Machine Learning and Gene Expression Data

M. T. Morgan*

April 14, 2006

Overview

Many biological experiments investigate the relationship between gene expression patterns and phenotypes. Machine learning algorithms provide a tool for gaining insight into this relationship. This lecture introduces machine learning, the diversity of machine learning algorithms available, and methods for assessing and interpreting the result of machine learning algorithms in light of our fundamental interest in the relationship between gene expression and phenotype. An accompanying lab provides hands-on experience.

Introduction to machine learning

Machine learning refers to computation and statistical methods of inference used to create reusable algorithms for prediction. Let's use gene expression profiles and subtypes of acute lymphatic leukemia (ALL) as a running example. We might want to develop, with as little direct intervention as possible, algorithms that will take as input the gene expression array profile observed in a particular sample, and produce as output a classification of whether or not the sample came from cancerous tissue. Gene expression arrays provide an abundance of data; it is often convenient and statistically sound to *filter* the data prior to analysis using machine learning algorithms.

The word *machine* conveys the idea that both the process of specifying the algorithm and subsequent application of the algorithm for prediction should occur with a minimum of human intervention. There is, of course, some human intervention. We have to provide general guidelines about

*based on previous labs by R. Gentleman, W. Huber, V. Carey, R. Irizarry the nature of the algorithm we wish to use (e.g., *neural networks*, *support vector machines*, *classification and regression trees*). We have to provide the algorithm with data to serve as a *training set*, so that the algorithm can automatically adjust parameter values for the particular type of data we are analyzing. Once trained, we normally also use a *test set* of data to evaluate algorithm performance.

Our focus is on *supervised* machine learning. Supervised learning means that some *a priori* information is available to guide the learning process. For instance, in the training set we might know whether the sample tissues have unique molecular genetic characteristics such as the BCR/ABL centric fusion. Supervised machine learning algorithms use this *a priori* information to automatically adjust parameters of the algorithm so that the predicted outcomes most closely match the *a priori* classification.

Supervised machine learning methods contrast with *unsupervised* methods (e.g., cluster analysis) where no *a priori* information on outcome is available and the main goal is to impose a structure that minimizes some overall measure of 'distance' between sample features.

Data and filtering

An important step in machine learning is to identify an appropriate sample to be used as a training set. The training set should be representative of the problem. It should be computationally tractable so that we feel free to explore our data without running into computational time or memory constraints. With gene expression data we likely want to take preliminary steps to reduce the number of genes included in our analysis. We might do this by filtering on various criteria like expression levels above a certain threshold, sufficient variation between samples, and differential expression between *a priori* classifications. We might also restrict our analysis to subsets of genes that are particularly interesting in the context of our research agenda, such as genes belonging to a particular GO category.

The accompanying lab takes us through some of these steps in more detail.

Linear machine learning algorithms

There are a great diversity of machine learning algorithms available. Mathematically simplest are *linear* methods such as linear discriminant analysis. Linear machine learning methods are reminiscent of linear regression, where prediction is based on linear combinations of observed features x and weight parameters w_0, w, e.g.,

$$g(x) = w_0 + w^T x \qquad (1)$$

In this expression, x corresponds to a sample. It is a vector with number of elements equal to the number of measured features. The vector of weights w are parameters determined during the training process (T represents transposition). The parameter w_0 is used to adjust the threshold for classification, so that values of $g(x)$ greater than zero indicate classification as one type, values less than zero correspond to classification as an alternative type. Though phrased as classification into one of two types, machine learning algorithms generally allow classification into many different types. Note that 'linear' refers to the linear combination of features; log-transforming values of x and then using the expression above still represents a linear algorithm.

A supervised machine learning algorithm is trained by providing a collection of samples and determining weights w, w_0 that perform 'best' at classifying the samples to their *a priori* types. Once trained, a machine learning algorithm applies classification to data for which prior information is not available. Linear machine learning algorithms provide a sense of how data, algorithms, and estimated parameters combine to classify samples.

Additional machine learning algorithms

Linear models like that in eq. 1 are the simplest machine learning algorithms. There are many other machine learning algorithms, and we present a brief overview of some of them here. A list of functions provided by the MLInterfaces package is provided in Table 1; Figure 16.1 illustrates some of the methods for two arbitrarily chosen features (genes ORB6B1, olfactory receptor 6 B1, and SST, somatostatin).

As their name suggests, *non-linear* models relax the assumption that features combine linearly to determine classification. Neural networks (illustrated in figure 16.2) are one commonly used non-linear model. Neural networks consist of a series of input nodes (representing features) connected through several layers to output nodes (representing predicted classification). Each node has several inputs and one output. Each node has a mathematical function that describes how the inputs are to be transformed to the output. The function can be as simple as a weighted average of inputs, analogous to $w^T x$ but specific to each node. The parameters of the functions at each node are adjusted during training. An additional adjustable parameter is the contribution or weight associated with the output of one node as it contributes to the next layer.

	Package	Functions covered
1	class	*knn1, knn.cv, lvq1, lvq2, lvq3, olvq1, som SOM*
2	cluster	*agnes, clara, diana, fanny, silhouette*
3	e1071	*bclust, cmeans, cshell, hclust, lca naiveBayes, svm*
4	gbm	*gbm*
5	ipred	*bagging, ipredknn, lda, slda*
6	MASS	*isoMDS, qda*
7	nnet	*nnet*
8	pamr	*cv, knn, pam, pamr*
9	randomForest	*randomForest*
10	rpart	*rpart*
11	stats	*kmeans*

Table 1: Packages and functions covered by MLInterfaces.

One use of neural networks is to start by specifying an overall structure (e.g., functional forms for calculating each node) of the network. During training, parameters of the neural network are adjusted to most closely match predicted with actual classifications. New samples are then classified by performing the calculations implied by the structure of the neural network and parameter by the values determined during training.

Regularized models such as support vector machines.

Local models develop classifiers that differ from one another based on the location of data in feature space, and make predictions by averaging the prediction of near-by classifiers; the k nearest neighbor ($k$nn) method is an example of a local classifier. Suppose we have two features. We create a two-dimensional plot, and color the points corresponding to each sample in our training set according to their classification, e.g., negative peach, BCR/ABL positive turquoise. We classify new observations by locating the new sample on the plot and constructing some kind of average based on the k nearest neighbors in the training set. With $k = 1$, we predict that each new sample belongs to the same classification as the nearest point in the training set. This allows us to divide the feature space into regions corresponding to classification as BCR/ABL, and to NEG.

Tree-based models such as classification and regression tree (CART) methods identify a decision tree-like structure that performs classification by partitioning the data at each node based on feature values (see Figure 16.3).

The overall structure of the tree is specified by how branching occurs (usually each node has two branches) and a function that describes the trade-off between greater precision and reduced accuracy of the classification (roughly, increasing the number of nodes in the tree increases precision, because each sample in the training set is assigned to its own node, but decreases accuracy, because the parameters associated with each branch are based on increasingly sparse amounts of information). Training identifies the structure of the tree, and the cut-off thresholds at each node. To classify a sample, start at the top of the tree. A sample below the threshold of the first node follows the left branch, above the threshold the right branch. This repeats until the terminal node, which represents the sample classification.

Cross-validation and assessing model fit

The diversity of machine learning algorithms provides us with an abundance of outcomes – perhaps an overwhelming abundance! The summary in Figure 16.1 illustrates both properties of different classifications (e.g., the division of predicted classifications into 'boxes' in the CART model) and differences in possible outcomes. How are we to identify which classification is best?

As a preliminary to answering this question, note that we need to carefully consider the meaning of 'best'. For instance we might be interested in maximizing our ability to correctly classify new samples, or to minimize the uncertainty of each classification. These different 'best' measures are not always consistent with one another. Here we focus on correctly classifying new samples.

There is a fundamental problem in measuring how good our machine learning algorithm is at classifying new samples. New samples are not already classified, so we cannot know when our classification is correct.

The solution to this quandary is to use *cross-validation*. We divide our samples previously used for training into two groups. One will be a training set used to construct the machine learning classifier. The other will be a test set used to evaluate the effectiveness of the classifier. We have *a priori* knowledge of the classification of the samples in the test set, so know when our trained machine learning algorithm results in the correct classification. A single cross validation provides us with one observation about how our algorithm performs, e.g., the number of correct assignments in the test set.

The next step in cross-validation is a little non-intuitive, but the underlying concepts might be familiar to those with exposure to jackknife or bootstrap techniques. A key idea is that the assignment of samples to test and training sets sketched in the previous paragraphs was in some sense arbitrary – we could have assigned different samples to different training sets, and come up with a different observation of how our machine learning algorithm preforms. In fact, we could repeatedly perform this cross-validation, assigning different samples to the test and to the training set each time. The result of each cross-validation represents a sample measuring the performance of the machine learning algorithm. The collection of samples from repeated cross-validation is analogous to samples drawn from a population – the population of machine learning algorithms generated from our data – and so we can use these observations to calculate statistics that summarize the overall performance of our machine learning algorithm.

How should we divide our data into training and testing sets? On the one hand, we want to use as many samples as possible in the training set, so that our machine learning algorithm is based on a broad range of samples. On the other hand, we need at least some samples in the test set to evaluate the performance of the trained algorithm. Frequently, we divide our data into a *leave-one-out* cross-validation. We assign all but one observation to the training set, so that the trained machine learning algorithm is based on the most available data. We assess the performance of the trained machine learning algorithm with the single remaining sample in the test set. Each cross-validation 'test' seems weak – just a single sample subject to test, classified either correctly or incorrectly. But if we have $n = 75$ samples to work with, then we perform $n = 75$ leave-one-out cross-validations, and have $n = 75$ observations with which to assess the performance of our machine learning algorithm.

The results of leave-one-out cross-validation can be summarized in a table, or with basic statistics summarizing correct classifications. Statistics generated from different machine learning algorithms can then be compared either informally (algorithm X has numerically higher correct classification, on average, than algorithm Y) or through more formal methods.

There are methods other than leave-one-out cross-validation, and methods other than cross-validation for the assessing machine learning algorithms.

Using trained machine learning algorithms, and beyond

The discussion so far brings us to the point where we have a trained and tested machine learning algorithm. We can use this to classify new samples. We can identify properties of our algorithm that might be useful in subsequent studies, or that might suggest aspects of our data that we were not previously aware of. We can also use machine learning algorithms in creative ways different from the basic scenario outlined here, of classifying samples based on gene expression profiles, to explore our data.

Feature selection identifies those features or feature combinations that are particularly important in successful classification. There are a diversity of ways to assess feature importance. One method is to perform a cross-validation, with the test set consisting of several samples. Classify the test set and record the number of correct classifications. Then randomly permute one feature across the test samples, redo the classification, and note again the number of correct classifications. Repeat this for each feature, and for many cross-validations. The idea is that permuting important features will greatly reduce the number of correct classifications, whereas permuting unimportant features will have little consequence on correct classification. The outcome is a ranking of features and their importance, such as that in Figure 16.4.

One example of a different, creative, use of machine learning to explore expression data is in the edd package. This package allows us to ask what the relationship is between gene expression levels and the phenotype of the samples. For a single gene, we might find that the NEG samples are best described as a bimodal distribution resulting from a mixture of two Gaussian (normal) distributions. Where does machine learning fit in? There are many different possible distributions or combinations of distributions that might describe gene expression patterns, and deciding that a bimodal distribution describes a particular genes' expression pattern represents a classification. The algorithm that classifies gene expression distributions is a machine learning algorithm, trained on simulated data sets representing combinations of the different distributions. Thus edd generates many simulated distributions of known composition as a training set. Once trained the machine learning classifier is used on the gene expression samples. We gain insight into the gene expression patterns of individual genes, of the collection of genes, and of contrasting gene expression patterns in different experimental groups.

Machine learning laboratory

Data

The lab will use data from the acute lymphatic lukemia (ALL) data set. We load the data and take a look at its structure:

```
> library(Biobase)
> library(ALL)
> data(ALL)
> ALL

Expression Set (exprSet) with
        12625 genes
        128 samples
                 phenoData object with 21 variables and 128 cases
         varLabels
                cod:  Patient ID
                diagnosis:  Date of diagnosis
                sex:  Gender of the patient
                age:  Age of the patient at entry
                BT:  does the patient have B-cell or T-cell ALL
                remission:  Complete remission(CR), refractory(REF) or NA. Derived fr
                CR:  Original remisson data
                date.cr:  Date complete remission if achieved
                t(4;11):  did the patient have t(4;11) translocation. Derived from ci
                t(9;22):  did the patient have t(9;22) translocation. Derived from ci
                cyto.normal:  Was cytogenetic test normal? Derived from citog
                citog:  original citogenetics data, deletions or t(4;11), t(9;22) sta
                mol.biol:  molecular biology
                fusion protein:  which of p190, p210 or p190/210 for bcr/able
                mdr:  multi-drug resistant
                kinet:  ploidy: either diploid or hyperd.
                ccr:  Continuous complete remission? Derived from f.u
                relapse:  Relapse? Derived from f.u
                transplant:  did the patient receive a bone marrow transplant? Derive
                f.u:  follow up data available
                date last seen:  date patient was last seen
```

Question: What sorts of measurements are included along with the expression data?

Question: What is the mean age of patients in the sample (hint: look at the help page for mean, and pay attention to 'NA' values)? The average female?

Question: Do you think it will be helpful to have more than just the expression data available in a single variable?

Filtering: samples

Our goal in filtering samples is to focus our initial efforts on a relatively simple problem. We start by restricting the number of molecular biological features under investigation. Identify those samples whose `mol.biol` level is BCR/ABL (fusion of the BCR and ABL genes) or NEG (no characterized molecular biology abnormalities).

```
> fusion <- ALL$mol.biol %in% c("BCR/ABL", "NEG")
```

Question: Explain, to yourself or someone nearby, what the previous statement is doing. What does `fusion` contain? Any guesses on how this might be used to select a subset of `ALL$mol.biol`?

Next we identify samples from patients with B-cell ALL. This done to further simplify the data for analysis.

```
> btLevels <- levels(ALL$BT)
> btLevels

 [1] "B"  "B1" "B2" "B3" "B4" "T"  "T1" "T2" "T3" "T4"

> isB <- ALL$BT %in% btLevels[1:5]
```

Question: What are each of these statements doing?

Question: How would you identify samples from patients with T-cell ALL?

Question: How might you combine the information in `fusion` and `isB` to produce something that indicates samples satisfying both conditions?

Having identified the samples we want to use, we make a subset of the data containing only those samples satisfying both conditions.

```
> ALLsubset <- ALL[, fusion & isB]
```

Question: Check that `ALLsubset` contains just the samples you expect.

Notice (in the first lines below) that in `ALLsubset$mol.biol` there are only two levels present in the data column, but that 6 levels are summarized. Correct this by recoding the factor:

```
> levels(ALLsubset$mol.biol)

[1] "ALL1/AF4" "BCR/ABL"  "E2A/PBX1" "NEG"      "NUP-98"   "p15/p16"

> ALLsubset$mol.biol
```

```
 [1] BCR/ABL NEG     BCR/ABL NEG     NEG     NEG     NEG     NEG     BCR/ABL
[10] BCR/ABL NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL BCR/ABL BCR/ABL
[19] NEG     BCR/ABL BCR/ABL NEG     BCR/ABL NEG     BCR/ABL NEG     BCR/ABL
[28] NEG     BCR/ABL BCR/ABL NEG     BCR/ABL BCR/ABL BCR/ABL NEG     BCR/ABL
[37] NEG     NEG     NEG     BCR/ABL BCR/ABL BCR/ABL NEG     NEG     NEG
[46] NEG     BCR/ABL BCR/ABL NEG     NEG     NEG     NEG     BCR/ABL NEG
[55] NEG     NEG     NEG     NEG     BCR/ABL BCR/ABL NEG     NEG     BCR/ABL
[64] BCR/ABL NEG     NEG     NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL
[73] BCR/ABL NEG     NEG     BCR/ABL NEG     BCR/ABL BCR/ABL
Levels: ALL1/AF4 BCR/ABL E2A/PBX1 NEG NUP-98 p15/p16
```

```
> ALLsubset$mol.biol <- factor(ALLsubset$mol.biol)
```

Question: Check that there are now only two levels in `ALLsubset$mol.biol`

Filtering: genes

To maximize the discriminatory ability of our classification algorithm, select the top 500 genes that show greatest differential expression between BCR/ABL and NEG samples

```
> library(limma)
> modelMatrix <- model.matrix(~ALLsubset$mol.biol)
> fit <- lmFit(ALLsubset, modelMatrix)
> eFit <- eBayes(fit)
> topGenes <- topTable(eFit, coef = 2, 500)
```

Question: What is a model matrix? Can you figure out what `~ALLsubset$mol.biol` represents (this is hard).

Question: What are IDs of the the top 5 genes? Any ideas about how to find out more information about these genes?

Finally, reduce our sample of genes to the top 500.

```
> ALLsubset <- ALLsubset[as.numeric(rownames(topGenes)), ]
```

Question: Explain the previous line of code.

Question: Check that ALLsubset contains the genes that you expect.

Machine learning: *k* nearest neighbors

We will use the MLInterfaces package. Load the library and explore its documentation:

```
> library(MLInterfaces)
```

Question: Use `library(help=MLInterfaces)`, `?"MLearn-methods"` and `openVignette()` to explore the package.

Question: Can you follow the example at the bottom of the `MLearn-methods` help page? Depending on the packages installed on your computer, you might have luck with the command `example("MLearn-methods")`.

A key function is `MLearn`. `MLearn` is designed for easy use with expression data. The first argument is the name of variable containing *a priori* classification information, e.g., `mol.biol`. The second argument is the `exprSet`, the third argument the name of the machine learning algorithm, and the fourth argument the individuals to be used for training. So to use the *k* nearest neighbors machine learning algorithm using the first 50 samples for training, do the following:

```
> knnResult <- MLearn("mol.biol", ALLsubset, "knn", 1:50)
> knnResult

MLOutput instance, method= knn
Call:
 knnB(exprObj = data, classifLab = formula, trainInd = as.integer(trainInd))
predicted class distribution:
BCR/ABL     NEG
     16      13
summary of class assignment quality scores:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

Question: Interpret each line of the input to *MLearn*. What would you do to change the training set? To use every second sample as the training set? To use all but the last sample for training? To use a training set of 50 individuals, chosen at random from the samples in `ALLsubset` (hint: use the `sample` function).

Question: Interpret the output of `MLearn`. In particular, look at the predicted class distribution and check that the right number of samples are being used for testing.

The confusion matrix compares the known classification of the testing set with the predicted classification based on the tuned machine learning algorithm.

```
> confuMat(knnResult)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      12   0
  NEG           4  13
```

Question: Interpret the confusion matrix. How well do you think the algorithm is doing? What might you do to improve the classification?

Question: What other information can you extract from the fitted model?

Machine learning: other methods

As an advanced exercise, explore other methods of machine learning. Some methods are available through the `MLearn` interface of the previous section. For other options you'll have to explore the vignettes available with the package and especially the help page `?allClass`.

Cross-validation

The MLInterfaces package has a method for performing cross-validation. The method is called `xval`. Ponder its help page (`?xval`) and think about how you might perform cross-validation of `ALLsubset`.

From the `xval` help page, it looks like we should be able to perform cross validation with a command like:

```
> knnXval <- xval(ALLsubset, "mol.biol", knnB, xvalMethod = "LOO")
```

The first two arguments should be familiar. The third argument, `knnB`, specifies that we will use the `knn` algorithm. The final argument, `xval-Method`, indicates the method that will be used for cross-validation. The cryptic `LOO` stands for leave-one-out. So...

Question: Describe in words the operation that xval is performing.

`xval` for leave-one-out cross-validation returns a list, with each element in the list being the result of a single cross validation.

Question: What is the length of `knnXval`? Why?

Question: Interpret the meaning of each element in `knnXval`.

Question: What information is provided by the following command? How would you use this to assess the performance of this machine learning algorithm?

```
> table(given = ALLsubset$mol.biol, predicted = knnXval)
```

```
         predicted
given     BCR/ABL NEG
  BCR/ABL      35   2
  NEG           5  37
```

Feature selection

As an advanced exercise, use `randomForest` on `ALLsubset` to construct and tune a *random forest* machine learning algorithm. The help page for `getVarImp` can be used to help formulate the appropriate commands. Consult chapter 16 of the book for guidance on interpretation.

As another advanced exercise, use the `edd` package and `ALLsubset` to explore how gene expression patterns differ between tissues. Use chapter 16 and the help and vignettes of `edd` to guide you.

Conclusions

We have covered a great deal of material in this lecture and lab, investigating how supervised machine learning provides insight into gene expression patterns. There are many machine learning algorithms available. A basic technique for assessing different algorithms is to use cross-validation on data divided into training and test sets. The result of machine learning can be an algorithm for classification. This can be used to classify new samples, or its properties can shed light on the relationship between gene expression and phenotype. R provides a very flexible environment for analyzing data in a machine learning context, with the MLInterfaces package being a particularly useful tool. We have only scratched the surface of machine learning, and of how machine learning can be used to gain insight into gene expression and other high-throughput biological data.