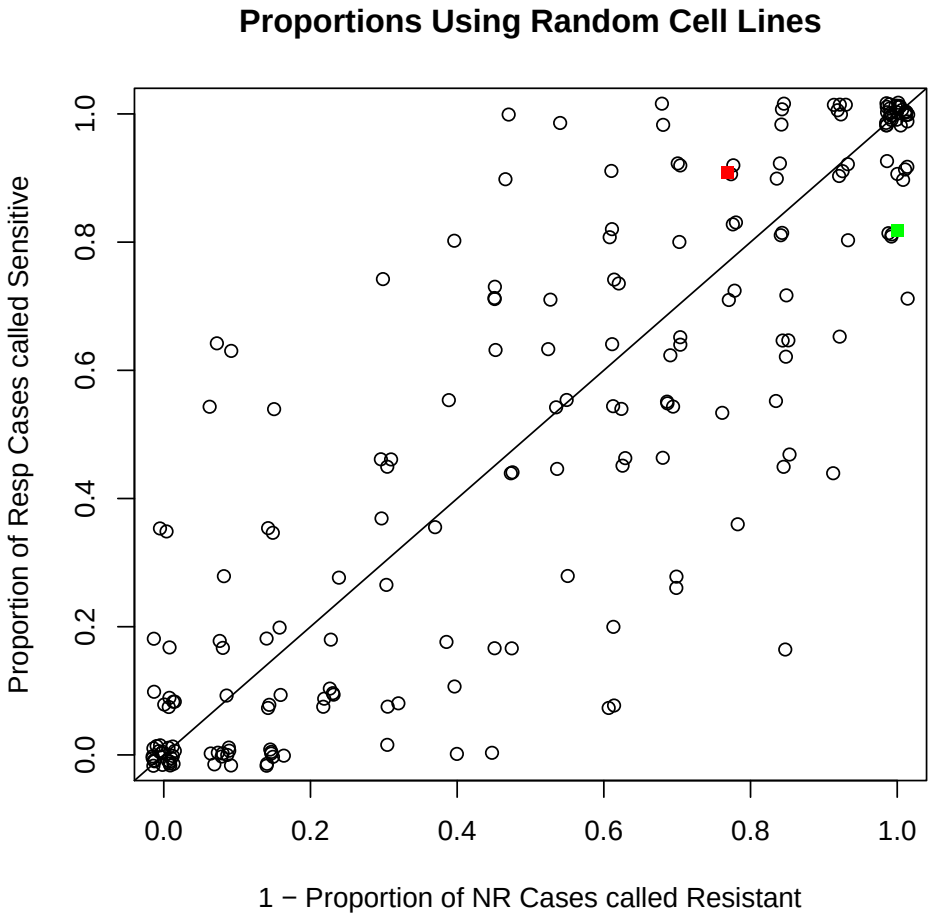

Figure 3: Prediction results using random cell lines to represent "sensitive" or "resistant" cases. The red square marks the accuracy using the cell lines chosen by Potti and colleagues.

5 Wrapup

```
> save(propnMat, ourMat, vPerf, file = file.path("RDataObjects",
+     "randomCL.Rda"))
```