Code ▾

# Week 7 Notes Part 1 Model Prediction

Author: Brendan Gongol

Last update: 03 January, 2023

## 1 Overview

In this Note we discuss how to assess the predictive accuracy of a model. Fitting the model to the whole available dataset will definitely provide the most accurate description of the available data, but this can result in overfitting (the model parameters can be driven by noise and outliers in a particular realization of data we are dealing with), and it is difficult if not impossible to quantify how the model will perform on new, yet unseen data. Since (or rather when) we want to generalize the model and infer underlying dependence that we can use for prediction, the question of model robustness and predictive accuracy is very important. In this Note, we will start building the framework for such assessment by introducing model cross-validation and model bootstrapping approaches (similar to the resampling and bootstrapping approaches we have learned about in previous week).

## 2 Setting up the Model

In order to study the accuracy of model predictions, we will peruse the example of the dependence between gene expression and time-to-remission (we studied this example in Week 3). To keep things simple, we will consider one pre-selected gene for now (Affymetrix probe 34852_g_at). Here's the code we used to set up all the variables into the R session (copied from week 3), run it to initialize the data:

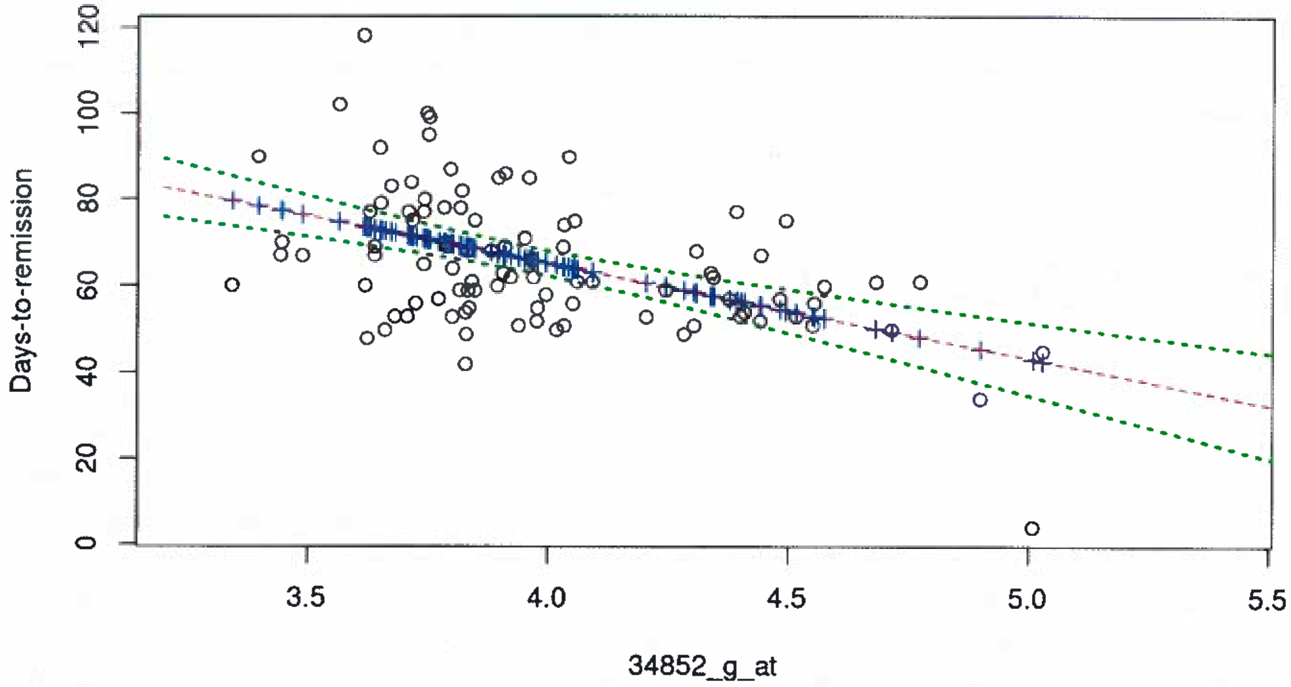
Hide

```
library(ALL); data(ALL)
ALL.pdat <- pData(ALL)
date.cr.chr <- as.character(ALL.pdat$date.cr)
diag.chr <- as.character(ALL.pdat$diagnosis)
date.cr.t <- strptime(date.cr.chr,"%m/%d/%Y")
diag.t <- strptime(diag.chr,"%m/%d/%Y")
days2remiss <- as.numeric(date.cr.t - diag.t)
x.d2r <- as.numeric(days2remiss)
exprs.34852 <- exprs(ALL)["34852_g_at",]
d2r.34852 <- data.frame(G=exprs.34852,D2R=x.d2r)
```

Let us now take look at the data and the fitted model (the code below should look familiar to you by now):

Hide

```
plot(d2r.34852[,c("G", "D2R")],
 xlab="34852_g_at",
 ylab="Days-to-remission") # plot the original data, D2R vs G
lm.34852.g.at <- lm(D2R~G,d2r.34852,na.action=na.exclude) # fit model to full data set
# draw predicted (fitted) values of D2R for each value in G:
points(d2r.34852$G,
 predict(lm.34852.g.at),
 col="blue",pch=3) # plots blue crosses at fitted values
# generate some 10 points within the range spanned by vector G:
x.tmp <- pretty(d2r.34852$G,10)
new.df.ci <- data.frame(G=x.tmp) # new data to make predictions for
pred.ci <- predict(lm.34852.g.at, # predict values and confint bounds
 new.df.ci,interval="confidence")
points(new.df.ci$G,pred.ci[,"fit"],
 col="red",type="l",lty=2) # plot the predicted values (red)
points(new.df.ci$G,pred.ci[,"lwr"],
 col="green",type="l",lty=3,lwd=2)
points(new.df.ci$G,pred.ci[,"upr"],
 col="green",type="l",lty=3,lwd=2) # plot confint bounds (green)
```



The only new parts in the code above are: 1) the convenience method `pretty()`: its purpose is to generate the requested number of points that (approximately) span the specified range and have some decent ("pretty") decimal representation (e.g. 1.2,1.3,.. etc, rather than 1.123765452); 2) the `predict()` function can work either on the linear model object itself (in which case it computes fitted values at the data points originally used to fit the model), or on a new dataframe explicitly passed to it – in the latter case, the predicted values are calculated for the values of the independent variable stored in the dataframe (x.tmp in the code above), using the specified (previously fitted on some data) model. In addition to the predicted values themselves, predict() can also generate lower/upper boundaries on those values using the confidence interval of the fit (you can examine in your R session what data the dataframe `pred.ci` contains). In our code we plot both the fitted values and the boundaries (green dashed lines), see the Figure below.

what we've been doing ↓

The code above sets up the problem for us and generates a visualization of both the original dataset and the fit. How good is this fit, and what does "how good" mean? We have learned in the past how to characterize the fit. We can examine t-test/anova p-values and confidence intervals of the coefficients: these metrics tell us how significant the fit is (with respect to the null hypothesis of no association between the variables in the original dataset we were trying to fit). We know that we can look at the magnitude of the residuals (sum of squares): this is actually what anova measures (relative to the total variation), as we remember. Finally, we can look at the diagnostic plots and try detecting any potential problems (non-homogeneous and/or non-normal distribution of residuals, few outliers the fit is hinging on, etc).

While extremely important, these metrics characterize how well we managed to fit (or "describe") the data at hand. However, when performing regression we are usually not looking to just summarize the data; our aim is to infer and generalize the underlying dependence and use it for prediction. In our example, we would like to be able to predict (with some finite accuracy, of course) the days-to-remission in a new, yet unseen patient based on her expression level of gene (Affy probe) 34852_g_at used as independent variable in our model. The goodness-of-fit definitely tells us something to that effect (poor fit probably won't predict well, and if the fit is not significant on the available data, there might be just no association in the first place, so we cannot use such model for prediction). However, this is not the whole story. The data may not satisfy all the assumptions we are making when running our significance tests and calculating confidence intervals; importantly, the fit (description of the data at hand!) can be also idiosyncratic and describe the limited data we have too well (the problem of overfitting: the fit to a particular realization of the data can capture noise that just accidentally looks like a dependence). This line of reasoning might have already made you guess that it is useful to perform some kind of resampling in order to assess the robustness of the model. In the following sections we will set up two simple frameworks for assessing an accuracy of a model using cross-validation and bootstrapping (the concepts that are already familiar to us).

# 3 Model Cross-Validation

In cross-validation approach, we split the data into a "training set" and "test" set. As the names suggest, we "train" the model (i.e. perform the regression) on a training set, and then assess the accuracy on the model using the new "yet unseen" test set. This is a very generic and universal approach. In general, when you build a predictive model (be that simple linear regression or some sophisticated hidden markov model, neural network, or anything really), you need to optimize parameters using a part of the dataset and then test the accuracy on the remaining part: this is the only way to avoid overfitting. If you ever participated (or will participate) in any kind of competition so popular these days (such as the infamous 'Netflix Challenge', now long defunct), you must have noted that this is exactly how it was done: the participants are given part of the data to train their models on, and then the winner model is determined from running all competing models on a new, yet unseen dataset that has never been released.

Here we will use a similar approach that has a flavor of resampling: we will split the dataset multiple times and assess the (average) robustness of the model against such multiple random stratifications of the data. Consider this simple code fragment:

5-fold cross validation to evaluate performance of our linear model.

Hide

```
n.xval <- 5 # number of groups to split data into
# generate and permute group labels (i.e. assign
# datapoints to groups 1..5 randomly):            Sample( ):shuffles w/o replacement
* xval.grps <- sample(1:dim(d2r.34852)[1]%%n.xval+1) = [illegible]
s2.xval <- numeric()                     modulus (remainder after division).
for ( i.xval in 1:n.xval ) { # for each group:
 # set group i aside as 'test set'
 test.df <- d2r.34852[xval.grps==i.xval,]   logical vector returns T/F. ~~then~~ then selects samples
                                                      from df that are true.
 train.df <- d2r.34852[xval.grps!=i.xval,] # the rest is "training"
 lm.xval <- lm(D2R~G,train.df) # fit the model on the training set
 test.pred <- predict(lm.xval,test.df) # predict on test set (used trained model) - if there's 25 points in the 1st set, there will be 25 predicted values
 s2.xval <- c(s2.xval,(test.pred-test.df$D2R)^2) compute sq difference b/w predicted (from test set)
}  after loop complete, final length = nrows in d2r.34852   and actual test set values (for each value - ∴ 25 values will be added).
mse.xval <- mean(s2.xval,na.rm=T)
mse.xval = out of sample error (how well the model generalizes new data).
```

does this 5 times

m of squared error

```
## [1] 202.0721
```

Hide

```
summary(lm.34852.g.at)$sigma^2   ← residual variance (in-sample error - how well the ~~training~~ model fits the training data)
```

* = sample(1:nrows %. 5 + 1) gives a repeating seq of numbers from 1 to 5, sample() randomly shuffles the sequence, so now each sample is randomly [illegible] to a group(s)

if $\sigma^2 \simeq MSE$ model generalises well
$\sigma^2 < MSE$ model fits training data better than test data
→ possible overfitting. (means that it learns noise as if it were a signal).

```
## [1] 196.7406
```

In this code fragment we split the data randomly into five groups of the same size. Then we:

- set aside one group as the "test set",
- fit a linear model on the remaining data ("training set")
- assess the model accuracy by running prediction on the test set and calculating (squared) residuals between the predictions and test set data.
- repeat for each available group used as 'test set'

As the result, we obtain a vector of squared residuals in the predicted (test!) data in each cross-validation run. We can see that the mean squared residual across all runs is ~204. Note that if we use the whole dataset and just find the best regression line, as in the first section in this document, the average squared residual is ~196 (you can simply run `anova(lm.34852.g.at)` on the model we have fitted earlier, the mean squared residual will be shown in the report among other statistics). This suggests that the model `lm.34852.g.at` might overfit the data slightly (there is another caveat here that we will discuss later)

↗ the simple lm from whole data set.

# 4 Model Bootstrapping

Let us now perform a similar assessment of model accuracy by using bootstrapping. In this approach, instead of setting aside a part of the data, we resample with replacement (just like we did last week when we were computing bootstrapped confidence intervals etc). The following code performs bootstrapping of our linear model:

Hide

```
n.boot <- 100                              # 100 bootstrap resamplings
s2.boot <- numeric()
n.obs <- dim(d2r.34852)[1]                 # number of observation (data points) rows.
for ( i.boot in 1:n.boot ) {
 train.idx <- sample(n.obs,replace=T)      # indexes for training set
 # test set=indices not used in training set:
 test.idx <- (1:n.obs)[!(1:n.obs)%in%train.idx]
 train.df <- d2r.34852[train.idx,]         # select training data
 test.df <- d2r.34852[test.idx,]           # select test data
 lm.boot <- lm(D2R~G,train.df)             # fit model on training set
 test.pred <- predict(lm.boot,test.df)     # predict on test set
 s2.boot <- c(s2.boot,(test.pred-test.df$D2R)^2)
}
mse.boot <- mean(s2.boot,na.rm=T)
mse.boot
```

sq difference b/w predicted + observed values from the test set of values.

```
## [1] 203.9496
```

The code above is very similar to the cross-validation code considered in the previous section. The only difference is that instead of splitting the data into n groups, in each pass of our bootstrapping analysis we now select training set by sampling data points with replacement (note that we sample from indices of the data points 1:n.obs), then use as the test set all the data points that were not selected into the training set.

This approach also results in greater average squared residual calculated on the predicted data than the average squared residual calculated on the fitted data themselves, so different approaches we are using are consistent.

# 5 Model Selection

As we have mentioned earlier, there is still a caveat with the approach we were following so far. If we had an externally defined model and just needed to assess its accuracy, what we did would be acceptable. However, our situation was different. Namely, while we assessed the model properly by splitting the data into the training and testing sets, the model selection was performed using the whole dataset. Indeed, you may remember how we ended up with a particular gene we were using here: we started with the days-to-remission variable $Y$, which we considered as being dependent on gene expression. We have checked the association of every gene's expression profile $X_i$ with $Y$ using all the patients (i.e. we started from a large number of potentially predictive models), and then we selected the model (gene) $Y \sim X$ with the most significant association. In light of the model accuracy assessment we are performing now, this is not fair: the very identity of the most significant association we have found is aware of all the data (and thus of the test set as well). We can thus suspect that the prediction quality estimation will be still too optimistic, because we chose a gene (model) in a way that captured to some extent an association present in a test set. Instead, we should have selected and trained the model on a training set, while being completely blind to the test set. We will realize this program in the rest of this week material.

Code ▾

# Week 7 Notes Part 3 Multiple Regression

Author: Brendan Gongol

Last update: 09 January, 2023

# 1 Overview

In this Note we finish exercise with cross-validation of a linear model: now we do it correctly, by selecting the most significant gene for each training set anew. Next we consider multiple linear regression (models with multiple independent variables), and finish the discussion by introducing a criterion useful for comparing models with different numbers of parameters.

# 2 Predictive Accuracy of a Model: Take 2

As we have discussed in the other Notes, the model cross-validation we have performed so far is not fully adequate. Namely, the gene we selected to fit our model against, was selected by looking into the whole dataset (including what we later try setting aside as a 'test set'). Let us now perform cross-validation in a correct way. We will use the same framework as developed in the previous Note, but this time we are going to use `lmFit()` for convenience, since we need to re-assess all fits run on a training set only. Let's see the code:

Hide

```
library(limma);library(ALL); data(ALL)
ALL.pdat <- pData(ALL)
date.cr.chr <- as.character(ALL.pdat$date.cr)
diag.chr <- as.character(ALL.pdat$diagnosis)
date.cr.t <- strptime(date.cr.chr,"%m/%d/%Y")
diag.t <- strptime(diag.chr,"%m/%d/%Y")
days2remiss <- as.numeric(date.cr.t - diag.t)
x.d2r <- as.numeric(days2remiss)
exprs.34852 <- exprs(ALL)["34852_g_at",]
d2r.34852 <- data.frame(G=exprs.34852,D2R=x.d2r)
################################################
days2remiss <- as.numeric(date.cr.t - diag.t) # as in Note 1
ALL.exprs <- exprs(ALL)[,!is.na(days2remiss)]
days2remiss <- days2remiss[!is.na(days2remiss)]
design.matrix <- cbind(rep(1,length(days2remiss)),days2remiss)
n.xval <- 5
s2.xval <- numeric()
xval.grps <- sample((1:dim(ALL.exprs)[2])%%n.xval+1)
for ( i.xval in 1:n.xval ) {
 ALL.exprs.train <- ALL.exprs[,xval.grps!=i.xval]
 design.matrix.train <- design.matrix[xval.grps!=i.xval,]
 d2r.fit.train <- lmFit(ALL.exprs.train,design.matrix.train)
 best.gene <- rownames( topTable( eBayes(d2r.fit.train),
 "days2remiss") )[1]
 cat(best.gene," ",fill=i.xval==n.xval)
 tmp.df <- data.frame(G=ALL.exprs[best.gene,],D2R=days2remiss)
 lm.xval <- lm(D2R~G,tmp.df[xval.grps!=i.xval,])
 test.pred <- predict(lm.xval,tmp.df[xval.grps==i.xval,])
 s2.xval <- c(s2.xval,(test.pred-
 tmp.df[xval.grps==i.xval,"D2R"])^2)
}
```

~~remove~~ select columns of expression df w/o NA before removing NA values from days2remiss.

creates a vector of 1s the same length of days2remiss

binds to vectors into a matrix. column 1 = 1s col. 2 = values of days2remiss.

5 groups.

store $\sigma^2$ of each x value

4 groups as training.

creates design matrix of only the training sample.

train simple linear model using best gene

ranks genes by association, then top table sorts by significance ↳ select top.

formatting - see copilot notes.

predicted values on test set for best gene

actual test observed values.

lm fit (expression_matrix, design-matrix)

```
## 34852_g_at  1213_at  1074_at  651_at  33901_at
```

different 4 best genes each fold.

Hide

```
mean(s2.xval)
```

```
## [1] 272.0612
```

Most of the code reproduces what we did before, so it should be familiar. We do cross-validation, so we again randomly split our dataset into 5 equal parts (n.xval). In a loop, we set aside one part as a test set and use the rest as the training set. Our new approach is that now we want to be consistent and not use the test set at all until it is time to assess the prediction accuracy of the model. Hence, every time we fit every gene against days-to-remission (using the training set only!) and find the gene with most significant association.

In order to find such most significant gene we could use one of the methods from previous weeks (loop over all genes or use `apply()` function in order to run `lm()` on every gene), but instead we are employing the `lmFit()` function in this example. Note that this function is implemented with microarray analysis in mind, so in that particular respect it is less flexible than `lm()` : the arguments are expression levels of multiple genes, and the design matrix. As the result, we cannot, strictly speaking, fit a formula days-to-remission ~ gene expression. Instead, `lmFit()` forces us to fit expression levels of each gene (rows of the input matrix) against an independent variable. This does not pose a problem, since association significant in one way ( $Y \sim X$) is also significant the other way around ($Y \sim X$). So we are going to fit expression of each gene against the vector of days-to-remission. Since instead of a formula, `lmFit()` takes design matrix, we generate one in the code above. Make sure you understand how the design matrix is built: first column is 1, and the second column contains values of the days-to-remission variable (see Eq. (2) in Part 2 of the Notes!). Using this design matrix we can make `lmFit()` fit the model gene_expression = A*days_to_remission+B with a continuous independent variable, just like `lm()` would do. b/c

Design matrix must contain first column of 1s when using refrence coding (as opposed to one-hot coding).

When the training set and corresponding design matrix are selected, we run `lmFit()`, adjust significance levels using `eBayes()`, and pick the most significant gene using `topTable()` (you can examine output of `topTable()` in order to see the full table it returns). After the best gene is selected, we re-fit days-to-remission against just that gene's expression using `lm()` and the same formula as in earlier Notes (days-to-remission ~ gene expression) - we want our predictions to be directly comparable to what we did before. The rest is the same as in our earlier exercise: we predict days-to-remission in a test set and compute and save the squared residuals.

There are two important observations that one can make when running the code fragment shown above:

- The mean squared residual in the test set is now much worse (~267) than the overly optimistic estimate we have obtained earlier (~204). The only reason is that in the previous runs, the very identity of the "most significantly associated" gene was determined in part by the data from the test set. As you can see, that mistake alone can result in our case in ~25% overestimation of the model's predictive accuracy.

-The gene with most significant association with days-to-remission can be different from run to run (depending on what we randomly selected into the training set!): our code prints the ID of the selected gene, and you can see that we found three different genes, two of them appeared twice (i.e. for two different training sets).

An interesting question without a definitive answer is what we are supposed to do with different "most significant" genes obtained in different training sets. Depending on the situation and the goal, few different interpretations and strategies are possible:

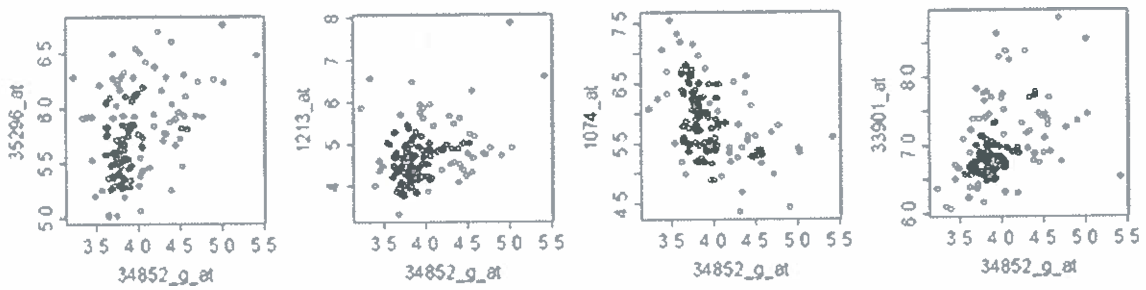
- The very fact that we obtain different genes in different runs suggests that there might be a few genes significantly (and comparably) associated with days-to-remission; selection of different subsets of data as a "training set" pushes one or the other(s) on top due to random sub-sampling of the data points.
- Alternatively, it is also possible that at least some of the random selections of the training set are "unlucky" in a sense that we pick outliers that distort the data (and may result in the wrong "most significant" gene, which will give relatively poor predictions on the test set).
- If the predictor is what we are really after, we can perform multiple resamplings as in the code fragment above, and then select gene(s) that show up as most significant in the largest number of randomly selected training sets. Note that this procedure (a) is ad-hoc; (b) strictly speaking, does not allow for fair estimation of prediction accuracy (since we do not have a pure, truly unseen test set anymore); (c) at the same time, is likely to be more robust and provide decent predictions (it's the true accuracy of those predictions that we cannot assess anymore as mentioned in (b)). Such "consensus" (and related) approaches are sometimes employed in practice, even though it can be hard to build rigorous theoretical foundation for them.

- If different genes showing up in different resampling runs are indeed all significant, then we may want to examine a fit that includes all/most of them at once (we will see how to do this in next section!); maybe together they provide even better fit.
- If we are trying to answer a general question about a power of our dataset, rather than building a specific predictor, we might be not even concerned about the identity of a specific gene: we can interpret the results above as a statement about how well (or how poorly) a model that uses a gene as an independent (explanatory) variable will predict days-to-remission. In other words, it is a statement about the degree of association and possibility (accuracy) of prediction.
- We may use a hybrid approach: set aside small set of data as an actual test set. Then on the training set alone perform additional resamplings as shown above and use some kind of consensus approach in order to select the model (gene) that shows up most often. Then we still can evaluate the prediction accuracy of our consensus fit on the initially reserved test set.

# 3 Multiple Linear Regression

In linear models we have been studying and practicing so far, we were postulating and analyzing the dependence of one variable on another (e.g. days-to-remission on gene expression). When multiple explanatory (independent) variables were available (e.g. 12,000 genes in the ALL dataset), we so far were selecting the one most significantly associated with the dependent variable. This is a legitimate approach, but what if the days to remission value in a given patient is indeed determined (either in a sense of being "better predicted", or even causally) by expression levels of more than one gene?

When we ran correct cross-validation procedure in the previous section (where we determined the most significant gene in each resampling iteration anew), we observed that different genes were selected in different runs: 34852_g_at, 35296_at. Few more top genes we could observe with more resamplings (or simply by looking at the top list of the most significant genes, not just one), are $1213_at$, $1074_at$, $33901_at, \ldots$ If all those genes are somehow associated with days-to-remission, is it possible that together they would form a better model? The answer may vary. Imagine two genes $g_1$, $g_2$, that are perfectly correlated, e.g. $g_1 = ag_2$. It is obvious that in this case either $g_1$ or $g_2$ will result in an equally good (or bad) model for days-to-remission, and adding the second gene to the model as another explanatory variable is not going to help at all: since expression level of this gene is completely determined by the expression of the first one, we are not adding any new information to the model! This suggests that we may want to look for genes that are strongly correlated to the dependent variable (days-to-remission in our case), i.e. provide information, but are not correlated too strongly among each other (i.e. the information each gene provides is sufficiently unique and not captured already in other genes). Such problem of selecting an "optimal" set of features (genes in our example) that gives the best prediction for the dependent variable is known as feature selection, and there are many methods and lots of research literature dedicated to this subject. While not plungingtoo deep into the feature selection, consider the scatterplots of a few genes listed above (you should be able to reproduce these plots very easily):



We can see that while some pairwise correlations between our top genes apparently exist, they are not too strong, and there is hope that by adding information provided by these genes together we may end up with a better model for days-to-remission.

In order to fit a linear model in R with the dependent variable Y and multiple independent variables $X_1, X_2, \ldots, X_N$, we can simply specify the formula as $Y \sim X_1 + X_2 + \ldots + X_N$. The regression hyperplane will be sought as $y = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b$. The following code provides an example:

Hide

```
exprs.35296 <- exprs(ALL)["35296_at",];
exprs.34852 <- exprs(ALL)["34852_g_at",]
exprs.1213 <- exprs(ALL)["1213_at",]
g3.df <- data.frame(D2R=x.d2r,G1=exprs.34852,  } df of D2R + 3 genes of intrest
 G2=exprs.35296,G3=exprs.1213)
g3.df <- g3.df[!is.na(g3.df$D2R),]  — remove rows where D2R is NA.
lm.g1 <- lm(D2R~G1,g3.df)
lm.g12 <- lm(D2R~G1+G2,g3.df)
summary(lm.g12)
```

```
##
## Call:
## lm(formula = D2R ~ G1 + G2, data = g3.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.737  -7.251  -0.147   7.721  44.256
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  207.133     21.630   9.576 1.61e-15 ***
## G1           -15.369      4.181  -3.676 0.000396 ***
## G2           -13.928      3.986  -3.494 0.000730 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.26 on 93 degrees of freedom
## Multiple R-squared:  0.3326, Adjusted R-squared:  0.3182
## F-statistic: 23.17 on 2 and 93 DF,  p-value: 6.827e-09
```

Hide

```
anova(lm.g12)
```

```
## Analysis of Variance Table
##
## Response: D2R
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## G1         1  6000.1  6000.1  34.135 7.526e-08 ***
## G2         1  2146.3  2146.3  12.210 0.0007298 ***
## Residuals 93 16347.3   175.8
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Hide

```
anova(lm.g1,lm.g12)
```

```
## Analysis of Variance Table
##
## Model 1: D2R ~ G1
## Model 2: D2R ~ G1 + G2
##   Res.Df   RSS Df Sum of Sq     F    Pr(>F)
## 1     94 18494 — 2146.3
## 2     93 16347  1    2146.3 12.21 0.0007298 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the code fragment above we start with packing days-to-remission and expression levels of three of our top genes into a dataframe (for convenience, as usual; this is not required); then we fit one model with days-to-remission (D2R) explained by expression of a single gene (this is the model we dealt with extensively in the past), and another model with $D2R$ explained by combination of gene expression levels of two genes ( $D2R \sim G1 + G2$). The summary of the second model shows that both slopes (for $G1$ and for $G2$) are quite significant. According to ANOVA, the coefficients in the two-gene model are significant as well. Remember that, as we discussed earlier, ANOVA for linear model fits is set up in such a way that it measures additional contribution of each model term: in the example above, the $G1$ term alone “explains away” sum of squares (variance) of 6000.1; adding next term $(G2)$ to the model helps “explaining” (i.e. attributing to $G2$) additional 2146.3 towards the total variance of the dependent variable.

In the last call to ANOVA in the code fragment above we compare the two models. Note that the models we are comparing are nested (i.e. one is the part of the other: whichever model fit we can obtain for $D2R \sim G1$, we can always get exactly the same fit with $D2R \sim G1 + G2$ by setting the slope of $G2$ to 0 and slope of $G1$ to the value in the simpler model; hence the second model can always reproduce the first one and can also provide more when both slopes are non-zero). If you look carefully at the output of the last anova command, you will see that the first line summarizes the first model $D2R \sim G1$ (RSS is residual sum of squares); the second line provides the information about the second model, $D2R \sim G1 + G2$, and this is essentially the information about the additional term (i.e. the “difference” between the models). The RSS of the second model decreases compared to the first one (exactly by the sum of squares explained away by the addition of the $G2$ term); the remaining data in the second line are the same as the parameters reported for the $G2$ coefficient in the previous call to ANOVA, so it's just a different presentation of the same data, more convenient for comparing the two models.

Let us continue the example and fit a two-gene model with different genes ($G1$ and $G3$ instead of $G1$ and $G2$); the code is otherwise identical:

Hide

```
lm.g13 <- lm(D2R~G1+G3,g3.df)
summary(lm.g13)$coef
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 174.643582  15.635437 11.169728 7.117849e-19
## G1          -17.361332   3.830502 -4.532391 1.732419e-05
## G3           -8.402194   2.060142 -4.078454 9.559015e-05
```

Hide

```
anova(lm.g13)
```

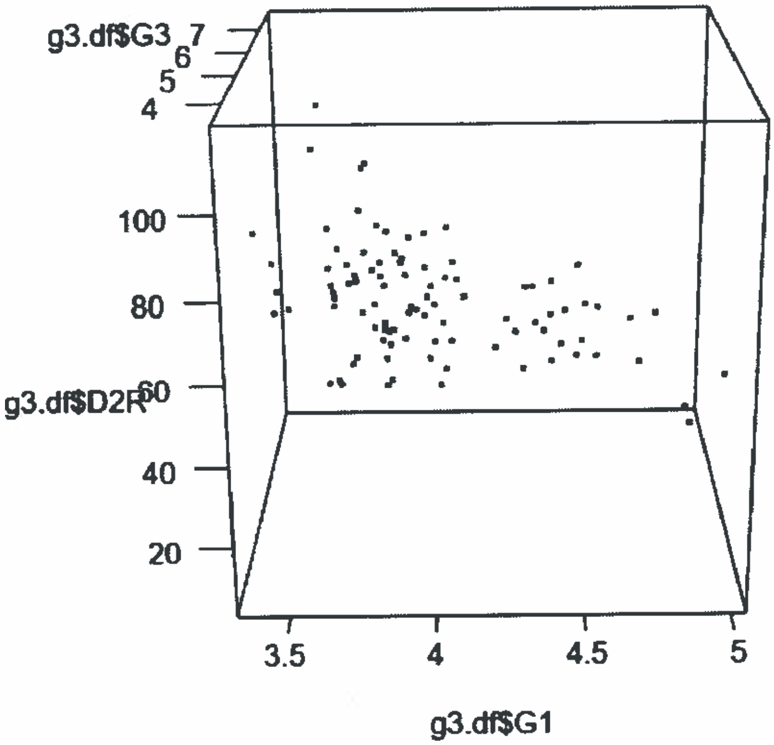
```
## Analysis of Variance Table
##
## Response: D2R
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## G1         1  6000.1  6000.1  35.570 4.404e-08 ***
## G3         1  2805.9  2805.9  16.634 9.559e-05 ***
## Residuals 93 15687.7   168.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that this model seems to provide even better fit: the slopes are more significant, and the second term ($G3$) explains away more variance (2805.2) than $G2$ did (2146.3) in the $D2R \sim G1 + G2$ model (the first term $G1$ explains away same variance in both models). The sum of squared residuals is thus smaller for the $D2R \sim G1 + G3$ model we just tried.

If you download package statterplot3d, you can use the following visualization of our two-gene fit:

Hide

```
# install.packages("scatterplot3d")
# install.packages("rgl")
library(rgl)
library(scatterplot3d)
# for ( a in 1:1800 ) {
#   x3d <- scatterplot3d(g3.df$G1, g3.df$G3, g3.df$D2R, angle=a%%360)
#   x3d$plane3d(coef(lm.g13))
#   Sys.sleep(0.01)
# }
plot3d(g3.df$G1,g3.df$G3,g3.df$D2R); rglwidget(width = 520, height = 520)
```



We can always look at the fitted coefficients of the model and the design matrix:

Hide

```
summary(lm.g1)$coef
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 152.59905  15.844606  9.630978 1.119051e-15
## G1          -21.87222   3.960595 -5.522459 2.958899e-07
```

Hide

```
summary(lm.g13)$coef
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 174.643582  15.635437 11.169728 7.117849e-19
## G1          -17.361332   3.830502 -4.532391 1.732419e-05
## G3           -8.402194   2.060142 -4.078454 9.559015e-05
```

Hide

```
model.matrix(lm.g1)[1:3,]
```

```
##       (Intercept)       G1
## 01005           1 3.628618
## 01010           1 4.041296
## 03002           1 3.827026
```

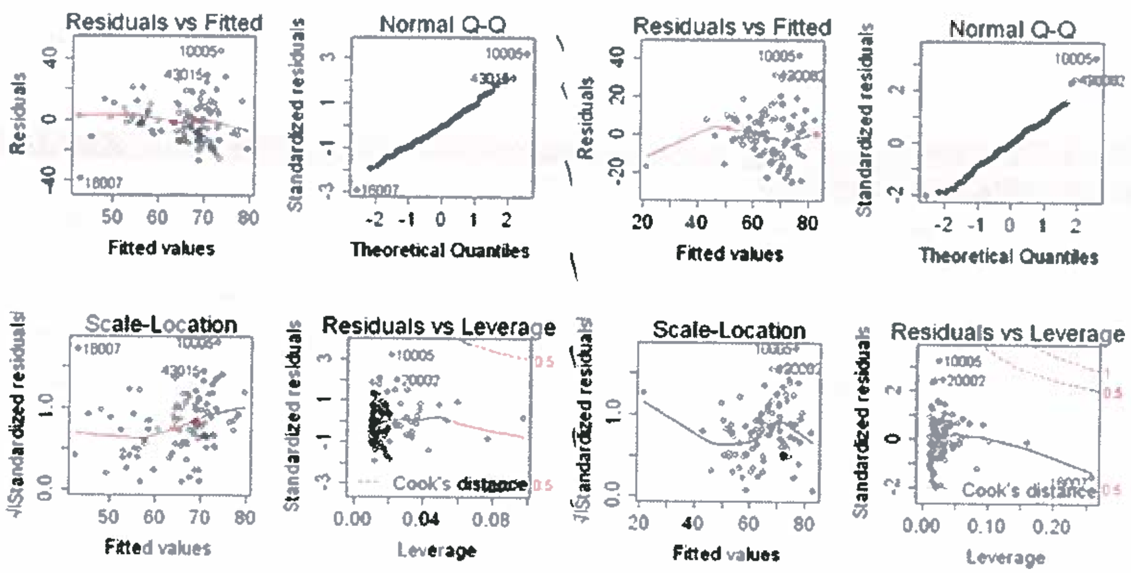
Hide

```
model.matrix(lm.g13)[1:3,]
```

```
##       (Intercept)       G1       G3
## 01005           1 3.628618 4.478010
## 01010           1 4.041296 4.122028
## 03002           1 3.827026 4.484498
```

As we can see, the design matrix for $D2R \sim G1 + G3$ model contains one column for values of $G1$, and one column for values of $G3$. As you remember from the previous note, the predicted (fitted) value is simply [design matrix] * [ fitted coefficients ] , where fitted coefficients is a column and the matrix multiplication is assumed (see Note 2, Eqs.(2)-(4)).

For linear models with multiple independent variables we can still use plot() in order to draw the same diagnostic plots as the one we extensively studies earlier in the course. Executing `plot(lm.g1)` and `plot(lm.g13)` will generate the following figures (left is $D2R \sim G1$, and the right is $D2R \sim G1 + G3$):



Note that while $D2R \sim G1 + G3$ provides a closer fit (smaller RSS, more variance in $D2R$ explained), this fit is a little worse with respect to the assumptions and statistics examined in the diagnostics.

Note that despite the fact that our two-gene model provides a closer fit (smaller residuals), the model is also more complex (and as we discussed, model D2R ~ G1 is nested into it). If we add new varaiables (model paramers), we always expect to have a better fit – even if what we are fitting is actually noise without any real dependence. Indeed, with nested models we are guaranteed that with D2R ~ G1+G3 we are going to perform at least as well as D2R ~ G1, and since the parameter space for the former model is larger, we can always get at least a slightly better fit. But that should hold true if we add another (gene), and yet another etc. The fit will keep improving but it is likely we are going to start fitting the noise, i.e. the models might overfit the limited data at hand (imagine trying to fit only 93 available D2R values using 1000 genes as independent variables – clearly we are going to fit almost any set of D2R values this way, nearly perfectly!). The significance of the coefficients for the additional genes is definitely helpful, but it does not tell the whole story, especially in the case of overfitting: it is quite possible that with addition of extra features (independent variables), we just start (truly) fitting unique realization of noise/outliers in the limited data we have available. So the fit might be still at least somewhat significant, just meaningless.

The question of comparing two models is a difficult one, and strictly speaking it does not have a single rigorous theoretical answer. There is a number of ad-hoc and (semi-)formal approaches, however, that allow for meaningful comparisons. One such approach is Akaike Information Criterion. In general form it is defined as

$$AIC = 2k - ln(L)$$

Where $k$ is the number of parameters in the model, and $L$ is the likelihood of data given the model (we will reproduce it here one more time for a linear model):

$$L = \prod_{i=1}^{n} (\frac{1}{2\pi\sigma_i^2})^{1/2} exp(-\sum_{i=1}^{n} \frac{(y_i - f(x))^2}{2\sigma_i^2})$$

When the standard deviations of the errors σi are identical at each data point, the expression can be also rewritten (up to unimportant additive constant) as

$$AIC = nlog(\frac{RSS}{n}) + 2k$$

Additive constant does not matter since what we need is the difference between the values calculated for different models in order to compare those models to each other.

The expression for AIC decreases when the likelihood L increases (i.e. when RSS decreases), so the models with better fit should have lesser AIC. However, AIC also penalizes models with high numbers of parameters, k (AIC value increases). As a result, the qualitative meaning of AIC (and other similar criteria) is that it ensures we get a sufficiently large improvement in the fit for each additional parameter (or independent variable) added. If we add parameters/independent variables but the resulting gain in the quality of the fit is “too small” (that's what the mathematical formalism behind those criteria does – defines “too small”) for the price (extra parameters) we paid, the AIC will increase and that will indicate that adding those extra parameters is not justified. Consider an example:

Hide

```
AIC(lm(D2R~G1,g3.df),
 lm(D2R~G3,g3.df),
 lm(D2R~G1+G3,g3.df),
 lm(D2R~G1*G3,g3.df))
```

```
##                          df      AIC
## lm(D2R ~ G1, g3.df)       3 783.4761
## lm(D2R ~ G3, g3.df)       3 786.8392
## lm(D2R ~ G1 + G3, g3.df)  4 769.6797
## lm(D2R ~ G1 * G3, g3.df)  5 770.5488
```

In the code above, we computed AIC for four fitted linear models at once (the last one, with $G1 * G3$ in the formula fits the model with linear and interaction terms: this is a shortcut for $y = a_1x_1 + a_2x_2 + cx_1x_2$ ). The results show that our two-gene model $G1 + G3$ has slightly better AIC than any of the single-gene model(s), so we may want to choose the two-gene $G1 + G3$ model indeed. At the same time, the model with an interaction term (i.e. a $G1 * G3$), has an additional parameter (denoted c in the formula above) compared to G1+G3, and likely provides even better fit, but when weighed against the price of adding that extra parameter this latter fit results in slightly worse value of AIC compared to just $G1 + G3$, so we might want to refrain from using $G1 * G3$ in this case.

Another often used and closely related approach uses Bayesian Information Criterion (BIC). It is derived under slightly different assumptions, and as a result is defined by a different expression:

$$BIC = klog(n) - 2logL$$

where k is again the number of model parameters and $L$ is the data likelihood. Note that BIC gradually increases penalty for additional model parameters when the number of available data points (n) increases. When RSS is known, BIC can be also expressed as $BIC = n \times \log\left(\frac{RSS}{n}\right) + k \times \log(n)$

Derivation of AIC, BIC or other criteria is beyond the scope of this course, but they generate similar results: consider applying BIC to the same models as above (the criteria are indeed so close that we can compute BICV using the `k` argument of the `AIC()` function):

Hide

```
AIC(lm(D2R~G1, g3.df),
    lm(D2R~G3, g3.df),
    lm(D2R~G1+G3, g3.df),
    lm(D2R~G1*G3, g3.df),
    k=log(dim(g3.df)[1]))
```

```
##                            df      AIC
## lm(D2R ~ G1, g3.df)         3 791.1692
## lm(D2R ~ G3, g3.df)         3 794.5323
## lm(D2R ~ G1 + G3, g3.df)    4 779.9371
## lm(D2R ~ G1 * G3, g3.df)    5 783.3705
```

Again, we see that the $G1 + G3$ model seems to provide the best (and most informative) fit out of all the models compared, while the model with the cross-term included ($G1 * G3$) does not provide sufficient improvement in the fit (i.e data likelihood) in order to justify introduction of the additional parameter into the model (so the overall value of BIC increases to -783, last line in the output above).

Note that BIC or AIC are specifically designed to compare models with different numbers of parameters (and such comparison is not a trivial task since the model with the larger number of parameters always gives a “better” fit); when the numbers of parameters are the same in the models we are trying to compare, AIC or BIC are simply equivalent to the maximum likelihood (terms with `k` are the same and thus not important in this case).