SVM Lab: Supervised learning of oncogenic pathway signatures

Steve Lianoglou, Xuejing Li, Christina Leslie

March 27, 2010

Department of Computational Biology
Memorial Sloan-Kettering Cancer Center

lianos@cbio.mskcc.org

Contents

1 Overview

This tutorial will present an approach that uses support vector machines (SVMs) in order to construct "gene signatures" for different cancer pathways. Conceptually, the approach outlined here approximates a study due to Bild et al. (1), which trained binary classifiers on expression data from human cell cultures expressing specific oncogenic activities versus control cells. They

then used these classifiers to make predictions in mouse model and human tumor data as well as human cancer cells lines. We will be using a subset of the same data, but our analysis will use standard SVM methods, where theirs used a somewhat more involved Bayesian classifier based on probit regression (5).

The biological goal of this lab and the Bild et al. study is to learn an expression signature that identifies an activated oncogenic pathway and then to demonstrate the clinical relevance of classifiers based on these signatures. The original paper discussed 5 oncogenic pathways, but we will restrict to the 3 pathways associated with the expression of the following 3 oncogenes: c-Myc, activated H-Ras, and human E2F3. However, from a machine learning point of view, there are also many interesting issues: how well defined is a cancer signature? does a random set of high-variance genes perform as well as SVM-extracted discriminative features? how stable is a feature set learned from small samples? where in the analysis are we just demonstrating the results of training vs. showing actual prediction performance on a test set?

1.0.1 Software Requirements

We will be using a handful of freely available packages from the CRAN and Bioconductor (4) repositories. These packages include:

- `e1071` (2) for SVM classification
- `gplots` for fancy heatmap plots
- `scatterplot3d` for 3d scatter plots
- `caret` for machine learning utility functions
- Optional `affy` (3) for general tools to deal with microarray data and interfacing with biomart.

2 Discovering Cancer Signatures

Bild et al. have prepared a set of (ordinarily quiescent) primary mammary epithelial cell cultures (HMECs) induced to express various oncogenic activities that are typical of the activation/disregulation of several oncogenic pathways. Ten replicate microarray experiments for each cell line were produced in order to identify their transcriptional activity. For the purposes of

this lab, we will only be concerning ourselves with four of these lines: (i) GFP (control), (ii) MYC, (iii) Ras, and (iv) E2F3. We have RMA normalized the data from these four cell lines together and made them available as `gfp.exp`, `myc.exp`, `ras.exp`, `e2f3.exp` after loading the `human.base` data object. Each of these datasets are presented as a 54,675 x 10 matrix: the rows are probes (genes), and the columns are individual microarray experiments. Let's load the data:

```
> library(gplots)
> library(caret)
> load("human.base.clean.rda")
> load("human.affymap.rda")
```

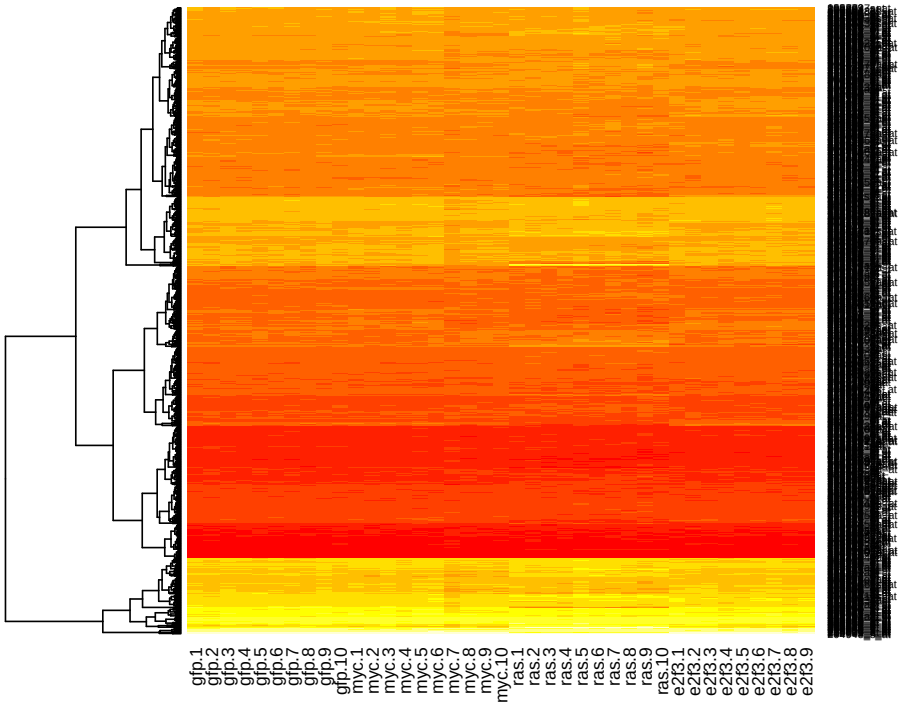
The `human.base.clean` are the RMA normalized experiments for each of the cell lines. Only probesets that match to known genes were kept. There is an `hmap` object that was loaded from `human.affymap` that has a mapping from affy probeset IDs to more recognizable gene symbols.

```
> head(hmap)
```

```
         affy     gene         ensembl
1 211942_x_at  SNORD34 ENSG00000202503
2 208729_x_at    HLA-H ENSG00000218492
3   228238_at  SNORD81 ENSG00000200710
4 227517_s_at  SNORD81 ENSG00000200710
5   204355_at MIRN1226 ENSG00000221585
6 210983_s_at   MIRN25 ENSG00000207547
```

Before we go any further, we may want to ask ourselves if there are any obvious patterns in our data that we can pick out "by eye." To try this, we can cluster the genes based on their expression profiles across the datasets to see if any visible patterns arise. We also want to check for "normalization" issues, not in the sense of low-level normalization but the fact that we are looking at quite different cell cultures. Let us start by looking at a random sample of 1000 genes using unscaled expression values (you can do more, if you'd like to exercise your computer a bit):

```
> set.seed(123)
> all.data <- cbind(gfp.exp, myc.exp, ras.exp, e2f3.exp)
> idxs <- sample(1:dim(gfp.exp)[1], 1000)
> heatmap(all.data[idxs, ], scale = "none", Colv = NA)
```



But let's make this look more like a microarray.

```
> palette.gr.marray <- colorRampPalette(c("green", "black", "red"))(256)
> palette.two.colorRamp <- colorRampPalette(c("black", "red"))(256)
> heatmap.2(all.data[idxs, ] - rowMeans(all.data[idxs, ]),
            scale = "row", dendrogram = "none", trace = "none",
            col = palette.gr.marray, symbreaks = TRUE, symkey = TRUE)
```

We used the `set.seed` call at first so that we all pick the same random values for the `idxs` vector and our results can be comparable to each other.

Playing with different values of the `scale` parameter in the `heatmap.2` function will show your data in different (perhaps surprising) ways. For now, using "row" makes a bit more sense when you just want to visualize your experiments against each other. Other possible values for `scale` can be

"`none`" or "`col`". If you'd like to talk about the differences in these options, please feel free to ask.

It seems as if we can identify some distinct block structure in our heatmap (especially some genes in the ras experiments), but by and large, the myc, e2f3 and gfp data sets don't seem *too* different and it's not clear what genes are useful in identifying cancer cells vs. control. Here is where machine learning steps into the picture.

2.1 SVM for Classification

In order to better isolate which genes we can use to classify one cancer from another, we'll build three separate SVM classifiers (one for each induced cell type). Our goal is to build an SVM such that it will be able to label a given microarray as coming from one of the three oncogenic-cells line or not. This type of classifier is known as a *binary* classifier.

The SVM will use the expression of each of the p probes on the microarray as a p-dimensional feature vector. Given that an SVM is a *supervised learning* method, we need to split our data into a training set and a testing set. We've put together the `prep.data` function that takes care of the tedium of these details for you. Pass it the two datasets, and it'll send you back a list structure with them "bound" together in the `$data` slot, as well as indices into the columns of the matrix that will be used for training and testing (`$train.idxs` and `$test.idxs`, respectively). If you like, you can also try to do this manually.

Let's prepare the data so we can build an SVM. We will need to separate our data into a training set, which we will build the model from, and a testing set which we will use to verify how good our model can predict on new data:

```
> training <- cbind(gfp.exp[, 1:7], myc.exp[, 1:7])
> testing <- cbind(gfp.exp[, 8:10], myc.exp[, 8:10])
```

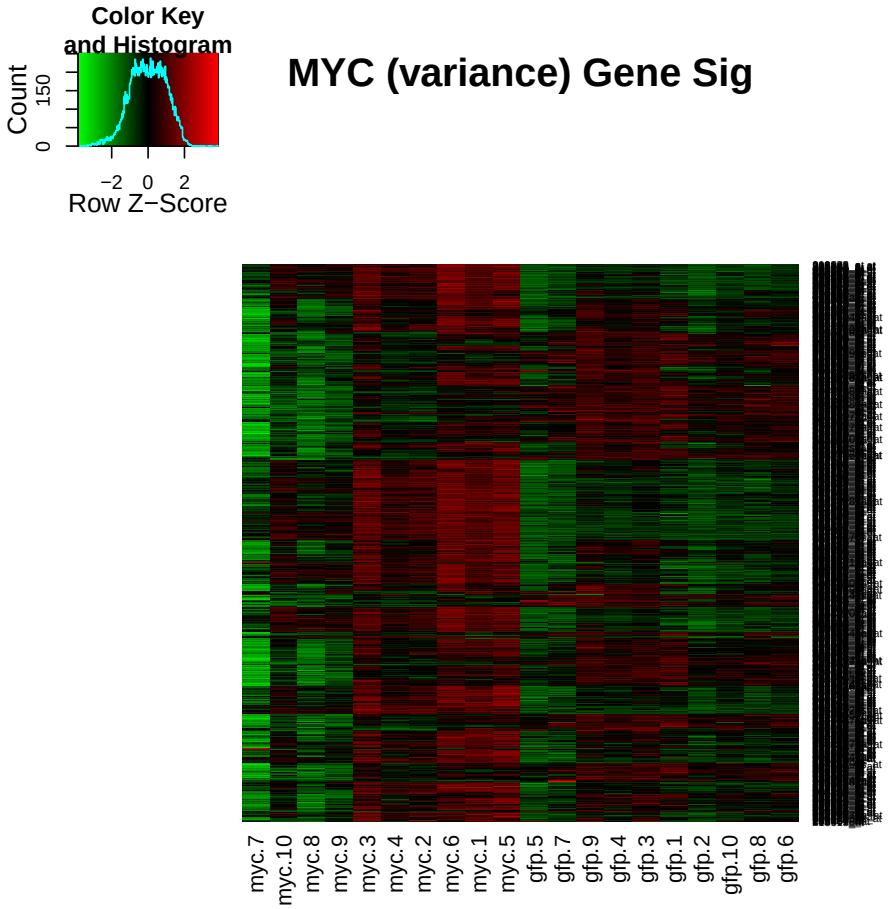
By default, the `prep.data` function will split the data in half: 5 arrays of GFP and 5 arrays of Ras data are used for training, and another 5+5 are held out to test the accuracy of the SVM. Given that we are starting with 54,675 such probes, and many of them may not vary too much, we can filter out some "uninteresting" probes by removing those with low variance in our training data. This process is referred to as "feature selection." Below we select the top 1000 genes based on their variance.

```
> train.variance <- apply(training, 1, var)
> most.varying <- order(train.variance, decreasing = TRUE)
```

```
> n.to.keep <- 1000
> train.d <- training[most.varying[1:n.to.keep], ]
> test.d <- testing[most.varying[1:n.to.keep], ]
```

Was *this* enough to help us distinguish which genes would be useful in a gene signature? Let's see:

```
> heatmap.2(cbind(train.d, test.d), scale = "row", dendrogram = "none",
            trace = "none", col = palette.gr.marray,
            main = "MYC (variance) Gene Sig", symbreaks = TRUE,
            symkey = TRUE)
```



Seems like a reasonable "cancer signature" already, doesn't it? We can see which genes we're looking at like so:

```
> var.30 <- subset(hmap, affy %in% rownames(train.d)[1:20])$gene
> as.character(var.30)

 [1] "RRP15"    "HSPA7"    "HSPA7"    "HSPA6"    "HSPA6"    "S100A8"
 [7] "MMP3"     "MMP1"     "KCNQ5"    "C12orf24" "SLC6A15"  "RARA"
[13] "MYC"      "PIGW"     "CYP2U1"   "HSPA1A"   "HSPA1A"   "TMEM97"
[19] "TMEM97"   "LAMP1"    "TMEM33"   "HSPA1B"   "HSPA1B"   "RRP15"
[25] "HSPA7"    "HSPA7"    "HSPA6"    "HSPA6"    "S100A8"   "MMP3"
[31] "MMP1"     "KCNQ5"    "C12orf24" "SLC6A15"  "RARA"     "MYC"
[37] "PIGW"     "CYP2U1"   "HSPA1A"   "HSPA1A"   "TMEM97"   "TMEM97"
[43] "LAMP1"    "TMEM33"   "HSPA1B"   "HSPA1B"
```

Let's see what the SVM can do.

We'll start by building the classifier by using the same 1000 most-varying genes. Note that in the `svm` call below, we are passing in the transpose of our `training.data`. The `svm` function expects each row to be an observation, and the columns are the corresponding features.

```
> library(e1071)
> train.labels <- factor(c(rep("gfp", 7), rep("myc", 7)))
> test.labels <- factor(c(rep("gfp", 3), rep("myc", 3)))
> myc.model <- svm(t(train.d), train.labels, kernel = "linear",
                   scale = TRUE, type = "C-classification")
```

Having used some of our data to learn a `myc.model`, we can use it to predict on the reaming left out data (aka testing data) to see how accurately it can classify as GFP or MYC-induced. You'll see that the SVM can classify such previously unseen data with 100% accuracy.

```
> myc.preds <- predict(myc.model, t(test.d))
> myc.acc <- sum(myc.preds == test.labels)/length(test.labels)
> cat("Testing accuracy: ", myc.acc * 100, "%\n", sep = "")

Testing accuracy: 100%
```

2.2 Cross Validation

Before we move on, we should probably try to verify how good our classifier is by doing train/test cycles on different splits of our data. Maybe we got lucky and the two experiments we held out weren't difficult to classify. This process is known as cross validation.

The `createFolds` function (from the `caret` library) takes a vector of labels (for classification, the labels should be `factor`s) and a value `k`. The `k` parameter tells the function how many folds you want to create. The function will return you a `list` that is `k` elements long, and each element has the indices you should hold out for testing for that fold.

```
> svm.data <- cbind(gfp.exp, myc.exp)
> svm.data <- svm.data[1:n.to.keep, ]
> svm.labels <- factor(c(rep("gfp", 10), rep("myc", 10)))
> test.idxs <- createFolds(svm.labels, k = 5)
> test.idxs

$`1`
[1]  2  8 15 19

$`2`
[1]  3  5 13 16

$`3`
[1]  1 10 17 18

$`4`
[1]  4  7 12 14

$`5`
[1]  6  9 11 20
```

We can now build 5 models on the different splits of the data to thoroughly test to further test how strong our classifier is.

```
> CV <- list()
> for (i in 1:length(test.idxs)) {
    cat("Performing cross validation fold:", i, "\n");
    cat("... holding out experiments", paste(test.idxs[[i]],
        collapse = ","), "\n");
    holdout <- test.idxs[[i]];
    train <- svm.data[, -holdout];
    test <- svm.data[, holdout];
    train.labels <- svm.labels[-holdout];
    test.labels <- svm.labels[holdout];
```

```
    model <- svm(t(train), train.labels, kernel = "linear", scale = TRUE,
                 type = "C-classification");
    preds <- predict(model, t(test));

    acc <- sum(preds == test.labels)/length(test.labels);
    cat(sprintf("   Testing accuracy in fold %d: %.2f%%\n", i,
        myc.acc * 100));
    CV[[i]] <- list(model = model, preds = preds);
}
```

```
Performing cross validation fold: 1
... holding out experiments 2,8,15,19
   Testing accuracy in fold 1: 100.00%
Performing cross validation fold: 2
... holding out experiments 3,5,13,16
   Testing accuracy in fold 2: 100.00%
Performing cross validation fold: 3
... holding out experiments 1,10,17,18
   Testing accuracy in fold 3: 100.00%
Performing cross validation fold: 4
... holding out experiments 4,7,12,14
   Testing accuracy in fold 4: 100.00%
Performing cross validation fold: 5
... holding out experiments 6,9,11,20
   Testing accuracy in fold 5: 100.00%
```

[Note: A more idiomatic way to write the code above would be to replace the for loop with a call to `lapply`]

Looks like our classifier is quite good: we have 100% accuracy over all folds! How many support vectors are we using to get these predictions?

```
> for (i in 1:length(CV)) {
    cat("Fold", i, "has ", CV[[i]]$model$tot.nSV, "support vectors (out of",
        ncol(svm.data) - length(test.idxs[[i]]), "examples)\n")
}
```

```
Fold 1 has  16 support vectors (out of 16 examples)
Fold 2 has  16 support vectors (out of 16 examples)
Fold 3 has  16 support vectors (out of 16 examples)
Fold 4 has  16 support vectors (out of 16 examples)
Fold 5 has  16 support vectors (out of 16 examples)
```

2.3 SVM for Gene Selection

We have seen how we can build a classifier for our separate cell-lines, but how does this help us get a better gene signature? We can first recover the examples that constitute the support vectors (and therefore define our w vector) from our data like so (we're simply listing the first 10 genes):

```
> myc.model$SV[, 1:10]
```

```
      X213418_at X202431_s_at   X204475_at X205828_at X202581_at X1565358_at
gfp.2 -1.0621492   -1.0203452 -0.534639008 -0.2562175 -1.6304133  0.81867017
gfp.3 -0.9214670   -1.0351475  1.491024734  1.3565050 -0.2740430  0.05268465
gfp.6 -0.9634217   -0.9568695  0.286374303  0.1153886 -1.1690274 -0.39933452
myc.1  0.9936407    0.9404194  0.007766504  0.2286362  1.0406577  0.31815946
myc.4  0.8985204    0.9562283 -1.075313842 -1.0582481  0.8523852 -0.42130920
myc.5  1.0149401    0.9652553 -0.345146160 -0.4324595  1.1771333 -1.30115112
myc.7  0.6207995    0.8627204 -0.470589692 -0.2832602  0.0070395 -0.63766172
      X200800_s_at X202917_s_at X212281_s_at  X200799_at
gfp.2   -1.4027830  -0.25311035   -1.2365600 -1.53905816
gfp.3   -0.5843035   1.17104609   -0.5022089 -0.44039857
gfp.6   -1.2326418   0.45218405   -0.6377080 -1.43738817
myc.1    0.9435978   0.18055037    1.1754115  0.95238667
myc.4    0.7641054  -1.32665991    0.7531746  0.74361497
myc.5    0.9783827   0.05903693    1.1106306  1.00040628
myc.7    0.4999018  -0.25100778    0.1508851  0.07380023
```

The following formula allows us to recover the w vector (which defines the decision boundary), where α gives us the weights of our features:

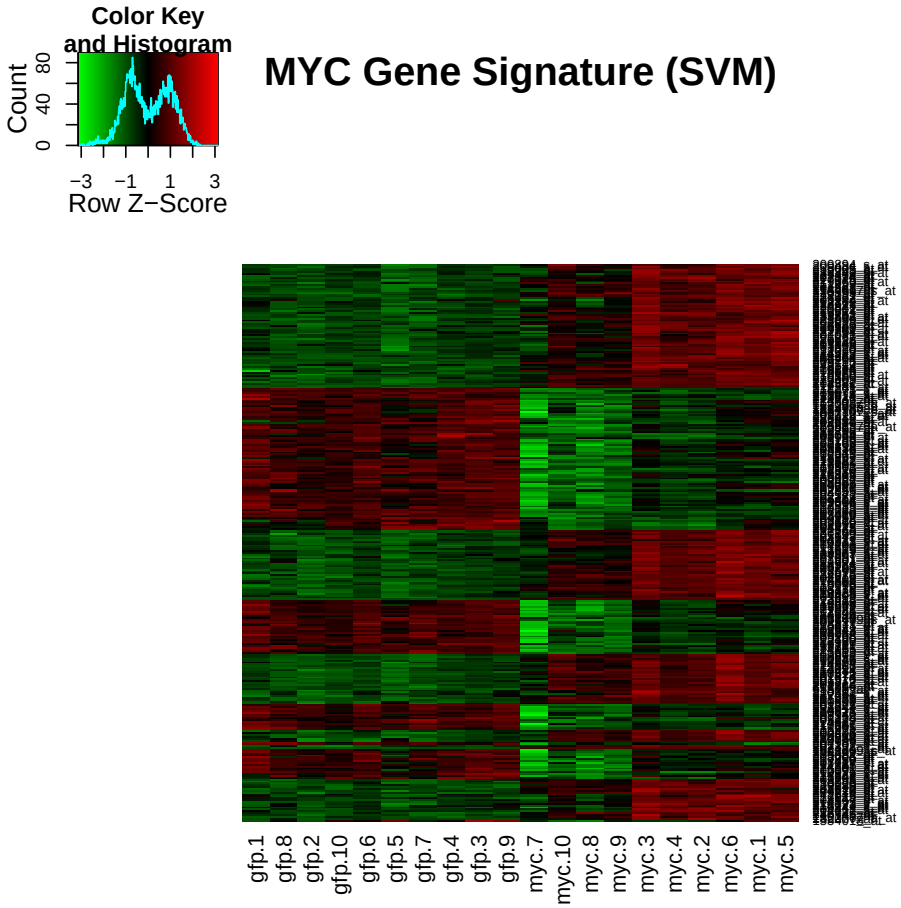
$$\vec{W} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i \tag{1}$$

N is the number of support vectors in our model, α is the weight of the support vector, y is the class label of the support vector, and $\vec{x}$ is the expression values of the probes for a given array in the training set. We can translate this expression into R rather easily by using matrix multiplication. $myc.model\$coefs$ gives us the value of our α's. Once we have our weight vector $\vec{w}$, we can then sort our features by their weights and pick the top M that we want to use as a signature.

```
> myc.weight.vector <- t(myc.model$coefs) %*% myc.model$SV
> myc.widxs <- order(abs(myc.weight.vector), decreasing = TRUE)
```

The `myc.widxs` vector now has *the indices* into the rows (probes) of our data that have the highest "weights". We can use these ordered features as our gene signature and examine what the *myc*-signature looks like using the top 300 features:

```
> heatmap.2(cbind(train.d, test.d)[myc.widxs[1:300], ], scale = "row",
            dendrogram = "none", trace = "none", col = palette.gr.marray,
            main = "MYC Gene Signature (SVM)", symbreaks = TRUE,
            symkey = TRUE)
> svm.30 <- subset(hmap, affy %in% rownames(train.d)[myc.widxs[1:30]])
> svm.30 <- as.character(svm.30$gene)
```



You'll find that the top 30 genes we found from filtering with the variance across the data are not the same as the top 30 genes the SVM found:

```
> intersect(var.30, svm.30)
```

```
[1] "HSPA7" "HSPA6" "MYC"
```

Is one better than the other? You can try to do GO enrichment testing for both sets of genes and see if we can find an "easy" answer to that question. Although this is out of the scope of this lab, please ask if you'd like help to do this GO analysis in R.

3 PCA

Principal Component Analysis (PCA) is a useful statistical technique for identifying patterns in data of high dimension. PCA expresses the data with principal components to highlight similarities and differences in the data.

After identifying gene signature sets for all three cancer cell lines, we hope to classify cancer samples using those gene signatures. The gene expression values of each signature are extracted from all cancer samples and then PCA is performed on samples to explore similarities and differences between samples. To find out how samples are separated after PCA, we display samples in respect to the first three components in a 3D scatterplot.

In order to do this, we'll need our SVM gene signature. The previous section has hopefully showed you that you can build an SVM that's accurate, and we tested as best we could to ensure that we aren't overfitting our data. Now that we want to use this technique for further downstream analysis, let's use *all* of the data we have available. So now we're going to build a MYC gene signature by using an SVM to predict MYC vs. not-MYC (ie. myc vs (gfp, ras, e2f3)).

```
> myc.data <- cbind(myc.exp, gfp.exp, ras.exp, e2f3.exp)
> myc.labels <- c(rep("Myc", 10),
                  rep("notMyc", ncol(myc.data) - 10))
> myc.labels <- factor(myc.labels)
> big.model <- svm(t(myc.data), myc.labels, kernel = "linear",
                   scale = TRUE, type = "C-classification")
> big.weight.vector <- t(big.model$coefs) %*% big.model$SV
> big.widxs <- order(abs(big.weight.vector), decreasing = TRUE)
```

```
> svm.big.30 <- subset(hmap, affy %in% rownames(myc.data)[big.widxs[1:30]])
> svm.big.30 <- as.character(svm.big.30$gene)
```

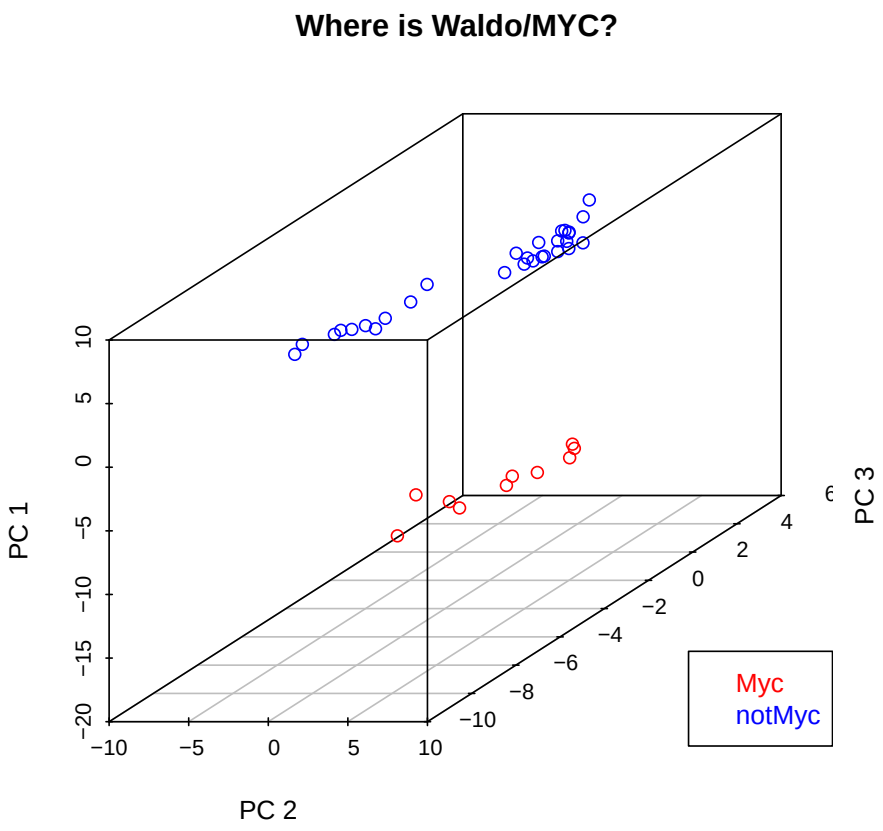
Are there any overlapping genes in our gene signature between this model trained from all the data vs. the gene signature from before?

```
> intersect(svm.big.30, svm.30)
```

```
[1] "TFB2M"   "HSPA7"   "HSPA6"   "SLC12A8" "MYC"     "SLC19A1" "ARNTL"
[8] "SLC1A3"  "TAF4B"
```

Now that we have our super MYC gene signature, let's extract the top 1000 genes as its gene signature. We'll do PCA on our the genes that define our signature, and see what this data our samples look like when we project it down to the top 3 dimensions after PCA.

```
> library(scatterplot3d)
> myc.sig.idxs <- big.widxs[1:1000]
> myc.sig.data <- myc.data[myc.sig.idxs, ]
> pca <- prcomp(t(myc.sig.data))
> loadings <- pca$x[, 1:3]
> s.colors <- c(rep("red", sum(myc.labels == "Myc")),
                rep("blue", sum(myc.labels != "Myc")))
> scatterplot3d(loadings[, 2], loadings[, 3], loadings[, 1],
                color = s.colors, xlab = "PC 2", ylab = "PC 3",
                zlab = "PC 1", grid = TRUE, box = TRUE,
                main = "Where is Waldo/MYC?")
> legend("bottomright", c("Myc", "notMyc"),
         text.col = c("red", "blue"))
```



Go ahead and try this approach to create PCA plots for your RAS and E2F3 signatures. You might want to try doing this PCA analysis using a gene signature you get from simply taking the top 1000 most varying genes across your data to see how it compares.

4 Human vs Mouse

To demonstrate the clinical relevance of these classifiers, we show that classifiers trained to recognize oncogenic pathways in human cell culture data can also accurately classify tumor data from related mouse models. Note here that the microarray platforms for human and mouse data are necessarily different. Following Bild et al., we restrict to probes for genes common

to mouse and human and use this reduced feature set for SVM training [1].

We've curated the human and mouse datasets for you into expression matrices so that each row in the matrix is for a gene that is "in" both human and mouse, and are for affy probesets that have "easy" mappings back to genes. The rows of the mouse and human experiment data are also in the same order (important!). This data is in the `common.data.clean` data package.

```
> load("common.data.clean.rda")
```

Let's rebuild our myc vs. all classifier on the human data and see how well we can predict the expression of our myc expression set.

```
> human.data <- cbind(human.myc.exp, human.gfp.exp, human.ras.exp,
                      human.e2f3.exp)
> human.labels <- c(rep("Myc", 10),
                    rep("notMyc", ncol(human.data) - 10))
> human.labels <- factor(human.labels)
> human.model <- svm(t(human.data), human.labels, kernel = "linear",
                     scale = TRUE, type = "C-classification")
> human.weight.vector <- t(human.model$coefs) %*% human.model$SV
> o.weights <- order(abs(human.weight.vector), decreasing = TRUE)
> human.30 <- subset(hmap, affy %in% rownames(human.data)[o.weights[1:30]])
> human.30 <- as.character(human.30$gene)
```

Now that we've trained the human myc classifier, let's see if our human myc classifier can predict which experiments in our mouse data come from "myc-like" mouse cancer models.

```
> mouse.data <- cbind(mouse.myc.exp, mouse.her2.exp, mouse.ras.exp,
                      mouse.rbnull.exp, mouse.wt.exp)
> mouse.labels <- c(rep("Myc", 5),
                    rep("notMyc", ncol(mouse.data) - 5))
> mouse.labels <- factor(mouse.labels)
```

There are two `Inf` values in our data, we need to do something with these, else our svm will choke. Let's set them to the mean value of all of the expression levels we have.

[1] The authors provide a tool to map human affymetrix probes to the homologous probes on the mouse affymetrix chip. You can also do this yourself using the bioconductor package `annotationTools`. The vignette for this package shows you how to do that.

```
> inf.data <- is.infinite(mouse.data)
> mouse.data[inf.data] <- mean(mouse.data[!inf.data])
> preds <- predict(human.model, t(mouse.data))
> acc <- sum(preds == mouse.labels)/length(mouse.labels)
> cat("SVM Accuracy: ", acc * 100, "%\n", sep = "")
```

```
SVM Accuracy: 100%
```

Instead of doing "pure" classification, the `svm` function has a `probability` parameter that up until now has defaulted to `FALSE`. We can set it to `FALSE` and recover score that reflects the probability of an example being in one class or another.

```
> human.model.probs <- svm(t(human.data), human.labels, kernel = "linear",
                           scale = TRUE, type = "C-classification",
                           probability = TRUE)
> mouse.probs <- predict(human.model.probs, t(mouse.data), probability = TRUE)
> prob.order <- order(attr(mouse.probs, "probabilities")[, "Myc"],
                      decreasing = TRUE)
> experiment.labels <- c(rep("Myc", ncol(mouse.myc.exp)),
                         rep("Her2", ncol(mouse.her2.exp)),
                         rep("Ras", ncol(mouse.ras.exp)),
                         rep("RbNull", ncol(mouse.rbnull.exp)),
                         rep("WT", ncol(mouse.wt.exp)))
> experiment.labels[prob.order]
```

```
 [1] "Myc"    "Myc"    "Myc"    "Myc"    "Myc"    "Ras"    "Ras"    "Ras"
 [9] "Her2"   "RbNull" "RbNull" "WT"     "RbNull" "WT"     "WT"     "RbNull"
[17] "RbNull" "Her2"   "Her2"   "WT"     "Her2"   "WT"     "WT"     "RbNull"
[25] "Her2"   "Her2"   "WT"     "Her2"
```

It might be interesting to note that even though our classifier has never seen data from RbNull or Her2 models, they somehow get "grouped together", anyway.

5 Trivia

The genes in normal microarray data are sill somehow "raw" in that they are listed with they affymetrix probe set IDs. At some point you might like to figure out what genes they actually refer to, eg. that probeset `231640_at` on the affymetrix hgu133plus2 array is probing the expression of k-RAS.

We can get a list of these mappings from the ensembl database, which is accessible from within bioconductor itself, too. In order to query the database, we'll need to use the `biomaRt` package.

```
> library(biomaRt)
> hmart <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")
> haffy.ids <- rownames(gfp.exp)
> want <- c("affy_hg_u133_plus_2", "hgnc_symbol", "ensembl_gene_id")
> hmap.2 <- getBM(attributes = want, filters = "affy_hg_u133_plus_2",
                  value = haffy.ids, mart = hmart)
```

References

[1] Andrea H. Bild, Guang Yao, Jeffrey T. Chang, Quanli Wang, Anil Potti, Dawn Chasse, Mary-Beth Joshi, David Harpole, Johnathan M. Lancaster, Andrew Berchuck, Jeffrey R. Olson, Holly K. Dressman, Mike West, and Joseph R. Nevins. Oncogenic pathway signatures in human cancers as a guide to targeted therapies. *Nature*, 439, 2006.

[2] Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch, David Meyer, , and Andreas Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2008. R package version 1.5-18.

[3] Laurent Gautier, Leslie Cope, Benjamin M. Bolstad, and Rafael A. Irizarry. affy—analysis of affymetrix genechip data at the probe level. *Bioinformatics*, 20(3):307–315, 2004.

[4] Robert C Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[5] Mike West, Carrie Blanchette, Holly Dressman, Erich Huang, Seiichi Ishida, Rainer Spang, Harry Zuzan, Jr. Olson, John A., Jeffrey R. Marks, and Joseph R. Nevins. Predicting the clinical status of human breast cancer by using gene expression profiles. *Proceedings of the National Academy of Sciences*, 98(20):11462–11467, 2001.