DNA Hydroxymethylation in Hepatocellular Carcinoma

Francois Collin

2020

Contents

Preamble

This vignette offers some exploratory data analyses of DNA Hydroxymethylation data available from NCBI GEO Series GSE112679. These data can be conveniently accessed through an R data package. See GSE112679 R Data Package page.

License

Chapter 1

Introduction

The goal of detecting cancer at the earliest stage of development with a non-invasive procedure has busied many groups with the task of perfecting techniques to support what has become commonly known as a liquid biopsy - the analysis of biomarkers circulating in fluids such as blood, saliva or urine. Epigenetic biomarkers present themselves as good candidates for this application (Gai and Sun (2019) [1]). In particular, given their prevalence in the human genome, close correlation with gene expression and high chemical stability, DNA modifications such as 5-methylcytosine (5mC) and 5-hydroxymethylcytosine (5hmC) are DNA epigenetic marks that provide much promise as cancer diagnosis biomarkers that could be profitably analyzed in liquid biopsies [2–5].

Li et al. (2017) [3] used a sensitive and selective chemical labeling technology to extract genome-wide 5hmC profiles from circulating cell-free DNA (cfDNA) as well as from genomic DNA (gDNA) collected from a cohort of 260 patients recently diagnosed with colorectal, gastric, pancreatic, liver or thyroid cancer and normal tissues from 90 healthy individuals They found 5hmC-based biomarkers of circulating cfDNA to be highly predictive of some cancer types. Similar small sample size findings were reported in Song et al. (2017) [4].

Focusing on hepatocellular carcinoma, Cai et al. (2019) [2] assembled a sizable dataset to demonstrate the feasibility of using features derived from 5-hydroxymethylcytosines marks in circulating cell-free DNA as a non-invasive approach for the early detection of hepatocellular carcinoma. The data that are the basis of that report are available on the NCBI GEO web site (Series GSE112679). The data have also been bundled in a R data package which can be installed from github:

```
if (!requireNamespace("devtools", quietly = TRUE))
    install.packages("devtools")
devtools::install_github("12379Monty/GSE112679")
```

An important question in the early development of classifiers of the sorts that are the basis of any liquid biopsy diagnostic tool is how many samples should be collected to make properly informed decisions. In this report we will explore the GSE112679 data to shed some light on the relationship between sample size and model performance in the context classifying samples based on 5hmC data.

The layout of the paper is the following:

- In Section **??** we provide background on fitting and analyizing predictive models in the `n << p` context and give some detail about the primary tool used in this report.
- In Section **??** we preprocess the 5hmC data that we will use for the classification analysis and perform some light QC analyses.
- In Section **??** we explore sparse models that discriminate between early stage HCC and control samples.
- In Section **??** we examine the results of fitting a suite of models to investigate the effect of sample size on model performance.
- Concluding remarks are in Section **??**.

Chapter 2

Classification Analysis in the `n << p` Context

The main challenge in calibrating predictive models to genomic data is that there are many more features than there are example cases to fit to; the now classic $n << p$ problem. In this scenario, fitting methods tend to over fit. The problem can be addressed by selecting variables, regularizing the fit or both.

In this report, we use genomic data to illustrate analyses of regularized regression models in the `n << p` context. Much of the analysis focuses on lasso models fitted and analyzed using tools from the `glmnet` R package.

Before providing some background on that glmnet package and the models that it supports, we briefly describe another R package which is an essential tool for anyone who is doing predicitive analytics in R - `caret` R pacakge.

2.1 caret for model evaluation

The `caret` Package provides a rich set of functions that streamline the process for fitting and evaluating a large number of predictive models in parallel. The package contains tools for:

- data splitting
- pre-processing
- feature selection

- model tuning using re-sampling
- variable importance estimation

The tools facilitate the process of automating randomly splitting data sets into training, testing and evaluating so that predictive models can be evaluated on a comparable and exhaustive basis. Especially useful is the functionality that is provided to repeatedly randomly stratify samples into train and test set so that any sample selection bias is removed.

What makes the `caret` package extremely useful is that it provides a common interface to an exhaustive collection of fitting procedures. Without this common interface one has to learn the specific syntax that used in each fitting procedure to be included in a comparative analysis, which can be quite burdensome.

Some of the models which can be evaluated with caret include: (only some of these can be used with multinomial responses)

- FDA - Flexible Discriminant Analysis
- stepLDA - Linear Discriminant Analysis with Stepwise Feature Selection
- stepQDA - Quadratic Discriminant Analysis with Stepwise Feature Selection
- knn - k nearest neighbors
- pam - Nearest shrunken centroids
- rf - Random forests
- svmRadial - Support vector machines (RBF kernel)
- gbm - Boosted trees
- xgbLinear - eXtreme Gradient Boosting
- xgbTree - eXtreme Gradient Boosting
- neuralnet - neural network

Many more models can be implemented and evaluated with `caret`, including some `deep learning` methods, `Simulated Annealing Feature Selection` and `Genetic Algorithms`. Many other methods found here are also worth investigating.

We mention `caret` here because it is an extremely useful tool for anyone interested in exploring and comparing the performance of many predictive models which could be applied to a given predictive analysis context. We have gone through this exercise with a number of genomic scale, RNA-Seq data sets in the past and have found that in this context regularized regression models perform as well as any. For this report, we will only consider regularized classification models and will focus on the particular set of tools for fitting and analyzing these models provided by the `glmnet` R package [6].

2.2 The `glmnet` R package

Several factors favor using the `glmnet` R package to analyze classification models fitted to genomic scale data:

- the glmnet package is a well supported package providing extensive functionality for regularized regression and classification models.
- the hyper-parameters of the elastic net enable us to explore the relationship between model size, or sparsity, and predictive accuracy. ie. we can investigate the "bet on sparsity" principle: *Use a procedure that does well in sparse problems, since no procedure does well in dense problems.*
- in our experience building classifiers from genomic scale data, regularized classification models using the elastic net penalty do as well as any other, and are more economical in terms of computing time, especially in comparison to the more exotic boosting algorithms.
- the `lasso` has been shown to be near optimal for the $n << p$ problem over a wide range of signal-to-noise regiments (Hastie et al. (2017) [7]).

Much of the following comes from the Glmnet Vignette.

Glmnet is a package that fits a generalized linear model via penalized maximum likelihood. The regularization path is computed for the lasso or elastic net penalty at a grid of values for the regularization parameter lambda ([6,8–10]).

`glmnet` solves the following problem:

$$\min_{\beta_0,\beta} \frac{1}{N} \sum_{i=1}^{N} w_i l(y_i, \beta_0 + \beta^T x_i) + \lambda \left[(1-\alpha)||\beta||_2^2/2 + \alpha||\beta||_1 \right],$$

over a grid of values of λ. Here $l(y, \eta)$ is the negative log-likelihood contribution for observation i; e.g. for the Gaussian case it is $\frac{1}{2}(y-\eta)^2$.

alpha hyper-parameter

The elastic-net penalty is controlled by α, and bridges the gap between lasso (α=1, the default) and ridge (α=0). The tuning parameter λ controls the overall strength of the penalty.

It is known that the ridge penalty shrinks the coefficients of correlated predictors towards each other while the lasso tends to pick one of them and discard the others. The elastic-net penalty mixes these two; if predictors are correlated in groups, an α=0.5 tends to select the groups in or out together. This is a higher level parameter, and users might pick a value upfront, else experiment with a few different values. One use of α is for numerical stability; for example, the *elastic net with* $\alpha = 1 - \epsilon$ *for some small* ϵ*>0 performs much like the lasso, but removes any degeneracies and wild behavior caused by extreme correlations.*

2.3 Signal-to-noise Ratio

A key characteristic of classification problems is the prevailing signal-to-noise ratio (SNR) of the problem at hand.

To define SNR, let $(x_0, y_0) \in \mathbb{R}^p \times \mathbb{R}$ be a pair of predcitor and response variables and define $f(x_0) = \mathbb{E}(y_0|x_0)$ and $\epsilon = y_0 - f(x_0)$ so that

$$y_0 = f(x_0) + \epsilon_0.$$

The signal-to-noise ratio in this model is defined as

$$SNR = \frac{var(f(x_0))}{var(\epsilon_0)}.$$

It is useful to relate the SNR of a model to the proportion of variance explained (PVE). For a given prediction function g — eg. one trained on n samples $(x_i, y_i)i = 1, \ldots, n$ that are i.i.d. to (x_0, y_0) — its associated proportion of variance explained is defined as

$$PVE(g) = 1 - \frac{\mathbb{E}(y_0 - g(x_0))^2}{Var(y_0)}.$$

This is maximized when we take g to be the mean function f itself, in which case

$$PVE(f) = 1 - \frac{Var(\epsilon_0)}{Var(y_0)} = \frac{SNR}{1 + SNR}.$$

Or equivalently,

$$SNR = \frac{PVE}{1 - PVE}.$$

Hastie, Tibshirani, and Tibshirani (2017) [7], point out that PVE is typically in the 0.2 range, and much lower in financial data. It is also much lower in 5hmC data, as we will see in the next section.

Note that the SNR is a different characterization of noise level than the coefficient of variation:

$$c_v = \frac{\sigma}{\mu} = \frac{Var(y)}{\mathbb{E}(y)}$$

Note that for small SNR, SNR $\approx$ PVE.

See Xiang et al. (2020) [11], Lozoya et al. (2018) [12], Simonson et al. (2018) [13] and Rapaport et al. (2013) [14] for SNR in RNA-Seq

2.4 Lasso vs Best Subset

Best subset selection finds the subset of k predictors that produces the best fit in terms of squared error, solving the nonconvex problem:

$$\min_{\beta \in \mathcal{R}^p} ||Y - X\beta||_2^2 \; subject\, to \; ||\beta||_0 \leq k \tag{2.1}$$

The lasso solves a covex relaxation of the above where we replace the l_0 norm by the l_1 norm, namely

$$\min_{\beta \in \mathcal{R}^p} ||Y - X\beta||_2^2 \; subject\, to \; ||\beta||_1 \leq t$$

(2.2)

where $||\beta||_1 = \sum_{i=1}^{p} |\beta_i|$, and $t \geq 0$ is a tuning parameter.

Bertsimas et al. (2016) [15] presented a mixed integer optimization (MIO) formulation for the best subset selection problem. Using these MIO solvers, one can solve problems with p in the hundreds and even thousands. Bertsimas et

al. showed evidence that best subset selection generally gives superior prediction accuracy compared to forward stepwise selection and the lasso, over a variety of problem setups.

In Hastie et al. (2017) [7], the authors countered by arguing that neither best subset selection nor the lasso uniformly dominate the other, with best subset selection generally performing better in high signal-to-noise (SNR) ratio regimes, and **the lasso better in low SNR regimes**. Best subset selection and forward stepwise perform quite similarly over a range of SNR contexts, but the relaxed lasso is the overall best option, performing just about as well as the lasso in low SNR scenarios, and as well as best subset selection in high SNR scenarios. Hastie et al. conclude that a blended mix of lasso and relaxed lasso estimator, the *shrunken relaxed lasso fit*, is able to use its auxiliary shrinkage parameter (γ) to get the "best of both worlds": it accepts the heavy shrinkage from the lasso when such shrinkage is helpful, and reverses it when it is not.

Suppose the **glmnet** fitted linear predictor at λ is $\hat{\eta}_\lambda(x)$ and the relaxed version is $\tilde{\eta}_\lambda(x)$, then the shrunken relaxed lasso fit is

$$_{\lambda,\gamma}(x) = (1-\gamma)\tilde{\eta}_\lambda(x) + \gamma\hat{\eta}_\lambda(x) \tag{2.3}$$

$\gamma \in [0, 1]$ is an additional tuning parameter which can be selected by cross validation.

The de-biasing will potentially improve prediction performance, and cross-validation will typically select a model with a smaller number of variables. This procedure is very competitive with forward-stepwise and best-subset regression, and has a considerable speed advantage when the number of variables is large. This is especially true for best-subset, but even so for forward stepwise. The latter has to plod through the variables one-at-a-time, while glmnet will just plunge in and find a good active set.

Further details may be found in Friedman, Hastie, and Tibshirani (2010), Tibshirani et al. (2012), Simon et al. (2011), Simon, Friedman, and Hastie (2013) and Hastie, Tibshirani, and Tibshirani (2017) ([6–10]).

Chapter 3

5hmc Data Preprocessing

Reader note:

- The most interesting part of this section is Subsection **??** which tells us where on the SNR spectrum the 5hmC data lies. In view of the work by Hastie et al. (2017) [7] summarized in Subsection **??**, we would expect to lasso model, and the relaxed version, to do very well with this classifiaction problem.

3.1 Load the data

The data that are available from NCBI GEO Series GSE112679 can be conveniently accessed through an R data package. Attaching the GSE112679 package makes the count data tables available as well as a gene annotation table and a sample description table. See GSE112679 R Data Package page. In the Cai et al. [2] paper, samples were separated into `Train` and `Val-1` subsets for model fitting and analysis. `Val-2` was used as an external validation set.

```
if (!("GSE112679" %in% rownames(installed.packages()))) {
  if (!requireNamespace("devtools", quietly = TRUE)) {
    install.packages("devtools")
  }
  devtools::install_github("12379Monty/GSE112679")
}
library(GSE112679)
sampDesc$DxStage <- with(sampDesc, ifelse(outcome=='HCC',
   paste0(outcome,':',stage), outcome))
```

Table 3.1: GSE112679 Samples by Dx Group and Subset

	Train	Val-1	Val-2
Benign	253	132	3
CHB	190	96	0
Cirrhosis	73	33	0
HCC:Early	335	220	24
HCC:Late	0	442	13
HCC:NA	0	147	23
Healthy	269	124	177

```
with(
  sampDesc %>% dplyr::filter(sampType == "blood"),
  knitr::kable(table(DxStage, trainValGroup, exclude = NULL),
   caption="GSE112679 Samples by Dx Group and Subset") %>%
  kableExtra::kable_styling(full_width = F)
)
```

For this analysis, we will consider early stage cancer samples and healthy or benign samples from the `Train` or `Val-1` subsets. The appropriate outcome variable will be renamed or aliased `group`

```
# Use suffix 'A' for Analysis samples
sampDescA <-
  sampDesc %>%
  dplyr::filter(sampType == "blood" &
    (trainValGroup %in% c("Train", "Val-1")) &
    ((outcome2 == "BenignHealthy") |
      (outcome2 == "HCC" & stage == "Early"))) %>%
  dplyr::rename(group = outcome2) %>%
  dplyr::arrange(group, sampID)
# Recode group
sampDescA$group <- with(
  sampDescA,
  ifelse(group == "BenignHealthy", "Control", group)
)
# set groupCol for later
groupCol <- c("#F3C300", "#875692")
names(groupCol) <- unique(sampDescA$group)

with(sampDescA,
 knitr::kable(table(group, exclude = NULL),
  caption="Samples used in this analysis") %>%
  kableExtra::kable_styling(full_width = F)
```

Table 3.2: Samples used in this analysis

group	Freq
Control	778
HCC	555

```
)
```

The features are counts of reads captured by chemical labeling, and indicate the level of 5-hydroxymethylcytosines within each gene body. Cai et al. (2019), Li et al. (2017) and Song et al. (2017) [2–4] all analyze 5hmC gene body counts using standard RNA-Seq methodologies, and we will do the same here.

Note that before conducting any substantive analyses, the data would normally be very carefully examined for any sign of quality variation between groups of samples. This analysis would integrate sample meta data - where and when were the blood samples collected - as well as library preparation and sequencing metrics in order to detect any sign of processing artifacts that may be present in the dataset. This is particularly important when dealing with blood samples as variable DNA quality degradation is a well known challenge that is encountered when dealing with such samples [16]. Although blood specimen handling protocols can be put in place to minimize quality variation [17], variability can never be completely eradicated, especially in the context of blood samples collected by different groups, working in different environments. The problem of variable DNA quality becomes paricularly pernicuous when it is compounded with a confounding factor that sneaks in when the control sample collection events are separated in time and space from the cancer sample collection events; an all too common occurence.

As proper data QC requires an intimate familiarity with the details of data collection and processing, such a task cannot be untertaken here. We will simply run a *minimal set of QC sanity checks* to make sure that there are no apparent systematic effects in the data.

```
featureCountsA <- cbind(
  Train_featureCount,
  Val1_featureCount,
  Val2_featureCount
)[, rownames(sampDescA)]
```

We first look at coverage - make sure there isn't too much disparity of coverage across samples. To detect shared variability, samples can be annotated and ordered according to sample features that may be linked to sample batch processing. Here we the samples have been ordered by group and sample id (an alias of geoAcc).

```
par(mar = c(1, 3, 2, 1))
boxplot(log2(featureCountsA + 1),
  ylim = c(3, 11), ylab='log2 Count',
  staplewex = 0,       # remove horizontal whisker lines
  staplecol = "white", # just to be totally sure :)
  outline = F,         # remove outlying points
  whisklty = 0,        # remove vertical whisker lines
  las = 2, horizontal = F, xaxt = "n",
  border = groupCol[sampDescA$group]
)
legend("top", legend = names(groupCol), text.col = groupCol,
  ncol = 2, bty = "n")
# Add reference lines
SampleMedian <- apply(log2(featureCountsA + 1), 2, median)
abline(h = median(SampleMedian), col = "grey")
axis(side = 4, at = round(median(SampleMedian), 1),
  las = 2, col = "grey", line = -1, tick = F)
```

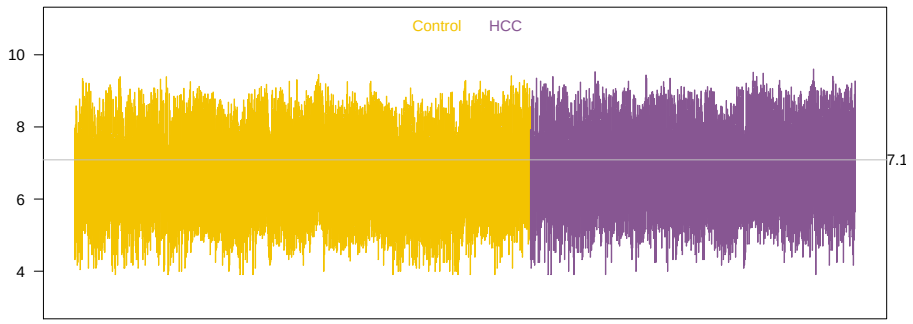

Figure 3.1: Sample log2 count boxplots

```
featureCountsA_quant <- apply(featureCountsA, 2, function(CC) {
  c(quantile(CC, prob = c(.15, (1:3) / 4)), totCovM = sum(CC) / 1e6)
})

featureCountsA_quant2 <- apply(featureCountsA_quant, 1, function(RR) {
  quantile(RR, prob = (1:3) / 4)
})

knitr::kable(featureCountsA_quant2,
  digits = 1,
```

Table 3.3: Coverage Summary - Columns are sample coverage quantiles and total coverage Rows are quartiles across samples

	15%	25%	50%	75%	totCovM
25%	4	24	111	321	5.5
50%	5	30	135	391	6.7
75%	6	35	162	468	8.0

```
  caption = paste(
    "Coverage Summary - Columns are sample coverage quantiles and total coverage",
    "\nRows are quartiles across samples"
  )
) %>% kableExtra::kable_styling(full_width = F)
```

From this table, we see that 25% of the samples have total coverage exceeding 8M reads, 25% of samples have a 15 percentile of coverage lower than 4, etc.

3.2 Differential representation analysis

In the remainder of this section, we will process the data and perform differential expression analysis as outlined in Law et al. (2018) [18]. The main analysis steps are:

- remove lowly expressed genes
- normalize gene expression distributions
- remove heteroscedascity
- fit linear models and examine DE results

It is good practice to perform this differential expression analysis prior to fitting models to get an idea of how difficult it will be to discriminate between samples belonging to the different subgroups. The pipeline outlined in Law et al. (2018) [18] also provides some basic quality assessment opportunities.

Remove lowly expressed genes

Genes that are not expressed at a biologically meaningful level in any condition should be discarded to reduce the subset of genes to those that are of interest, and to reduce the number of tests carried out downstream when looking at differential expression. Carrying un-informative genes may also be a hindrance to classification and other downtream analyses.

To determine a sensible threshold we can begin by examining the shapes of the distributions.

```
par(mar = c(4, 3, 2, 1))
plot(density(lcpm_mtx[, 1]),
  col = groupCol[sampDescA$group[1]],
  lwd = 2, ylim = c(0, .25), las = 2, main = "", xlab = "log2 CPM"
)
abline(v = 0, col = 3)
# After verifying no outliers, can plot a random subset
for (JJ in sample(2:ncol(lcpm_mtx), size = 100)) {
  den <- density(lcpm_mtx[, JJ])
  lines(den$x, den$y, col = groupCol[sampDescA$group[JJ]], lwd = 2)
} # for(JJ
legend("topright", legend = names(groupCol),
  text.col = groupCol, bty = "n")
```

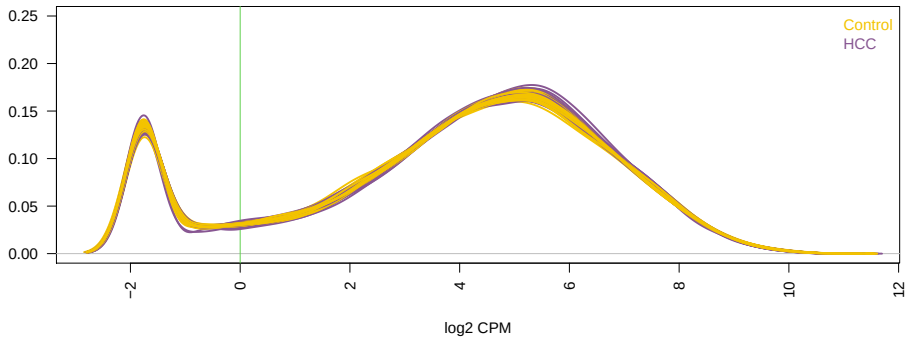

Figure 3.2: Sample log_2 CPM densities

As is typically the case with RNA-Seq data, we notice many weakly represented genes in this dataset. A cpm value of 1 appears to adequatly separate the expressed from the un-expressed genes, but we will be slightly more strict here and require a CPM threshold of 3 . Using a nominal CPM value of 3, genes are deeemed to be `represented` if their expression is above this threshold, and not represented otherwise. For this analysis we will require that genes be `represented` in at least 25 samples across the entire dataset to be retained for downstream analysis. Here, a CPM value of 3 means that a gene is represented if it has at least 9 reads in the sample with the lowest sequencing depth (library

size 2.9 million). Note that the thresholds used here are arbitrary as there are no hard and fast rules to set these by. The voom-plot, which is part of analyses done to remove heteroscedasticity, can be examined to verify that the filtering performed is adequate.

Remove weakly represented genes and replot densities.

$$ Removing 17.5% of genes...

```
par(mar = c(4, 3, 2, 1))
plot(density(lcpm_mtx[, 1]),
  col = groupCol[sampDescA$group[1]],
  lwd = 2, ylim = c(0, .25), las = 2, main = "", xlab = "log2 CPM"
)
#abline(v = 0, col = 3)
# After verifying no outliers, can plot a random subset
for (JJ in sample(2:ncol(lcpm_mtx), size = 100)) {
  den <- density(lcpm_mtx[, JJ])
  lines(den$x, den$y, col = groupCol[sampDescA$group[JJ]], lwd = 2)
} # for(JJ
legend("topright", legend = names(groupCol),
  text.col = groupCol, bty = "n")
```

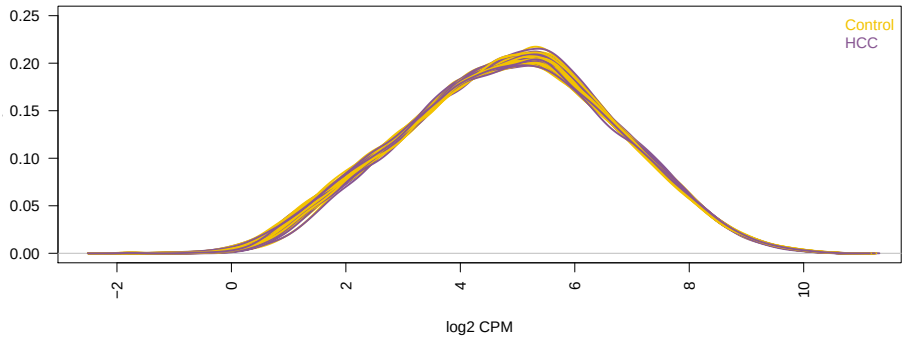

Figure 3.3: Sample log_2 CPM densities after removing weak genes

As another sanity check, we will look at a multidimensional scaling plot of distances between gene expression profiles. We use `plotMDS` in limma package [19]), which plots samples on a two-dimensional scatterplot so that distances on the plot approximate the typical log2 fold changes between the samples.

Before producing the MDS plot we will normalize the distributions. We will store the data into s `DGEList` object as this is convenient when running many of the analyses implemented in the edgeR and limma packages. Call the set 'AF', for set 'A', 'Filtered'.

```
AF_dgel <- edgeR::DGEList(
  counts = featureCountsAF,
  genes = genes_annotAF,
  samples = sampDescA,
  group = sampDescA$group
)
AF_dgel <- edgeR::calcNormFactors(AF_dgel)
AF_lcmp_mtx <- edgeR::cpm(AF_dgel, log = T)

# Save AF_dgel to facilitate restarting
# remove from final version
save(list = "AF_dgel", file = "RData/AF_dgel")
```

Verify that the counts are properly normalized.

```
par(mar = c(1, 3, 2, 1))
boxplot(AF_lcmp_mtx,
  ylim = c(1, 8), ylab='Normalized Log CPM',
  staplewex = 0,       # remove horizontal whisker lines
  staplecol = "white", # just to be totally sure :)
  outline = F,         # remove outlying points
  whisklty = 0,        # remove vertical whisker lines
  las = 2, horizontal = F, xaxt = "n",
  border = groupCol[sampDescA$group]
)
legend("top", legend = names(groupCol), text.col = groupCol,
  ncol = 2, bty = "n")
# Add reference lines
SampleMedian <- apply(AF_lcmp_mtx, 2, median)
abline(h = median(SampleMedian), col = "grey")
axis(side = 4, at = round(median(SampleMedian), 1),
  las = 2, col = "grey", line = -1, tick = F)
```

Proceed with MDS plots.

```
par(mfcol = c(1, 2), mar = c(4, 4, 2, 1), xpd = NA, oma = c(0, 0, 2, 0))

# wo loss of generality, sample 500 samples
# simply a matter of convenience to save time
# remove from final version
set.seed(1)
samp_ndx <- sample(1:ncol(AF_lcmp_mtx), size = 500)
MDS.out <- limma::plotMDS(AF_lcmp_mtx[, samp_ndx],
  col = groupCol[sampDescA$group[samp_ndx]], pch = 1
```

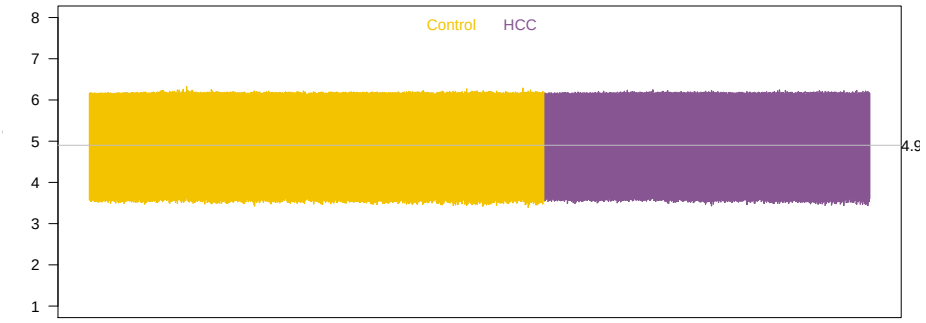

Figure 3.4: Sample log2 count boxplots

```
)
legend("topleft",
  legend = names(groupCol),
  text.col = groupCol, bty = "n"
)

MDS.out <- limma::plotMDS(AF_lcmp_mtx[, samp_ndx],
  col = groupCol[sampDescA$group[samp_ndx]], pch = 1,
  dim.plot = 3:4
)
```

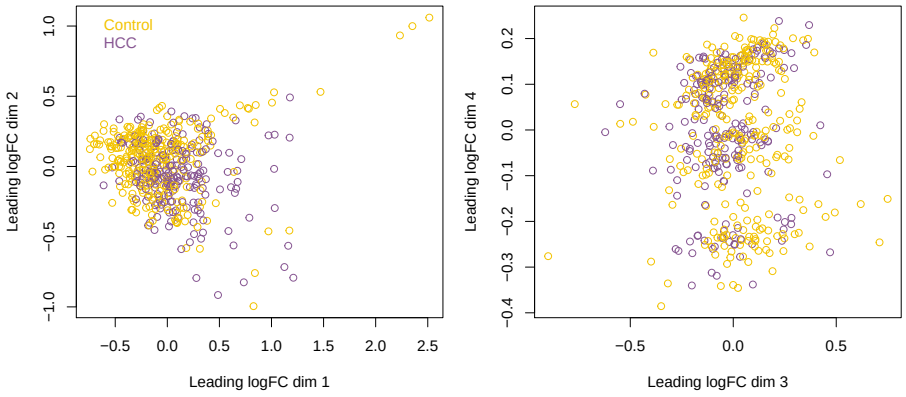

Figure 3.5: MDS plots of log-CPM values

The MDS plot, which is analogous to a PCA plot adapted to gene exression data, does not indicate strong clustering of samples. The fanning pattern observed in the first two dimensions indicates that a few samples are drifting way from the core set, but in no particular direction. There is some structure in the 3rd and 4th dimension plot which should be investigated.
`glMDSPlot` from package `Glimma` provides an interactive MDS plot that can extremely usful for exploration

```
Glimma::glMDSPlot(AF_dgel[, samp_ndx],
  groups = AF_dgel$samples[
    samp_ndx,
    c("group", "trainValGroup", "sampType", "tissue", "title", "stage")
  ],
  main = paste("MDS plot: filtered counts"), #### , Excluding outlier samples"),
  path = ".", folder = figures_DIR,
  html = paste0("GlMDSplot"), launch = F
)
```

Link to glMDSPlot: Here

No obvious factor links the samples in the 3 clusters observed on the 4th MDS dimensions. The percent of variance exaplained by this dimension or $\approx 4\%$. The glMDSPlot indicates further segregation along the 6th dimension. The percent of variance exaplained by this dimension or $\approx 2\%$. Tracking down this source of variability may be quite challenging, especially without having the complete information about the sample attributes and provenance.

Unwanted variability is a well-documented problem in the analysis of RNA-Seq data (see Peixoto et al. (2015) [20]), and many procedures have been proposed to reduce the effect of unwanted variation on RNA-Seq analsys results ([20–22]). There are undoubtedly some similar sources of systematic variation in the 5hmC data, but it is beyond the scope of this work to investigate these in this particular dataset. Given that the clustering of samples occurs in MDS dimensions that explain a small fraction of variability, and that these is no assocation with the factor of interest, HCC vs Control, these sources of variability should not interfere too much with our classification analysis. It would nonetheless be interesting to assess whether downstream results can be improved by removing this variability.

Creating a design matrix and contrasts

Before proceeding with the statistical modeling used for the differential expression analysis, we need to set up a model design matrix.

```
Design_mtx <- model.matrix( ~  -1 + group, data=AF_dgel$samples)
colnames(Design_mtx) <- sub('group', '', colnames(Design_mtx))

cat("colSums(Design_mtx):\n")
```

```
## colSums(Design_mtx):
```

```
colSums(Design_mtx)
```

```
## Control     HCC
##     778     555
```

```
Contrasts_mtx <- limma::makeContrasts(
  HCCvsControl = HCC  - Control,
  levels=colnames(Design_mtx))

cat("Contrasts:\n")
```

```
## Contrasts:
```

```
Contrasts_mtx
```

```
##          Contrasts
## Levels    HCCvsControl
##   Control           -1
##   HCC                1
```

Removing heteroscedasticity from the count data

As for RNA-Seq data, for 5hmC count data the variance is not independent of the mean. In `limma`, the R package we are using for our analyses, linear modeling is carried out on the log-CPM values which are assumed to be normally distributed and the mean-variance relationship is accommodated using precision weights calculated by the voom function. We apply this transformation next.

```
par(mfrow=c(1,1))
filteredCountsAF_voom <- limma::voom(AF_dgel, Design_mtx, plot=T)
```

Note that the voom-plot provides a visual check on the level of filtering performed upstream. If filtering of lowly-expressed genes is insufficient, a drop in variance levels can be observed at the low end of the expression scale due to very small counts.

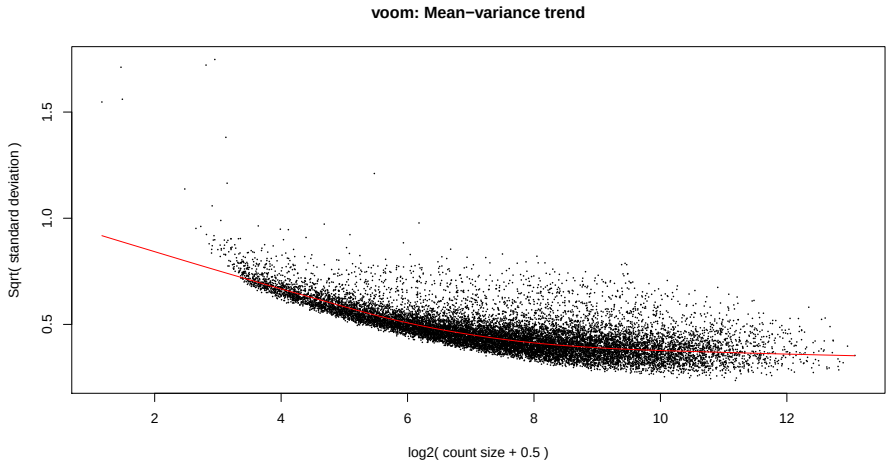

Figure 3.6: Removing heteroscedascity

Fit linear models and examine the results

Having properly filtered and normalized the data, the linear models can be fitted to each gene and the results examined to assess differential expression between the two groups of interest, in our case HCC vs Control.

Table **??** displays the counts of genes in each DE category:

```
filteredCountsAF_voom_fit <- limma::lmFit(filteredCountsAF_voom, Design_mtx)
colnames(filteredCountsAF_voom_fit$coefficients) <- sub("\\(Intercept\\)", "Intercept
colnames(filteredCountsAF_voom_fit$coefficients) )

filteredCountsAF_voom_fit <- limma::contrasts.fit(
   filteredCountsAF_voom_fit, contrasts=Contrasts_mtx)

filteredCountsAF_voom_efit <- limma::eBayes(filteredCountsAF_voom_fit)

filteredCountsAF_voom_efit_dt <-
limma::decideTests(filteredCountsAF_voom_efit,adjust.method = "BH", p.value = 0.05)

knitr::kable(t(summary(filteredCountsAF_voom_efit_dt)),
 caption="DE Results at FDR = 0.05") %>%
 kableExtra::kable_styling(full_width = F)
```

Table 3.4: DE Results at FDR = 0.05

	Down	NotSig	Up
HCCvsControl	5214	5280	5258

Graphical representations of DE results: MD Plots

To summarise results for all genes visually, mean-difference plots (aka MA plot), which display log-FCs from the linear model fit against the average log-CPM values can be generated using the plotMD function, with the differentially expressed genes highlighted.

We may also be interested in whether certain gene features are related to gene identification. Gene GC content, for example, might be of interest.

```
par(mfrow=c(1,3), mar=c(4.5,4.5,2,1),oma=c(1,1,2,0))

# log-fold-change vs ave-expr
limma::plotMD(filteredCountsAF_voom_efit,
 ylim = c(-0.4, 0.4),
 column='HCCvsControl',
 status=filteredCountsAF_voom_efit_dt[,'HCCvsControl'],
 hl.pch = 16, hl.col = c("lightblue", "pink"), hl.cex = .5,
 bg.pch = 16, bg.col = "grey", bg.cex = 0.5,
 main = '',
 xlab = paste0(
    "Average log-expression: IQR=",
    paste(round(quantile(filteredCountsAF_voom_efit$Amean, prob = c(1, 3) / 4), 2),
      collapse = ", "
    )
  ),
  ylab = paste0(
    "log-fold-change: IQR=",
    paste(round(quantile(filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'], prob = c(1, 3
      collapse = ", "
    )
  ),
  legend = F, cex.lab=1.5
)
abline(h = 0, col = "black")
rug(quantile(filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'], prob = c(1, 2, 3) / 4),
  col = "purple",
  ticksize = .03, side = 2, lwd = 2
)
rug(quantile(filteredCountsAF_voom_efit$Amean, prob = c(1, 2, 3) / 4),
```

```
  col = "purple",
  ticksize = .03, side = 1, lwd = 2
)

# log-fold-change vs identification

boxplot(split(
 filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'],
 filteredCountsAF_voom_efit_dt[,'HCCvsControl']),
 outline=F,
 border=c("pink", "grey", "lightblue"), xaxt='n',
 ylab='log-fold-change', ylim=c(-.4, .4),
 cex.lab=1.5
)
axis(side=1, at=1:3, c('down', 'notDE', 'up'), cex.axis=1.5)

# gc vs identification
genes_ndx <- match(rownames(filteredCountsAF_voom_efit), genes_annotAF$Symbol)
if(sum(is.na(genes_ndx))) stop("filteredCountsAF_voom_efit/genes_annotAF: genes mismat
GC_vec <- with(genes_annotAF[genes_ndx,],(G+C)/(A+C+G+T))


boxplot(split(
 GC_vec,
 filteredCountsAF_voom_efit_dt[,'HCCvsControl']),
 outline=F,
 border=c("pink", "grey", "lightblue"), xaxt='n',
 ylab='gene-gc', cex.lab=1.5
)
axis(side=1, at=1:3, c('down', 'notDE', 'up'), cex.axis=1.5)


 #mtext(side=3, outer=T, cex=1.25, "Genes identified at adjusted p-value=0.05")

featureCountsAF_logFC_sum <- sapply(
 split(
 filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'],
 filteredCountsAF_voom_efit_dt[,'HCCvsControl']),
 quantile, prob = (1:3) / 4)

colnames(featureCountsAF_logFC_sum) <- as.character(factor(
 colnames(featureCountsAF_logFC_sum),
 levels=c("-1", "0", "1"),
 labels=c('down', 'notDE', 'up')
))
```

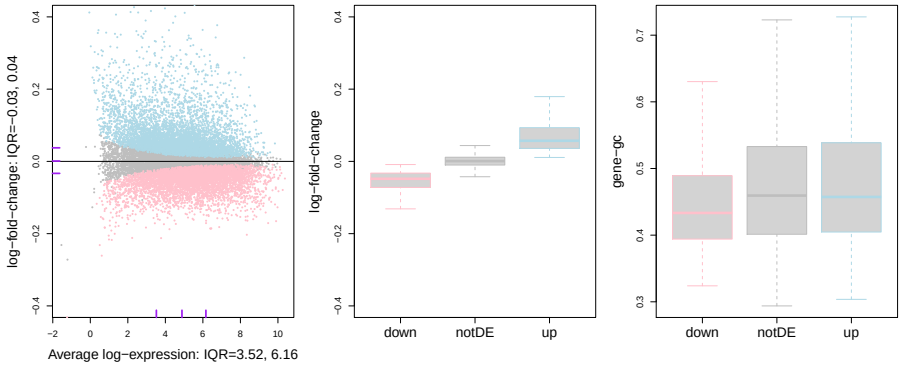

Figure 3.7: HCC vs Control - Genes Identified at FDR = 0,05

Table 3.5: log FC quartiles by gene identification

	down	notDE	up
25%	-0.07	-0.01	0.04
50%	-0.05	0.00	0.06
75%	-0.03	0.01	0.09

```
knitr::kable(featureCountsAF_logFC_sum,
  digits = 2,
  caption = "log FC quartiles by gene identification") %>%
  kableExtra::kable_styling(full_width = F)
```

While many genes are identified, the effect sizes are quite small, which results in a low signal-to-noise ratio context. See Section **??** below.

The log-fold-change distribution for up-represented genes is long-tailed, with many high log fold-change values. By contrast, log-fold-change distribution for down-represented genes closer to symmetric and has few genes with low log fold-change values. We will see how this affects the results of identifying genes with an effect size requirement.

The GC content of down regulated genes tends to be slightly lower than the rest of the genes. A statistical test would find that the difference between the mean of the down regulated gene population is singificantly different than the mean of the other gene population even though the difference is quite small (-0.028).

These asymmetries are minor, but it would still be good to establish that they

relfect biology rather than processing artifacts.

DE genes at 10% fold change

For a stricter definition on significance, one may require log-fold-changes (log-FCs) to be above a minimum value. The treat method (McCarthy and Smyth 2009 [23]) can be used to calculate p-values from empirical Bayes moderated t-statistics with a minimum log-FC requirement. The number of differentially expressed genes are greatly reduced if we impose a minimal fold-change requirement of 10%.

```
filteredCountsAF_voom_tfit <- limma::treat(filteredCountsAF_voom_fit, lfc=log2(1.10))
filteredCountsAF_voom_tfit_dt <- limma::decideTests(filteredCountsAF_voom_tfit)

cat("10% FC Gene Identification Summary - voom, adjust.method = BH, p.value = 0.05:\n")
```

```
## 10% FC Gene Identification Summary - voom, adjust.method = BH, p.value = 0.05:
```

```
summary(filteredCountsAF_voom_tfit_dt)
```

```
##        HCCvsControl
## Down              3
## NotSig        15550
## Up              199
```

```
# log-fold-change vs ave-expr
limma::plotMD(filteredCountsAF_voom_efit,
 ylim = c(-0.5, 0.5),
 column='HCCvsControl',
 status=filteredCountsAF_voom_tfit_dt[,'HCCvsControl'],
 hl.pch = 16, hl.col = c("blue", "red"), hl.cex = .7,
 bg.pch = 16, bg.col = "grey", bg.cex = 0.5,
 main = '',
 xlab = paste0(
    "Average log-expression: IQR=",
    paste(round(quantile(filteredCountsAF_voom_efit$Amean, prob = c(1, 3) / 4), 2),
      collapse = ", "
    )
  ),
  ylab = paste0(
    "log-fold-change: IQR=",
    paste(round(quantile(filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'], pr
      collapse = ", "
```

```
    )
  ),
  legend = F
)
abline(h = 0, col = "black")
rug(quantile(filteredCountsAF_voom_efit$coefficients[, 'HCCvsControl'], prob = c(1, 2, 3) / 4),
  col = "purple",
  ticksize = .03, side = 2, lwd = 2
)
rug(quantile(filteredCountsAF_voom_efit$Amean, prob = c(1, 2, 3) / 4),
  col = "purple",
  ticksize = .03, side = 1, lwd = 2
)
```

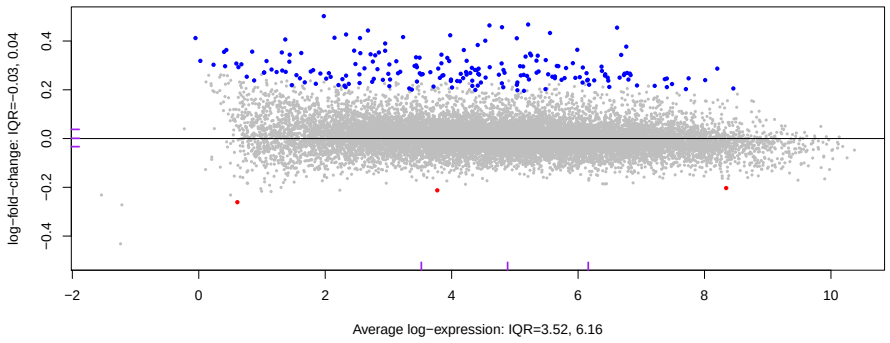

Figure 3.8: HCC vs Control - Identified Genes at FDR = 0,05 and logFC > 10%

As noted above, the log-fold-change distribution for the up-represented genes is long-tailes in comparison to log-fold-change distribution for the down-represented genes. As a result fewer down-represented than up-regulated genes are identified when a minimum log-FC requirement is imposed.

3.3 Signal-to-noise ratio regime

In Hastie et al. (2017) [7]) results from `lasso` fits are compared with `best subset` and `forward selection` fits and it is argued that while `best subset` is optimal for high signal-to-noise regimes, the lasso gains some competitive advantage when the prevailing signal-to-noise ratio of the dataset is lowered.

We can extract sigma and signal from the fit objects to get SNR values for each gene to see in what SNR regime the 5hmC gene body data are.

```
lib.size <- colSums(AF_dgel$counts)

fit <- filteredCountsAF_voom_efit
sx <- fit$Amean + mean(log2(lib.size + 1)) - log2(1e+06)
sy <- sqrt(fit$sigma)

CV <- sy/sx
```

```
Effect <- abs(filteredCountsAF_voom_efit$coefficients[,'HCCvsControl'])
Noise <- filteredCountsAF_voom_efit$sigma
SNR <- Effect/Noise

plot(spatstat::CDF(density(SNR)),
  col = 1, lwd = 2, ylab = "Prob(SNR<x)",
  xlim = c(0, 0.2)
)

SNR_quant <- quantile(SNR, prob=c((1:3)/4,.9))
rug(SNR_quant,
    lwd = 2, ticksize = 0.05, col = 1
  )
```

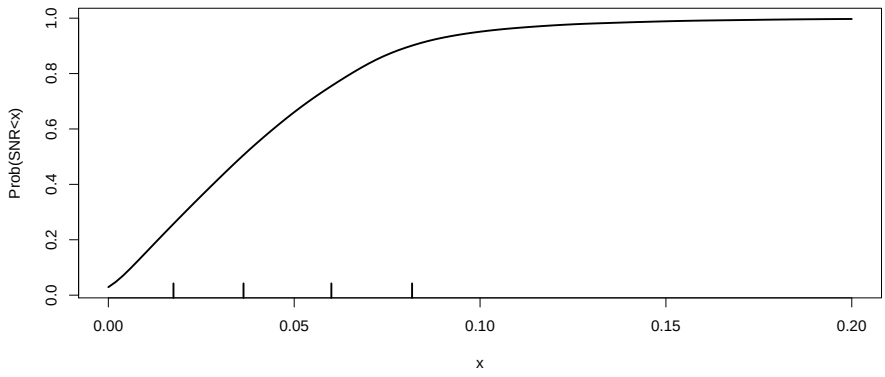

Figure 3.9: Cumulative Distribution of SNR - rug = 25, 50, 75 and 90th percentile

Table 3.6: SNR Quantiles

25%	50%	75%	90%
0.018	0.036	0.06	0.082

```
knitr::kable(t(SNR_quant),
  digits = 3,
  caption = paste(
    "SNR Quantiles")
) %>% kableExtra::kable_styling(full_width = F)
```

These SNR values are in the range where the lasso and relaxed lasso gain some advantage over best subset and forward selection fits (see Hastie et al. (2017) [7]).

Chapter 4

The bet on sparsity

In this section we explore various fits that can be computed and analyzed with tools provided in the `glmnet` package. Refer to the Glmnet Vignette for a quick reference guide. We focus our analyses on lasso fits which tend to favor sparse models.

4.1 Cross-validation analysis setup

```
K_FOLD <- 10
trainP <- 0.8
```

First we divide the analysis dataset into `train` and `test` in a 4:1 ratio.

```
set.seed(1)
train_sampID_vec <- with(AF_dgel$samples,
AF_dgel$samples$sampID[caret::createDataPartition(y=group, p=trainP, list=F)]
)

test_sampID_vec <- with(AF_dgel$samples,
setdiff(sampID, train_sampID_vec)
)

train_group_vec <- AF_dgel$samples[train_sampID_vec, 'group']
names(train_group_vec) <- AF_dgel$samples[train_sampID_vec, 'sampID']

test_group_vec <- AF_dgel$samples[test_sampID_vec, 'group']
names(test_group_vec) <- AF_dgel$samples[test_sampID_vec, 'sampID']
```

Table 4.1: Train set

train_group_vec	Freq
Control	623
HCC	444

Table 4.2: Test set

test_group_vec	Freq
Control	155
HCC	111

```
knitr::kable(table(train_group_vec),
  caption="Train set") %>%
   kableExtra::kable_styling(full_width = F)
```

```
knitr::kable(table(test_group_vec),
  caption="Test set") %>%
   kableExtra::kable_styling(full_width = F)
```

```
train_lcpm_mtx <- t(lcpm_mtx[,train_sampID_vec])
test_lcpm_mtx <- t(lcpm_mtx[,test_sampID_vec])
```

We explore some glmnet fits and the "bet on sparsity". We consider three models, specified by the value of the **alpha** parameter in the elastic net parametrization:
- lasso: $\alpha = 1.0$ - sparse models
- ridge $\alpha = 0$ - shrunken coefficients models
- elastic net: $\alpha = 0.5$ - semi sparse model

Some questions of interest include:

- How sparse are models enabling good 5hmC classification of Early HCC vs Control samples?
- Does the shrunken relaxed lasso (aka the blended mix) improve performance in this case?
- Is the degree of sparsity, or the size of the model, a stable feature of the problem and data set?

In this analysis, we will only evaluate models in terms of model sparsity, stability and performance. We leave the question of significance testing of hypotheses about model parameters completely out. See Lockhart et al. (2014) [24] and Wassermam (2014) [25] for a discussion of this topic.

In this section we look at the relative performance and sparsity of the models considered. The effect of the size of the sample set on the level and stability of performance will be investigated in the next section.

First we create folds for 10-fold cross-validation of models fitted to training data. We'll use caret::createFolds to assign samples to folds while keeping the outcome ratios constant across folds.

```
# This is too variable, both in terms of fold size And composition
#foldid_vec <- sample(1:10, size=length(train_group_vec), replace=T)

set.seed(1)
train_foldid_vec <- caret::createFolds(
 factor(train_group_vec),
 k=K_FOLD,
 list=F)

# train_foldid_vec contains the left-out IDs
# the rest are kept
fold_out_tbl <- sapply(split(train_group_vec, train_foldid_vec),
  table)
rownames(fold_out_tbl) <- paste(rownames(fold_out_tbl), '- Out')

fold_in_tbl <- do.call('cbind', lapply(sort(unique(train_foldid_vec)),
  function(FOLD) table(train_group_vec[train_foldid_vec != FOLD])))
rownames(fold_in_tbl) <- paste(rownames(fold_in_tbl), '- In')
colnames(fold_in_tbl) <- as.character(sort(unique(train_foldid_vec)))


knitr::kable(rbind(fold_in_tbl, fold_out_tbl[,colnames(fold_in_tbl)]),
  caption="training samples fold composition") %>%
   kableExtra::kable_styling(full_width = F)
```

Note that the folds identify samples that are left-out of the training data for each fold fit.

Table 4.3: training samples fold composition

	1	2	3	4	5	6	7	8	9	10
Control - In	561	561	561	560	561	561	560	561	560	561
HCC - In	399	400	400	399	400	399	400	399	400	400
Control - Out	62	62	62	63	62	62	63	62	63	62
HCC - Out	45	44	44	45	44	45	44	45	44	44

4.2 Fit and compare models

`glmnet` provides cross-validation methods to pick the parameter **lambda** which controls to size of the penalty function. The "one standard error rule" produces a model with fewer predictors then the minimum cv error model. On the training data, this usually results in increased MSE and more biased parameter estimates (see Engebretsen et al. (2019) [26] for example). The question of iterest though is the performance on unseen data; not on the training data. In the analysis below, we compare the cv error rates with out-of-fold and test set error rates. The results show that out-of-fold error rates computed from the training data are good indicators of test set error rates, and that the one standard error rule models do as well as the minimim cv error models for the lasso, which has the best overall performance.

4.2.1 Logistic regression in `glmnet`

`glmnet` provides functionality to extract various predicted of fitted values from calibrated models. Note that some folks make a distinction between **fitted** or **estimated** values for sample points in the training data versus **predicted** values for sample points that are not in the training dataset. `glmnet` makes no such distinction and the `predict` function is used to produce both fitted as well as predicted values. When predict is invoked to make predictions for design points that are part of the training dataset, what is returned are fitted values. When predict is invoked to make predictions for design points that are not part of the training dataset, what is returned are predicted values.

For logistic regressions, which is the model fitted in a regularized fashion when models are fitted by glmnet with the parameter `family='binomial'`, three fitted or predicted values can be extracted at a given design point. Suppose our response variable Y is either 0 or 1 (Control or HCC in our case). These are specified by the `type` parameter. `type='resp'` returns the fitted or predicted probability of $Y = 1$. `type='class'` returns the fitted or predicted class for the design point, which is simply dichotomizing the response: class = 1 if the fitted or predicted probability is greater than 0.5 (check to make sure class is no the Bayes estimate). `type='link'` returns the fitted or predicted value of

the linear predictor $\beta' x$. The relationship between the linear predictor and the response can be derided from the logistic regression model:

$$P(Y = 1|x, \beta) = g^{-1}(\beta' x) = h(\beta' x) = \frac{e^{\beta' x}}{1 + e^{\beta' x}}$$

where g is the link function, g^{-1} the mean function. The link function is given by:

$$g(y) = h^{-1}(y) = ln(\frac{y}{1-y})$$

This link function is called the *logit* function, and its inverse the *logistic* function.

```
logistic_f <- function(x) exp(x)/(1+exp(x))
```

We should also point out that the cv error rates quoted in various `glmnet` summaries are computed from out-of-fold predictions. In other words, we cannot recover the `cvm` component of a glmnet fit object by comparing class predictions extracted by invoking `predict()` with paramters `newx = train_data` and `type = 'class'` to the true class labels.

`glmnet` fitting functions have a parameter, *keep*, which instructs the fitting function to keep the `out-of-fold`, or `prevalidated`, predictions as part of the returned object. The `out-of-fold` predictions are predicted values for the samples in the left-out folds, pooled across all cv folds. For each hyper-parameter specification, we get one full set of `out-of-fold` predictions for the training set samples. Performance assessments based on these values are usually more generalizable. The `cvm` component of glmnet fitted objects are derived from these `out-of-fold` predictions. See Hofling and Tibshirani (2008) [27] for a description of the use of pre-validation in model assessment.

```
start_time <-  proc.time()

cv_lasso <- glmnet::cv.glmnet(
 x=train_lcpm_mtx,
 y=train_group_vec,
 foldid=train_foldid_vec,
 alpha=1,
 family='binomial',
 type.measure = "class",
 keep=T,
 nlambda=30
)

message("lasso time: ", round((proc.time() - start_time)[3],2),"s")
```

```
## lasso time: 12.96s
```

```
start_time <-  proc.time()

cv_ridge <- glmnet::cv.glmnet(
 x=train_lcpm_mtx,
 y=train_group_vec,
 foldid=train_foldid_vec,
 alpha=0,
 family='binomial',
 type.measure = "class",
 keep=T,
 nlambda=30
)

message("ridge time: ", round((proc.time() - start_time)[3],2),"s")
```

```
## ridge time: 105.8s
```

```
start_time <-  proc.time()

cv_enet <- glmnet::cv.glmnet(
 x=train_lcpm_mtx,
 y=train_group_vec,
 foldid=train_foldid_vec,
 alpha=0.5,
 family='binomial',
 type.measure = "class",
 keep=T,
 nlambda=30
)

message("enet time: ", round((proc.time() - start_time)[3],2),"s")
```

```
## enet time: 13.53s
```

```
plot_cv_f <- function(cv_fit, Nzero=T, ...) {

 suppressPackageStartupMessages(require(glmnet))

 # No nonger used
 #lambda.1se_p <- cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se]
 #lambda.min_p <- cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min]
```

```r
# Get oof error - cv errors produced by extraction method ARE oof!!!
ndx_1se <- match(cv_fit$lambda.1se,cv_fit$lambda)
train_oofPred_1se_vec <- ifelse(
 logistic_f(cv_fit$fit.preval[,ndx_1se]) > 0.5, 'HCC', 'Control')
train_oofPred_1se_error <- mean(train_oofPred_1se_vec != train_group_vec)

ndx_min <- match(cv_fit$lambda.min,cv_fit$lambda)
train_oofPred_min_vec <- ifelse(
 logistic_f(cv_fit$fit.preval[,ndx_min]) > 0.5, 'HCC', 'Control')
train_oofPred_min_error <- mean(train_oofPred_min_vec != train_group_vec)

# Get test set error
test_pred_1se_vec <- predict(
 cv_fit,
 newx=test_lcpm_mtx,
 s="lambda.1se",
 type="class"
)
test_pred_1se_error <- mean(test_pred_1se_vec != test_group_vec)

test_pred_min_vec <- predict(
 cv_fit,
 newx=test_lcpm_mtx,
 s="lambda.min",
 type="class"
)
test_pred_min_error <- mean(test_pred_min_vec != test_group_vec)


plot(
 log(cv_fit$lambda),
 cv_fit$cvm,
 pch=16,col="red",
 xlab='',ylab='',
 ...
)
abline(v=log(c(cv_fit$lambda.1se, cv_fit$lambda.min)))
if(Nzero)
axis(side=3, tick=F, at=log(cv_fit$lambda),
 labels=cv_fit$nzero, line = -1
)
LL <- 2
#mtext(side=1, outer=F, line = LL, "log(Lambda)")
#LL <- LL+1
mtext(side=1, outer=F, line = LL, paste(
```

```
  #ifelse(Nzero, paste("1se p =", lambda.1se_p),''),
  "1se: train =", round(100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se], 1),
  ##"oof =", round(100*train_oofPred_1se_error, 1), ### REDUNDANT
  "test =", round(100*test_pred_1se_error, 1)
 ))
 LL <- LL+1
 mtext(side=1, outer=F, line = LL, paste(
  #ifelse(Nzero, paste("min p =", lambda.min_p),''),
  "min: train =", round(100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min], 1),
  ##"oof =", round(100*train_oofPred_min_error, 1),  ### REDUNDANT
  "test =", round(100*test_pred_min_error, 1)
 ))

 tmp <-
 cbind(
  error_1se = c(
   p = cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se],
   train  = 100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se],
   #train_oof = 100*train_oofPred_1se_error,  ### REDUNANT
   test = 100*test_pred_1se_error),
  error_min = c(
   p = cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min],
   train  = 100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min],
   #train_oof = 100*train_oofPred_min_error, ### REDUNDSANT
   test = 100*test_pred_min_error)
  )
  # Need to fix names
  rownames(tmp) <- c('p', 'train', 'test')
  tmp
}
```

Examine model performance.

```
par(mfrow=c(1,3), mar=c(5, 2, 3, 1), oma=c(3,2,0,0))

lasso_errors_mtx <- plot_cv_f(cv_lasso, ylim=c(0,.5))
```

```
## Warning: package 'glmnet' was built under R version 4.0.2
```

```
title('lasso')

rifge_errors_mtx <- plot_cv_f(cv_ridge, Nzero=F, ylim=c(0,.5))
title('ridge')
```

```
enet_errors_mtx <-  plot_cv_f(cv_enet, ylim=c(0,.5))
title('enet')

mtext(side=1, outer=T, cex=1.25, 'log(Lambda)')
mtext(side=2, outer=T, cex=1.25, cv_lasso$name)
```

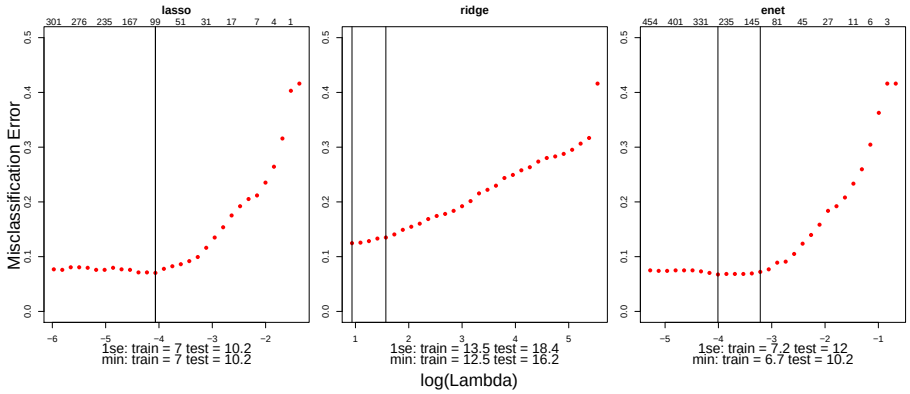

Figure 4.1: compare fits

```
errors_frm <- data.frame(
  lasso = lasso_errors_mtx, ridge = rifge_errors_mtx, enet = enet_errors_mtx
)
colnames(errors_frm) <- sub('\\.error','', colnames(errors_frm))

knitr::kable(t(errors_frm),
 caption = 'Misclassifiaction error rates',
 digits=1) %>%
  kableExtra::kable_styling(full_width = F)
```

We see that the lasso and enet models do better than the ridge model. The *one standard error rule* (1se) lambda lasso fit is only slightly more parsimonious than the *1se* elastic net fit, but its test set accuracy is better. The *min* lambda models have lower test set error rates than the more parsinomious *1se* models in the ridge and elastice net case. (for the lasso, 1se and minimum lambda rules give the same model). The mimimum lambda elastic net fit performs as well as the lasso model, but is much less parsimonious.

Note that the cv estiamtes of misclassification error produced by the glmnet extraction method are `out-of-fold` estimated error rates. In this particular dataset, these estimates of error are optimistic in comparison the the test set error rates.

Table 4.4: Misclassifiaction error rates

	p	train	test
lasso_1se	99	7.0	10.2
lasso_min	99	7.0	10.2
ridge_1se	15752	13.5	18.4
ridge_min	15752	12.5	16.2
enet_1se	118	7.2	12.0
enet_min	278	6.7	10.2

4.3 The relaxed lasso and blended mix models

Next we look at the so-called `relaxed lasso` model, and the `blended mix` which is an optimized shrinkage between the relaxed lasso and the regular lasso. See (**??**) in Section **??**.

```
library(glmnet)

cv_lassoR_sum <- print(cv_lassoR)
```

```
##
## Call:  glmnet::cv.glmnet(x = train_lcpm_mtx, y = train_group_vec, type.measure = "cl
##
## Measure: Misclassification Error
##
##     Gamma Lambda Measure       SE Nonzero
## min   0.5 0.0379 0.06748 0.005274      35
## 1se   0.5 0.0379 0.06748 0.005274      35
```

```
plot(cv_lassoR)
```

```
# only report  1se
ndx_1se <- match(cv_lassoR$lambda.1se, cv_lassoR$lambda)
ndx_min <- match(cv_lassoR$lambda.min, cv_lassoR$lambda)

# only show 1se anyway
# if(ndx_1se != ndx_min) stop("lambda.1se != lambda.min")


# train oof data - NOT CLEAR WHY THESE DIFFER FROM CV ERRORS EXTRACTED FROM MODEL
# Get relaxed lasso (gamma=0) oof error
train_oofPred_relaxed_1se_vec <- ifelse(
```

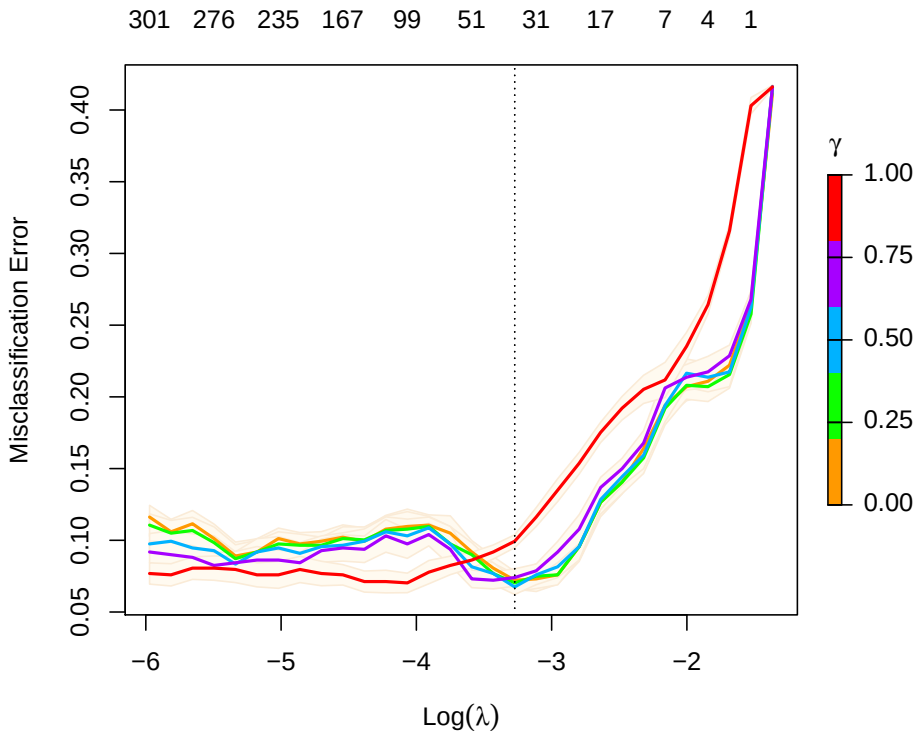

Figure 4.2: Relaxed lasso fit

```
  logistic_f(cv_lassoR$fit.preval[["g:0"]][, ndx_1se]) > 0.5, "HCC", "Control"
)
train_oofPred_relaxed_1se_error <- mean(train_oofPred_relaxed_1se_vec != train_group_ve

# blended mix (gamma=0.5)
train_oofPred_blended_1se_vec <- ifelse(
  logistic_f(cv_lassoR$fit.preval[["g:0.5"]][, ndx_1se]) > 0.5, "HCC", "Control"
)
train_oofPred_blended_1se_error <- mean(train_oofPred_blended_1se_vec != train_group_ve


# Test set error - relaxed
test_pred_relaxed_1se_vec <- predict(
  cv_lassoR,
  newx = test_lcpm_mtx,
  s = "lambda.1se",
  type = "class",
  gamma = 0
)
test_pred_relaxed_1se_error <- mean(test_pred_relaxed_1se_vec != test_group_vec)

# Test set error - blended
test_pred_blended_1se_vec <- predict(
  cv_lassoR,
  newx = test_lcpm_mtx,
  s = "lambda.1se",
  type = "class",
  gamma = 0.5
)
test_pred_blended_1se_error <- mean(test_pred_blended_1se_vec != test_group_vec)


cv_lassoR_1se_error <- cv_lassoR$cvm[cv_lassoR$lambda==cv_lassoR$lambda.min]

cv_blended_statlist <- cv_lassoR$relaxed$statlist[['g:0.5']]
cv_blended_1se_error <- cv_blended_statlist$cvm[cv_blended_statlist$lambda==
   cv_lassoR$relaxed$lambda.1se]



knitr::kable(t(data.frame(
  train_relaxed  = cv_lassoR_1se_error,
  train_blended  = cv_blended_1se_error,
  #train_relaxed_oof = train_oofPred_relaxed_1se_error,
  #train_blended_oof = train_oofPred_blended_1se_error,
```

Table 4.5: Relaxed lasso and blended mix error rates

train_relaxed	7.0
train_blended	6.7
test_relaxed	10.9
test_blended	10.2

```
  test_relaxed  = test_pred_relaxed_1se_error,
  test_blended  = test_pred_blended_1se_error
)) * 100,
digits = 1,
caption = "Relaxed lasso and blended mix error rates"
) %>%
  kableExtra::kable_styling(full_width = F)
```

The relaxed lasso and blended mix error rates are comparable to the regular lasso fit error rate. We see here too that the reported cv error rates are optimistic.

The *1se* lambda rule applied to the relaxed lasso fit selected a model with 99 features, while for the blended mix model (See (**??**) in Section **??**) the *1se* lambda rule selected 35 features (vertical dotted reference line in Figure **??**). This feature is pointed out in the glmnet 3.0 vignette: *The debiasing will potentially improve prediction performance, and CV will typically select a model with a smaller number of variables.*

4.4 Examination of sensitivity vs specificity

In the results above we reported error rates without inspecting the sensitivity versus specificity trade-off. ROC curves can be examined to get a sense of the trade-off.

4.4.1 Training data out-of-fold ROC curves

```
# train
# lasso
ndx_1se <- match(cv_lasso$lambda.1se,cv_lasso$lambda)
train_lasso_oofProb_vec <- logistic_f(cv_lasso$fit.preval[,ndx_1se])
train_lasso_roc <- pROC::roc(
 response = as.numeric(train_group_vec=='HCC'),
 predictor = train_lasso_oofProb_vec)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
# enet
ndx_1se <- match(cv_enet$lambda.1se,cv_enet$lambda)
train_enet_oofProb_vec <- logistic_f(cv_enet$fit.preval[,ndx_1se])
train_enet_roc <- pROC::roc(
 response = as.numeric(train_group_vec=='HCC'),
 predictor = train_enet_oofProb_vec)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
# lasso - relaxed
ndx_1se <- match(cv_lassoR$lambda.1se,cv_lassoR$lambda)
train_relaxed_oofProb_vec <- logistic_f(cv_lassoR$fit.preval[['g:0']][,ndx_1se])
train_relaxed_roc <- pROC::roc(
 response = as.numeric(train_group_vec=='HCC'),
 predictor = train_relaxed_oofProb_vec)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
# blended mix (gamma=0.5)
ndx_1se <- match(cv_lassoR$lambda.1se,cv_lassoR$lambda)
train_blended_oofProb_vec <- logistic_f(cv_lassoR$fit.preval[['g:0.5']][,ndx_1se])
train_blended_roc <- pROC::roc(
 response = as.numeric(train_group_vec=='HCC'),
 predictor = train_blended_oofProb_vec)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
plot(train_lasso_roc, col=col_vec[1])
lines(train_enet_roc, col=col_vec[2])
lines(train_relaxed_roc, col=col_vec[3])
lines(train_blended_roc, col=col_vec[4])

legend('bottomright', title='AUC',
 legend=c(
  paste('lasso =', round(train_lasso_roc[['auc']],3)),
  paste('enet =', round(train_enet_roc[['auc']],3)),
```

```
  paste('relaxed =', round(train_relaxed_roc[['auc']],3)),
  paste('blended =', round(train_blended_roc[['auc']],3))
 ),
 text.col = col_vec[1:4],
 bty='n'
)
```

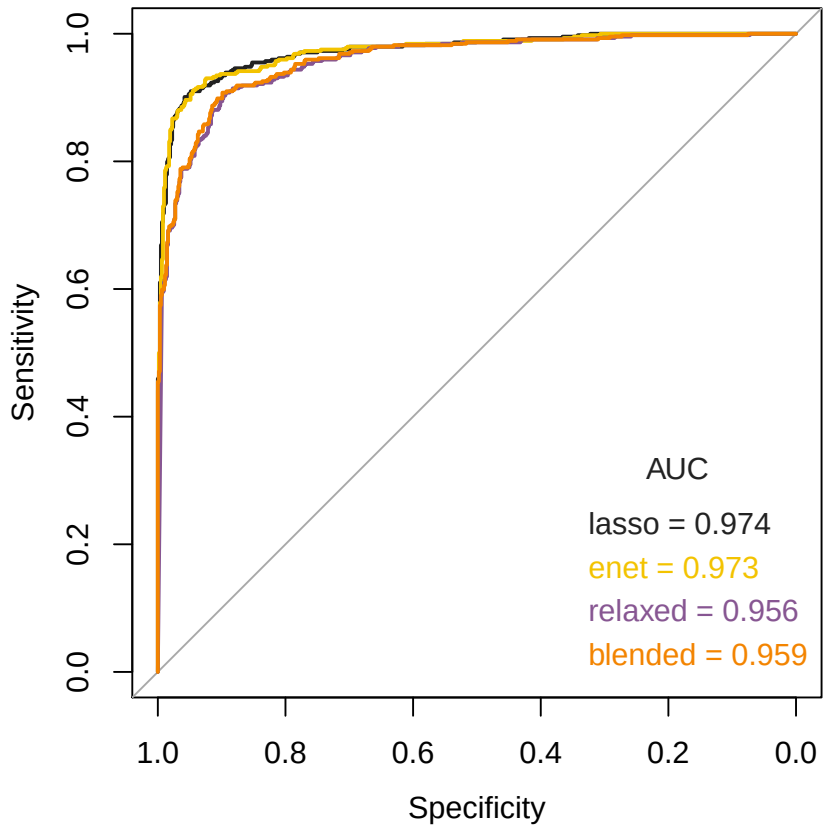

Figure 4.3: Train data out-of-sample ROCs

Compare thresholds for 90% Specificity:

```
lasso_ndx <- with(as.data.frame(pROC::coords(train_lasso_roc, transpose=F)),
  min(which(specificity >= 0.9)))
```

Table 4.6: Specificity = .90 Coordinates

	threshold	specificity	sensitivity
lasso	0.337	0.9	0.932
enet	0.347	0.9	0.937
relaxed	0.003	0.9	0.894
blended	0.031	0.9	0.899

```
enet_ndx <- with(as.data.frame(pROC::coords(train_enet_roc, transpose=F)),
  min(which(specificity >= 0.9)))

lassoR_ndx <- with(as.data.frame(pROC::coords(train_relaxed_roc, transpose=F)),
  min(which(specificity >= 0.9)))

blended_ndx <- with(as.data.frame(pROC::coords(train_blended_roc, transpose=F)),
  min(which(specificity >= 0.9)))

 spec90_frm <- data.frame(rbind(
 lasso=as.data.frame(pROC::coords(train_lasso_roc, transpose=F))[lasso_ndx,],
 enet=as.data.frame(pROC::coords(train_enet_roc, transpose=F))[enet_ndx,],
 relaxed=as.data.frame(pROC::coords(train_relaxed_roc, transpose=F))[lassoR_ndx,],
 blended=as.data.frame(pROC::coords(train_blended_roc, transpose=F))[blended_ndx,]
))


knitr::kable(spec90_frm,
  digits=3,
  caption="Specificity = .90 Coordinates"
) %>%
  kableExtra::kable_styling(full_width = F)
```

This is strange.

```
par(mfrow = c(2, 2), mar = c(3, 3, 2, 1), oma = c(2, 2, 2, 2))

# lasso
plot(density(train_lasso_oofProb_vec[train_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(train_lasso_oofProb_vec[train_group_vec == "HCC"]),
  col = "red"
)
title("lasso")
legend("topright", legend = c("Control", "HCC"), text.col = c("green", "red"))
```

```
# enet
plot(density(train_enet_oofProb_vec[train_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(train_enet_oofProb_vec[train_group_vec == "HCC"]),
  col = "red"
)
title("enet")

# lassoR
plot(density(train_relaxed_oofProb_vec[train_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(train_relaxed_oofProb_vec[train_group_vec == "HCC"]),
  col = "red"
)
title("lassoR")

# blended
plot(density(train_blended_oofProb_vec[train_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(train_blended_oofProb_vec[train_group_vec == "HCC"]),
  col = "red"
)
title("blended")

mtext(side = 1, outer = T, "out-of-fold predicted probability", cex = 1.25)
mtext(side = 2, outer = T, "density", cex = 1.25)
```

The relaxed lasso fit results in essentially dichotomized predicted probability distribution - predicted probabilities are very close to 0 or 1.

Look at test data ROC curves.

```
# plot all
plot(test_lasso_roc, col = col_vec[1])
lines(test_enet_roc, col = col_vec[2])
lines(test_relaxed_roc, col = col_vec[3])
lines(test_blended_roc, col = col_vec[4])

legend("bottomright",
  title = "AUC",
  legend = c(
    paste("lasso =", round(test_lasso_roc[["auc"]], 3)),
```

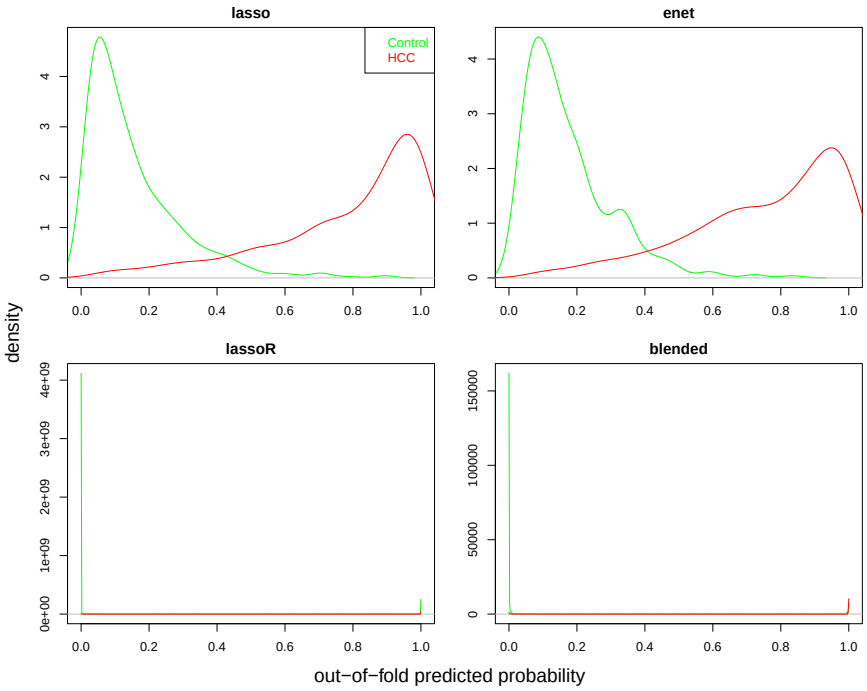

Figure 4.4: Train data out-of-fold predicted probabilities

```
    paste("enet =", round(test_enet_roc[["auc"]], 3)),
    paste("relaxed =", round(test_relaxed_roc[["auc"]], 3)),
    paste("blended =", round(test_blended_roc[["auc"]], 3))
  ),
  text.col = col_vec[1:4],
  bty='n'
)
```

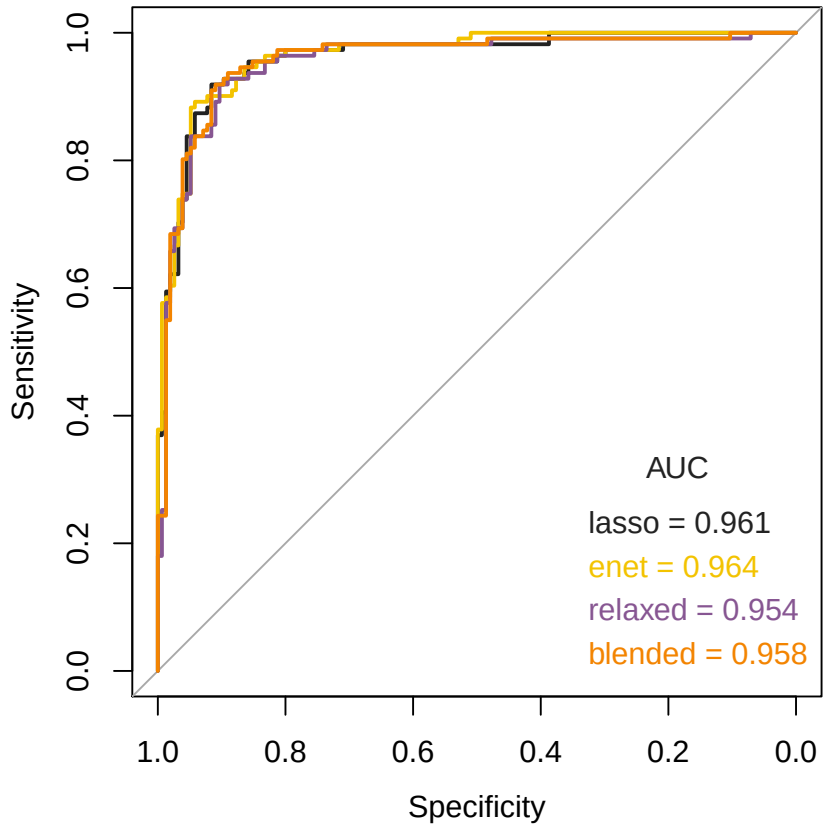

Figure 4.5: Test data out-of-sample ROCs

Look at densities of predicted probabilities.

```
par(mfrow = c(2, 2), mar = c(3, 3, 2, 1), oma = c(2, 2, 2, 2))

# lasso
plot(density(test_lasso_predProb_vec[test_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(test_lasso_predProb_vec[test_group_vec == "HCC"]),
  col = "red"
)
title("lasso")
legend("topright", legend = c("Control", "HCC"), text.col = c("green", "red"))

# enet
plot(density(test_enet_predProb_vec[test_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(test_enet_predProb_vec[test_group_vec == "HCC"]),
  col = "red"
)
title("enet")

# relaxed
plot(density(test_relaxed_predProb_vec[test_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(test_relaxed_predProb_vec[test_group_vec == "HCC"]),
  col = "red"
)
title("relaxed")

#sapply(split(test_relaxed_predProb_vec, test_group_vec), summary)


# blended
plot(density(test_blended_predProb_vec[test_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(test_blended_predProb_vec[test_group_vec == "HCC"]),
  col = "red"
)
title("blended")

mtext(side = 1, outer = T, "test set predicted probability", cex = 1.25)
mtext(side = 2, outer = T, "density", cex = 1.25)
```

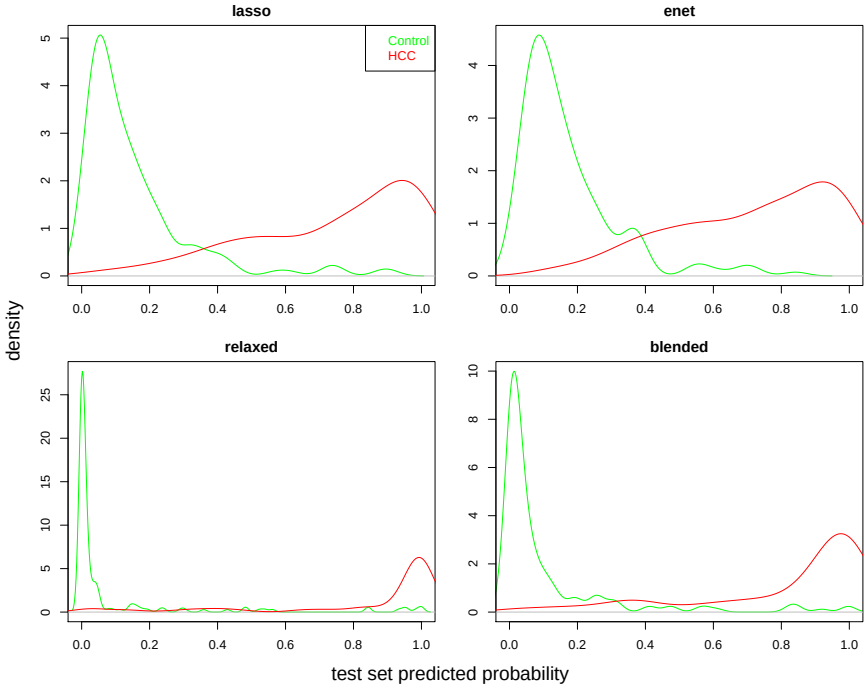

Figure 4.6: Test data out-of-fold predicted probabilities

```r
# Define plotting function
bxpPredProb_f <- function(cv_fit, Gamma=NULL) {
  # Train - preval is out-of-fold linear predictor for training design points
  onese_ndx <- match(cv_fit$lambda.1se, cv_fit$lambda)
  if(is.null(Gamma))
   train_1se_preval_vec <- cv_fit$fit.preval[, onese_ndx] else
   train_1se_preval_vec <- cv_fit$fit.preval[[Gamma]][, onese_ndx]

  train_1se_predProb_vec <- logistic_f(train_1se_preval_vec)

  # Test
  test_1se_predProb_vec <- predict(
    cv_fit,
    newx = test_lcpm_mtx,
    s = "lambda.1se",
    type = "resp"
  )

  tmp <- c(
    train = split(train_1se_predProb_vec, train_group_vec),
    test = split(test_1se_predProb_vec, test_group_vec)
  )
  names(tmp) <- paste0("\n", sub("\\.", "\n", names(tmp)))

  boxplot(tmp)
}

par(mfrow = c(2, 2), mar = c(5, 3, 2, 1), oma = c(2, 2, 2, 2))

bxpPredProb_f(cv_lasso)
title('lasso')

bxpPredProb_f(cv_enet)
title('enet')

bxpPredProb_f(cv_lassoR, Gamma='g:0')
title('relaxed')

bxpPredProb_f(cv_lassoR, Gamma='g:0.5')
title('blended')
```

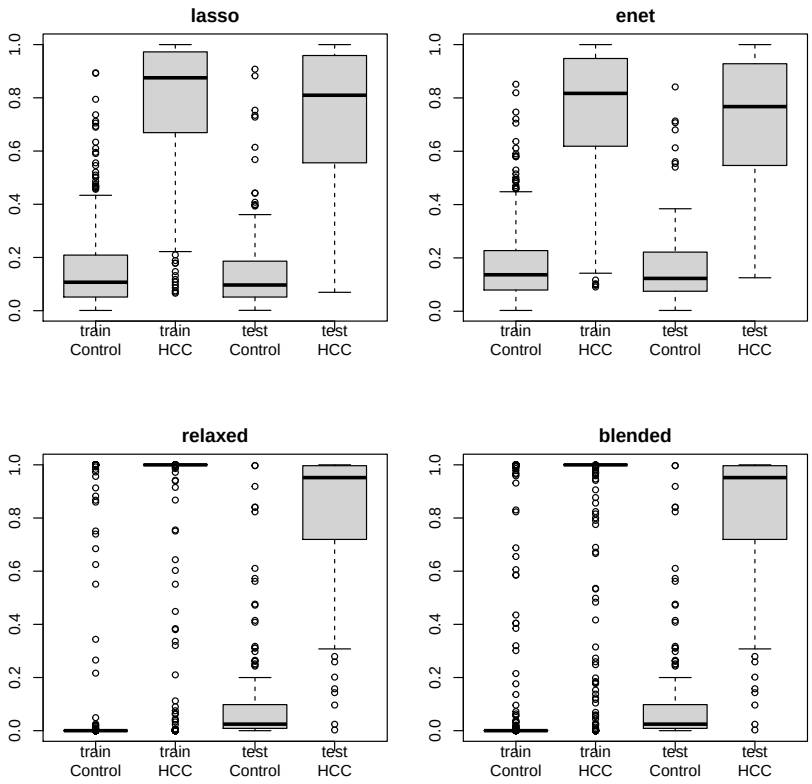

Figure 4.7: Predicted Probabilities - Train and Test

4.5 Compare predictions at misclassified samples

It is useful to examine classification errors more carefully. If models have different failure modes, one might get improved performance by combining model predictions. Note that the models considered here are not expected to compliment each other usefully as they are too similar in nature.

```
# NOTE: here we use computred oofClass rather than predClass
# as predClass extracted from predict() are fitted values.

# train - oof
ndx_1se <- match(cv_lasso$lambda.1se,cv_lasso$lambda)
train_lasso_oofProb_vec <- logistic_f(cv_lasso$fit.preval[,ndx_1se])
train_lasso_oofClass_vec <- ifelse(
   train_lasso_oofProb_vec > 0.5, 'HCC', 'Control')

ndx_1se <- match(cv_enet$lambda.1se,cv_enet$lambda)
train_enet_oofProb_vec <- logistic_f(cv_enet$fit.preval[,ndx_1se])
train_enet_oofClass_vec <- ifelse(
   train_enet_oofProb_vec > 0.5, 'HCC', 'Control')

# RECALL: cv_lassoR$nzero[cv_lassoR$lambda==cv_lassoR$lambda.1se]
# train - oof
ndx_1se <- match(cv_lassoR$lambda.1se,cv_lassoR$lambda)
train_relaxed_oofProb_vec <- logistic_f(cv_lassoR$fit.preval[['g:0']][,ndx_1se])
train_relaxed_oofClass_vec <- ifelse(
   train_relaxed_oofProb_vec > 0.5, 'HCC', 'Control')

# RECALL $`r cv_lassoR$relaxed$nzero.1se`$ features (vertical
#  cv_blended_statlist <- cv_lassoR$relaxed$statlist[['g:0.5']]
#  cv_blended_1se_error <- cv_blended_statlist$cvm[cv_blended_statlist$lambda==
      #cv_lassoR$relaxed$lambda.1se]

# train - oof
cv_blended_statlist <- cv_lassoR$relaxed$statlist[['g:0.5']]
ndx_1se <- match(cv_lassoR$relaxed$lambda.1se, cv_blended_statlist$lambda)
train_blended_oofProb_vec <- logistic_f(cv_lassoR$fit.preval[['g:0.5']][,ndx_1se])
train_blended_oofClass_vec <- ifelse(
   train_blended_oofProb_vec > 0.5, 'HCC', 'Control')


misclass_id_vec <- unique(c(
 names(train_lasso_oofClass_vec)[train_lasso_oofClass_vec != train_group_vec],
 names(train_enet_oofClass_vec)[train_enet_oofClass_vec != train_group_vec],
```

```
 names(train_relaxed_oofClass_vec)[train_relaxed_oofClass_vec != train_group_vec],
 names(train_blended_oofClass_vec)[train_blended_oofClass_vec != train_group_vec]
 )
)


missclass_oofProb_mtx <- cbind(
 train_lasso_oofProb_vec[misclass_id_vec],
 train_enet_oofProb_vec[misclass_id_vec],
 train_relaxed_oofProb_vec[misclass_id_vec],
 train_blended_oofProb_vec[misclass_id_vec]
)
colnames(missclass_oofProb_mtx) <- c('lasso','enet', 'lassoR', 'blended')

row_med_vec <- apply(missclass_oofProb_mtx, 1, median)
missclass_oofProb_mtx <- missclass_oofProb_mtx[
  order(train_group_vec[rownames(missclass_oofProb_mtx)], row_med_vec),]

plot(
 x=c(1,nrow(missclass_oofProb_mtx)), xlab='samples',
 y=range(missclass_oofProb_mtx), ylab='out-of-fold predicted probability',
 xaxt='n', type='n')

for(RR in 1:nrow(missclass_oofProb_mtx))
points(
 rep(RR, ncol(missclass_oofProb_mtx)),
 missclass_oofProb_mtx[RR,],
 col=ifelse(train_group_vec[rownames(missclass_oofProb_mtx)[RR]] == 'Control',
  'green', 'red'),
 pch=1:ncol(missclass_oofProb_mtx))

legend('top', ncol=2, legend=colnames(missclass_oofProb_mtx),
 pch=1:4, bty='n')

abline(h=0.5)
```

As we've seen above, predictions from lassoR and the blended mix model are basically dichotomous; 0 or 1. Samples have been order by group, and median P(HCC) within group. For the Controls (green), predicted probabilities less than 0.5 are considered correct here. For the HCC (red) samples, predicted probabilities greater than 0.5 are considered correct here.

Now look at the same plot on the test data set.

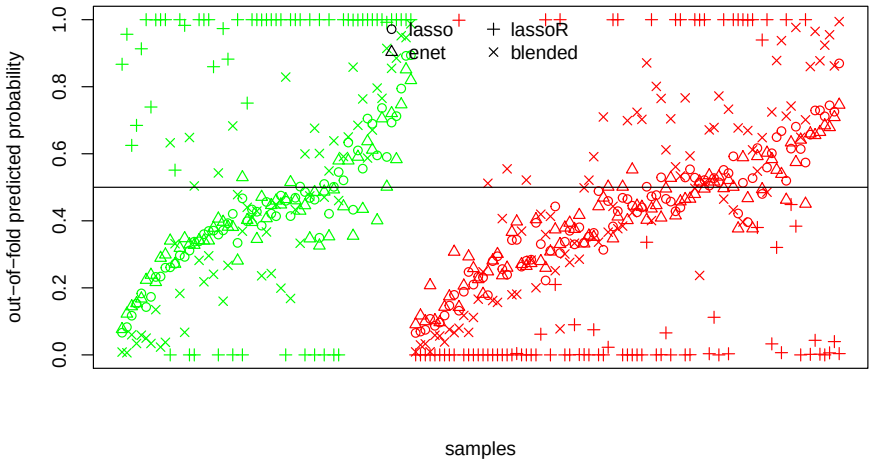

Figure 4.8: out-of-fold predicted probabilities at miscassified samples

```
test_lasso_predClass_vec <- predict(
 cv_lasso,
 newx=test_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

test_enet_predClass_vec <- predict(
 cv_enet,
 newx=test_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

test_relaxed_predClass_vec <- predict(
 cv_lassoR,
 g=0,
 newx=test_lcpm_mtx,
 s='lambda.1se',
 type='class'
)
```

```
test_blended_predClass_vec <- predict(
 cv_lassoR,
 g=0.5,
 newx=test_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

misclass_id_vec <- unique(c(
 names(test_lasso_predClass_vec[,1])[test_lasso_predClass_vec != test_group_vec],
 names(test_enet_predClass_vec[,1])[test_enet_predClass_vec != test_group_vec],
 names(test_relaxed_predClass_vec[,1])[test_relaxed_predClass_vec != test_group_vec],
 names(test_blended_predClass_vec[,1])[test_blended_predClass_vec != test_group_vec]
 )
)



missclass_oofProb_mtx <- cbind(
 test_lasso_predProb_vec[misclass_id_vec,],
 test_enet_predProb_vec[misclass_id_vec,],
 test_relaxed_predProb_vec[misclass_id_vec,],
 test_blended_predProb_vec[misclass_id_vec,]
)
colnames(missclass_oofProb_mtx) <- c('lasso','enet', 'lassoR', 'blended')

row_med_vec <- apply(missclass_oofProb_mtx, 1, median)
missclass_oofProb_mtx <- missclass_oofProb_mtx[
  order(test_group_vec[rownames(missclass_oofProb_mtx)], row_med_vec),]

plot(
 x=c(1,nrow(missclass_oofProb_mtx)), xlab='samples',
 y=range(missclass_oofProb_mtx), ylab='out-of-fold predicted probability',
 xaxt='n', type='n')

for(RR in 1:nrow(missclass_oofProb_mtx))
points(
 rep(RR, ncol(missclass_oofProb_mtx)),
 missclass_oofProb_mtx[RR,],
 col=ifelse(test_group_vec[rownames(missclass_oofProb_mtx)[RR]] == 'Control',
  'green', 'red'),
 pch=1:ncol(missclass_oofProb_mtx))

legend('top', ncol=2, legend=colnames(missclass_oofProb_mtx),
 pch=1:4, bty='n')
```

```
abline(h=0.5)
```

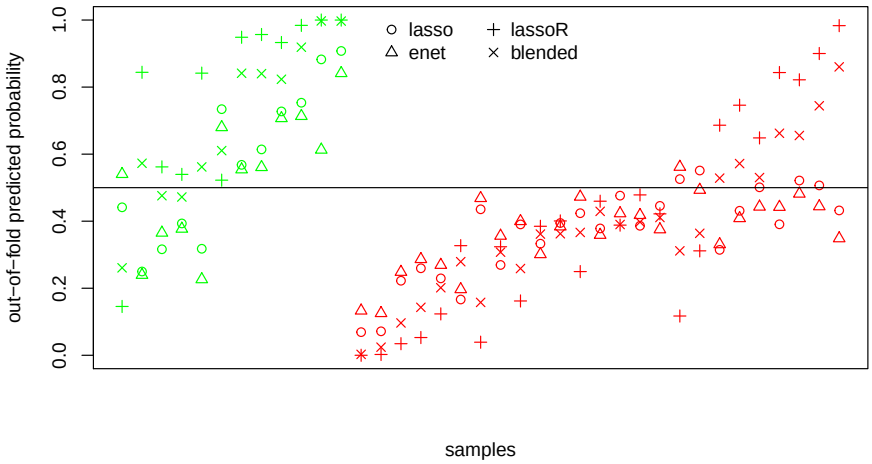

Figure 4.9: Test data predicted probabilities at miscassified samples

The relaxed lasso fit results in essentially dichotomized predicted probability distribution - predicted probabilities are very close to 0 or 1.

We see that for design points in the training set, the predicted probabilies from the relaxed lasso are essentially dichotomized to be tightly distributed at the extremes of the response range. For design points in the test set, the predicted probabilies from the relaxed lasso are comparable to the lasso model predicted porbabilities. This seems to indicate over-fitting in the relaxed lasso fit.

4.6 Compare coefficient profiles

```
# lasso
##########################
# train - cv predicted
lasso_coef <- coef(
 cv_lasso,
 s='lambda.1se'
)
```

```
lasso_coef_frm <- data.frame(
 gene=lasso_coef@Dimnames[[1]][c(1, lasso_coef@i[-1])],
 lasso=lasso_coef@x)


# enet
############################
enet_coef <- coef(
 cv_enet,
 s='lambda.1se'
)
enet_coef_frm <- data.frame(
 gene=enet_coef@Dimnames[[1]][c(1, enet_coef@i[-1])],
 enet=enet_coef@x)

# THESE ARE NOT CORRECT - SKIP
# relaxed lasso (gamma=0)
############################
SKIP <- function() {
lassoR_coef <- coef(
 cv_lassoR,
 s='lambda.1se',
 g=0
)
lassoR_coef_frm <- data.frame(
 gene=lassoR_coef@Dimnames[[1]][c(1, lassoR_coef@i[-1])],
 lassoR=lassoR_coef@x)
}

# blended mix (gamma=0.5)
################################
blended_coef <- coef(
 cv_lassoR,
 s='lambda.1se',
 g=0.5
)
blended_coef_frm <- data.frame(
 gene=blended_coef@Dimnames[[1]][c(1, blended_coef@i[-1])],
 blended=blended_coef@x)


# put it all together
all_coef_frm <-
 base::merge(
 x = lasso_coef_frm,
```

Table 4.7: Zero Ceofficient: rows are lasso, columns enet

	FALSE	TRUE
FALSE	88	12
TRUE	31	0

```
 y = base::merge(
     x = enet_coef_frm,
     y = blended_coef_frm,
         by='gene', all=T),
 by='gene', all=T)

# SKIPPED
#base::merge(
         #x = lassoR_coef_frm,
         #y = blended_coef_frm,
         #by='gene', all=T),

all_coef_frm[,-1][is.na(all_coef_frm[,-1])] <- 0

par(mfrow=c(ncol(all_coef_frm)-1,1), mar=c(0,5,0,1), oma=c(3,1,2,0))

for(CC in 2:ncol(all_coef_frm)) {
 plot(
  x=1:(nrow(all_coef_frm)-1), xlab='',
  y=all_coef_frm[-1, CC], ylab=colnames(all_coef_frm)[CC],
  type='h', xaxt='n')
}
```

Note that there is little difference between the elastic net and the lasso in the selected features, and when the coefficient is zero in one set, it is smaell in the other. By contrast, the blended fit produces more shrinkage.

```
knitr::kable(
with(all_coef_frm[,-1], table(lassoZero=lasso==0, enetZero=enet==0)),
 caption='Zero Ceofficient: rows are lasso, columns enet') %>%
  kableExtra::kable_styling(full_width = F)
```

Coefficients in the blended fit are larger than those in the lasso fit, or zero.

We can also examine these with a scatter plot matrix.

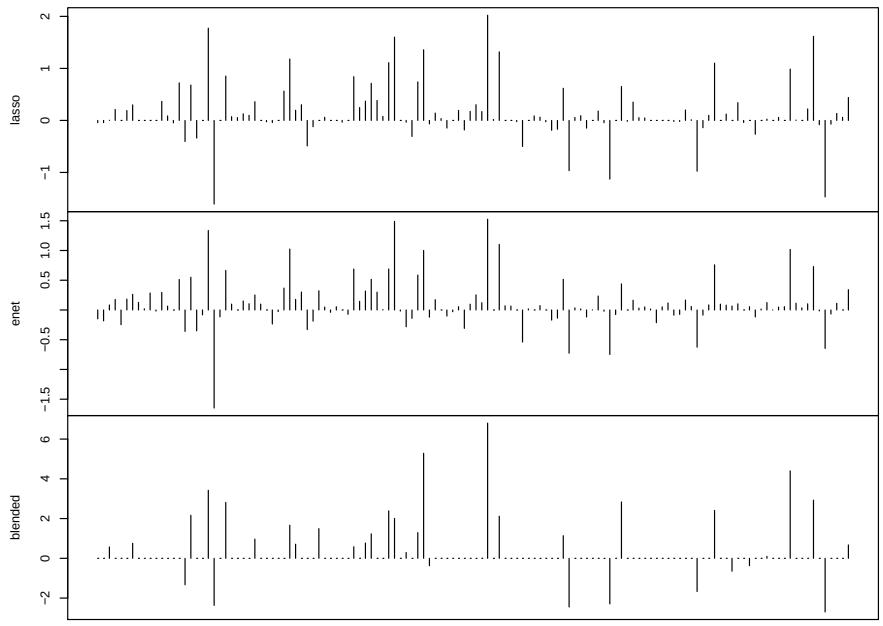

Figure 4.10: Coefficient Profiles

```
pairs(all_coef_frm[-1,-1],
  lower.panel = NULL,
  panel = function(x, y) {
    points(x, y, pch = 16, col = "blue")
  }
)
```

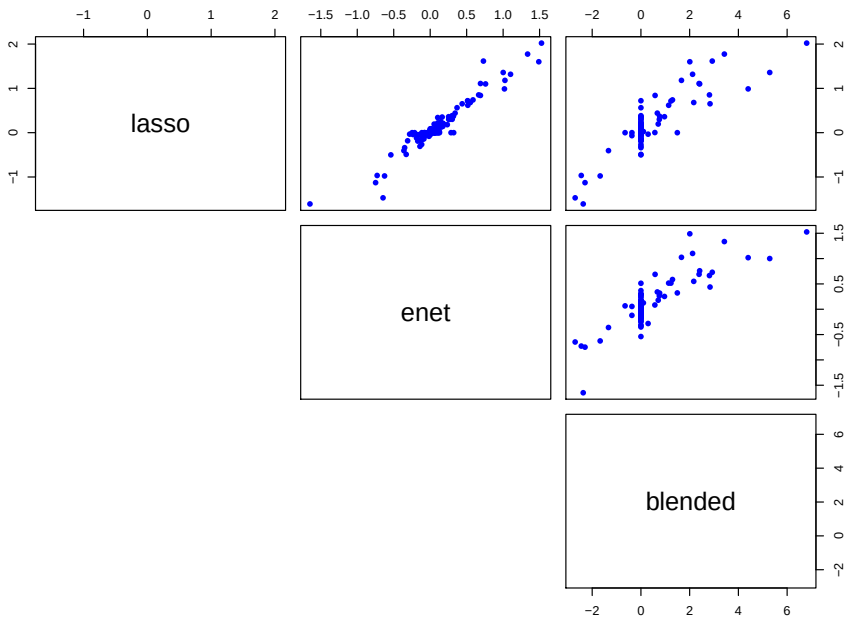

Figure 4.11: Coefficients from fits

4.7 Examine feature selection

Recall from `glmnet` vignette:

```
It is known that the ridge penalty shrinks the coefficients of correlated predictors
towards each other while the lasso tends to pick one of them and discard the others.
The elastic-net penalty mixes these two; if predictors are correlated in groups,
an $\alpha$=0.5 tends to select the groups in or out together.
This is a higher level parameter, and users might pick a value upfront,
else experiment with a few different values. One use of $\alpha$ is for numerical stab
```

```
for example, the *elastic net with $\alpha = 1 - \epsilon$ for some small $\epsilon$>0
performs much like the lasso, but removes any degeneracies and wild behavior caused
by extreme correlations*.
```

To see how this plays out in this dataset, we can look at feature expression heat maps.

Reader notes:

```
Heat maps are rarely useful other than to display the obvious.
Here too heat maps fail to yield any insights, or confirmation
of the relationship between feature correlation and lasso vs enet
feature selection.
```

```
 suppressPackageStartupMessages(require(gplots))

# train - cv predicted
lasso_coef <- coef(
 cv_lasso,
 s='lambda.1se'
)
lasso_coef_frm <- data.frame(
 gene=lasso_coef@Dimnames[[1]][c(1, lasso_coef@i[-1])],
 lasso=lasso_coef@x)


  Mycol <- colorpanel(1000, "blue", "red")
  heatmap.2(
    x=t(train_lcpm_mtx[,lasso_coef_frm$gene[-1]]),
    scale="row",
    labRow=lasso_coef_frm$gene,
    labCol=train_group_vec,
    col=Mycol,
    trace="none", density.info="none",
    #margin=c(8,6), lhei=c(2,10),
    #lwid=c(0.1,4), #lhei=c(0.1,4)
    key=F,
    ColSideColors=ifelse(train_group_vec=='Control', 'green','red'),
    dendrogram="both",
    main=paste('lasso genes - N =', nrow(lasso_coef_frm)-1))
```

```
 suppressPackageStartupMessages(require(gplots))

# train - cv predicted
```

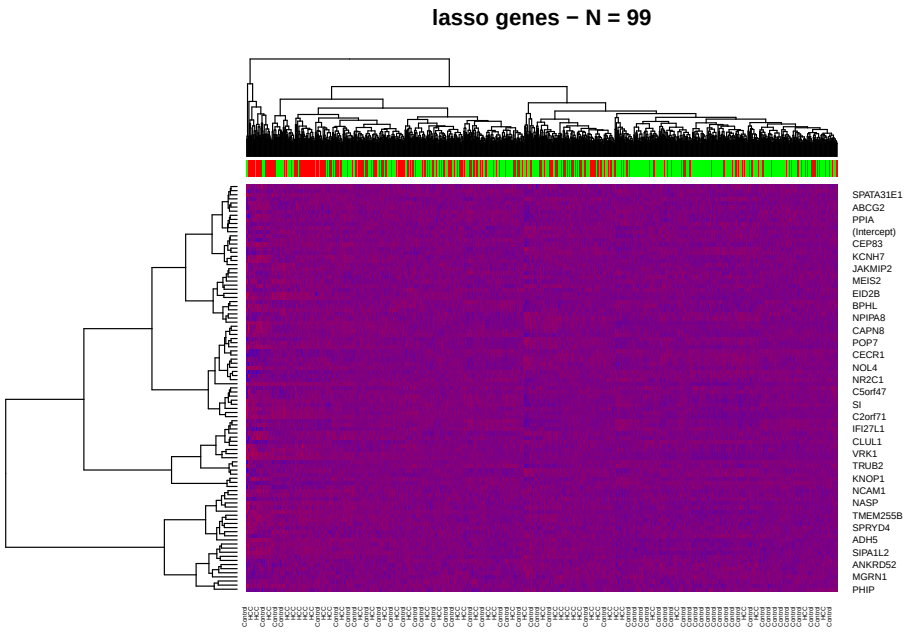

Figure 4.12: Lasso Model Genes

```
enet_coef <- coef(
 cv_enet,
 s='lambda.1se'
)
enet_coef_frm <- data.frame(
 gene=enet_coef@Dimnames[[1]][c(1, enet_coef@i[-1])],
 enet=enet_coef@x)


  Mycol <- colorpanel(1000, "blue", "red")
  heatmap.2(
    x=t(train_lcpm_mtx[,enet_coef_frm$gene[-1]]),
    scale="row",
    labRow=enet_coef_frm$gene,
    labCol=train_group_vec,
    col=Mycol,
    trace="none", density.info="none",
    #margin=c(8,6), lhei=c(2,10),
    #lwid=c(0.1,4), #lhei=c(0.1,4)
    key=F,
    ColSideColors=ifelse(train_group_vec=='Control', 'green','red'),
    dendrogram="both",
    main=paste('enet genes - N =', nrow(enet_coef_frm)-1))
```

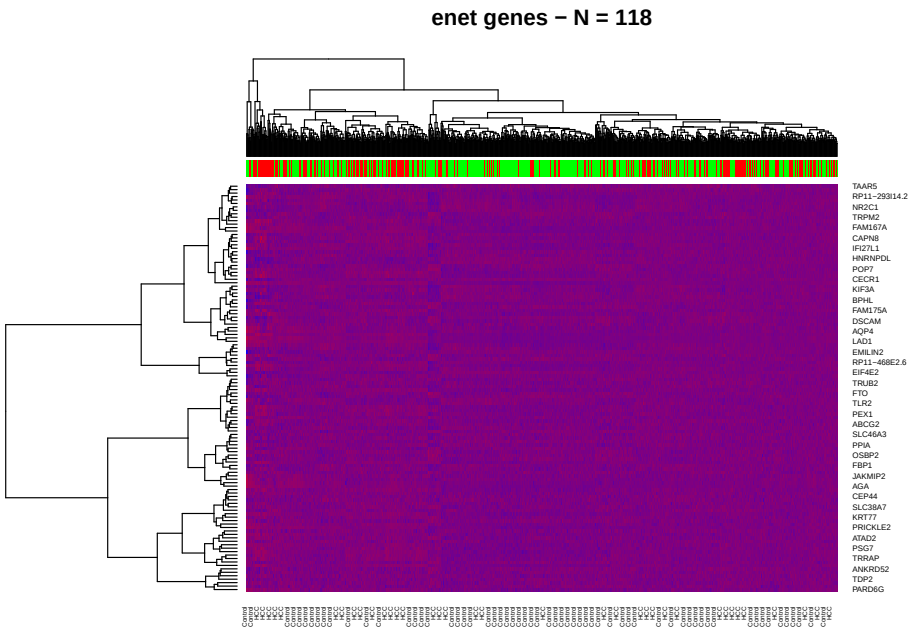

Figure 4.13: Enet Model Genes

Chapter 5

Sample size investigation

We now examine the results of fitting a suite of lasso models to investigate the effect of sample size on various aspects of model performance:

- assessed accuracy: out-of-fold estimates of precision and variability vs train set estimates
- selected feature profile stability - to what extent does the feature set implicitly selected by the lasso vary across random sampling and what is the effect of sample size.

It is hypothesized that below a certain threshold, sample sizes are too small to provide reliable estimates of performance or stable selected feature profiles.

In examining the relationship between sample size and model performance, including variability and reliability of performance indicators derived from fits, we should bear in mind that in addition to sample size, sample composition might also play a factor. If a training data set contains many outliers or otherwise *hard to properly classify* samples, the performance of models fitted to such a data set is expected to be negatively impacted.

In the simulations that we run below we will track sample quality to examine its impact on fitted model performance. Predicted probabilities from fitted model can be transformed into sample quality scores: $Q_i = p_i^{y_i}(1-p_i)^{1-y_i}$, where p_i is the estimated probability of HCC for sample i and y_i is 1 for HCC samples and 0 for Controls. ie. we use the fitted sample contribution to the likelihood function as a sample quality score. To derive the sample quality scores, we will use the predicted response from a lasso model fitted to the entire data set.

Hard to classify samples will have low quality scores. In the results that we discuss below, when we look at variability across repeated random sampling of different sizes, we can use sample quality scores to investigate how much of the

variability is due to sample selection. Note that quality here is not used to say anything about the sample data quality. Low quality here only means that a sample is different from the core of the data set in a way that makes it hard to properly classify. That could happen if the sample were mislabeled, in which case we could think of this sample as being poor quality of course.

The variability in sample set *quality* the we get from simple random sampling, as is done in the simulation below, is not expected to adequately reflect sample set quality variation encountered in practice when data for a particular study are accumulated over extensive time horizons and across varied physical settings. In order to accrue study samples at an acceptable rate, it is not uncommon for the study sponsor to work with several clinics, medical centers, or other tissue or blood sample provider. Or the sponsor may sub-contract the sample acquisition task to a third party who may have suppliers distributed across the globe. This is great news for the rate of sample acquisition, but not such great news for the uniformity of quality of samples. In such a context, the variability in sample set quality as the data set grows over time would not be adequately captured by simple random sampling variability; the variability would be more akin to cluster sampling with potential confounding batch effects. The impact that these effects can have on the classification analysis results cannot be understated, especially in the context of a new technology that is exploiting biological processes that are still not fully understood. All this to make the point that the variability the we are studying here has to be regarded as a lower bound on the variability that is to be expected in practice, and that without an external validation data set to verify results, one should be cautious when interpreting empirical findings, especially in the absence of solid biological underpinnings.

5.1 Full data set fit

We begin by fitting a model to the entire data set in order to:

- obtain a baseline clssification performance against which to judge the performance obtained from the fits to smaller sample sets,
- obtain sample quality scores which can be used to explain variability in the performance of model fitted to data sets of a fixed size, and
- produce a *full model* gene signature which can be used to evaluate the stability of selected features in models fitted to data sets of diffferent sizes.

First assemble the data set. This entails simply re-combining the train and test data.

Table 5.1: samples by group

group	Freq
Control	778
HCC	555

```
# combine train and test
all_lcpm_mtx <- rbind(train_lcpm_mtx, test_lcpm_mtx)

# we have to be careful with factors!
# We'll keep as a character and change to factor when needed
all_group_vec <- c(
 as.character(train_group_vec),
 as.character(test_group_vec)
)
# I suspect adding names to vectors breaks one of the tidy commandments,
# but then again I am sure I have already offended the creed beyond salvation
names(all_group_vec) <- c(
 names(train_group_vec),
 names(test_group_vec)
)

knitr::kable(table(group = all_group_vec),
  caption = "samples by group") %>%
   kableExtra::kable_styling(full_width = F)
```

Now fit the losso model through cross-validation. Note that the results of a cv fit are random due to the random allocation of samples to folds. We can reduce this varibility by properly averaging results over repeated cv fits. Here we will obtain sample quality scores by averaging results over 30 cv runs.

```
set.seed(1)

start_time <-  proc.time()

cv_lassoAll_lst <- lapply(1:30, function(REP) {
glmnet::cv.glmnet(
 x = all_lcpm_mtx,
 y = factor(all_group_vec,levels = c('Control', 'HCC')),
 alpha = 1,
 family = 'binomial',
 type.measure  =  "class",
 keep = T,
 nlambda = 100
```

```
)
}
)

message("lassoAll time: ", round((proc.time() - start_time)[3],2),"s")
```

Examine the fits.

```
### CLEAR CACHE
plot(
 log(cv_lassoAll_lst[[1]]$lambda),
 cv_lassoAll_lst[[1]]$cvm,
 lwd=2,
 xlab='log(Lambda)', ylab='CV Misclassification Error', type='l', ylim=c(0, .5)
)

for(JJ in 2:length(cv_lassoAll_lst))
 lines(
  log(cv_lassoAll_lst[[JJ]]$lambda),
  cv_lassoAll_lst[[JJ]]$cvm,
  lwd=2
)
```

These cv curves are remarkably consistent meaning that the determination of the size or sparsity of the model fitted through cross validation to the full data set is fairly precise:

```
library(magrittr)

par(mfrow=c(1,2), mar=c(3,4, 2, 1))

# nzero
nzero_1se_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se])

nzero_min_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min])

boxplot(list(`1se`=nzero_1se_vec, min = nzero_min_vec), ylab="Selected Features")

# error
error_1se_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])
```

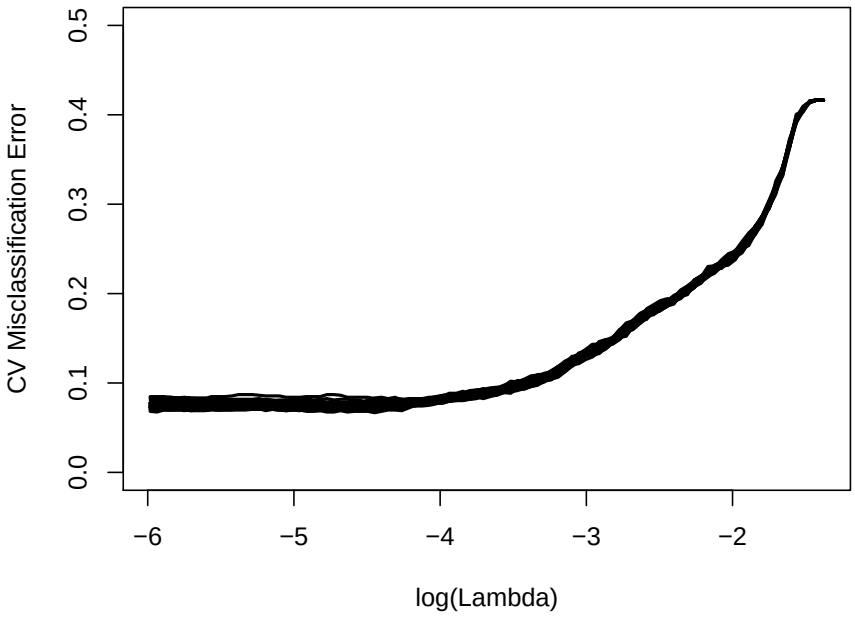

Figure 5.1: Repeated cv lasso models fitted to all samples

```
error_min_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])

boxplot(
 list(`1se`=error_1se_vec, min = error_min_vec),
 ylab=cv_lassoAll_lst[[1]]$name,
 ylim=c(0.06, .10)
)
```

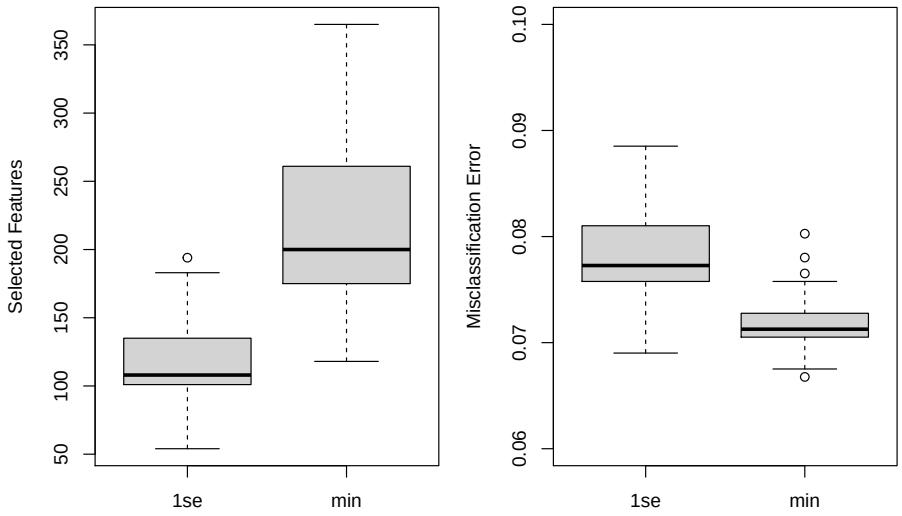

Figure 5.2: Feature selection and estimated error by repeated cv lasso models

```
# tabular format
tmp <- data.frame(rbind(
 `features_1se` = summary(nzero_1se_vec),
 features_min = summary(nzero_min_vec),
 `features:min-1se` = summary(nzero_min_vec - nzero_1se_vec),
 `cv_error_1se` = summary(100*error_1se_vec),
 cv_error_min = summary(100*error_min_vec),
 `cv_error:1se-min` = summary(100*(error_1se_vec-error_min_vec))
))

knitr::kable(tmp %>% dplyr::select(-Mean),
  caption = "Number of selected features",
  digits=1) %>%
```

Table 5.2: Number of selected features

	Min.	X1st.Qu.	Median	X3rd.Qu.	Max.
features_1se	54.0	102.0	108.0	132.5	194.0
features_min	118.0	175.2	200.0	261.0	365.0
features:min-1se	10.0	55.0	86.0	141.5	247.0
cv_error_1se	6.9	7.6	7.7	8.1	8.9
cv_error_min	6.7	7.1	7.1	7.3	8.0
cv_error:1se-min	0.2	0.5	0.6	0.7	1.0

```
   kableExtra::kable_styling(full_width = F)
```

The number of features selected by the minimum lambda models are larger than the number selected by the "one standard error" rule models by a median of 86. The cv error rates obtained from the minimum lambda models are lower then "one standard error" rule models error rates by a median of 0.6%.

The cv error rates observed in this set are comparable to the rates oberved in the lasso models fitted to the training sample set which consisted of 80% of the samples in this set. In other words, there is no obvious gain in performance in moving from a data set with $622 vs 444$ samples to a data set with $778 vs 555$ samples. See Table **??**.

It's not clear at this point whether the minimum lambda model is truly better than the "one standard error" rule model. We would need and external validation set to make this determination. We can compare the two sets of out-of-fold predicted values, averaged across cv replicates, to see if there is a meaningful difference between the two.

```
# predicted probs - 1se
lassoAll_predResp_1se_mtx <- sapply(cv_lassoAll_lst, function(cv_fit) {
  ndx_1se <- match(cv_fit$lambda.1se,cv_fit$lambda)
  logistic_f(cv_fit$fit.preval[,ndx_1se])
 })
lassoAll_predResp_1se_vec <- rowMeans(lassoAll_predResp_1se_mtx)

# predicted probs - min
lassoAll_predResp_min_mtx <- sapply(cv_lassoAll_lst, function(cv_fit) {
  ndx_min <- match(cv_fit$lambda.min,cv_fit$lambda)
  logistic_f(cv_fit$fit.preval[,ndx_min])
 })
lassoAll_predResp_min_vec <- rowMeans(lassoAll_predResp_min_mtx)

# plot
par(mfrow=c(1,2), mar=c(5,5,2,1))
```

```
tmp <- c(
 `1se` = split(lassoAll_predResp_1se_vec, all_group_vec),
 min = split(lassoAll_predResp_min_vec, all_group_vec)
)
names(tmp) <- sub('\\.', '\n', names(tmp))

boxplot(
 tmp,
 ylab='Predicted oof probability',
 border=c('green', 'red'),
 xaxt='n'
)
axis(side=1, at=1:length(tmp), tick=F, names(tmp))


# compare the two
plot(
 x = lassoAll_predResp_1se_vec, xlab='1se model oof Prob',
 y = lassoAll_predResp_min_vec, ylab='min lambda model oof Prob',
 col = ifelse(all_group_vec == 'HCC', 'red', 'green')
)

# Add referecne lines at 10% false positive
thres_1se <- quantile(lassoAll_predResp_1se_vec[all_group_vec == 'Control'], prob=.9)
thres_min <- quantile(lassoAll_predResp_min_vec[all_group_vec == 'Control'], prob=.9)
abline(v = thres_1se, h = thres_min, col='grey')
```

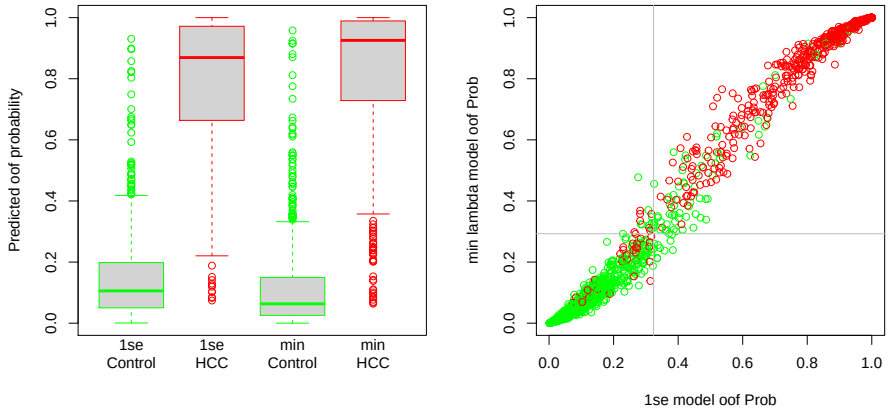

Figure 5.3: Predicted probabilities - averaged over cv replicates

Table 5.3: Classifications: rows are truth

	1se-Control	1se-HCC	min-Control	min-HCC
Control	700	78	700	78
HCC	39	516	32	523

We see that there isn't a big difference in out-of-fold predicted probabilities between the one-standard-error rule and the minimum lamda models. One way to quantify the difference in classification errors is to classify samples according to each vector of predicted probabilities, setting the thresholds to achieve a fixed false positive rate, 10% say. These thresholds are indicated by the grey lines in the scatter plot on the right side of Figure **??**.

```
lassoAll_predClass_1se_vec <- ifelse(
 lassoAll_predResp_1se_vec > thres_1se, 'HCC', 'Control')

lassoAll_predClass_min_vec <- ifelse(
 lassoAll_predResp_min_vec > thres_min, 'HCC', 'Control')

tmp <- cbind(
 table(truth=all_group_vec, `1se-pred`=lassoAll_predClass_1se_vec),
 table(truth=all_group_vec, `min-pred`=lassoAll_predClass_min_vec)
)
# Hack for printing
colnames(tmp) <- c('1se-Control', '1se-HCC', 'min-Control', 'min-HCC')

knitr::kable(tmp,
  caption = "Classifications: rows are truth",
  digits=1) %>%
   kableExtra::kable_styling(full_width = F)
```

When we fix the false positive rate at 10% (ie. the control samples error rates are fixed), the `1se` model makes 39 false negative calls whereas the minimum lambda model makes 32. A difference of 1.3%

Get sample quality scores

To compute quality scores, we will use the out-of-fold predicted probabilities.

```
# get qual scores

y <- as.numeric(all_group_vec == 'HCC')
# 1se
```

```
p <- lassoAll_predResp_1se_vec
sample_1se_qual_vec <- p^y*(1-p)^(1-y)

# min
p <- lassoAll_predResp_min_vec
sample_min_qual_vec <- p^y*(1-p)^(1-y)
```

We can examine quality scores as a function of classification bin.

```
y <- as.numeric(all_group_vec == 'HCC')

# 1se
lassoAll_1se_conf_vec <- paste(
 y,
 as.numeric(lassoAll_predClass_1se_vec=='HCC'),
 sep = ':'
)

# min
lassoAll_min_conf_vec <- paste(
 y,
 as.numeric(lassoAll_predClass_min_vec=='HCC'),
 sep = ':'
)


tmp <- c(
 split(sample_1se_qual_vec, lassoAll_1se_conf_vec),
 split(sample_min_qual_vec, lassoAll_min_conf_vec)
)

par(mfrow=c(1,2), mar=c(4,3,3,2), oma=c(2,2,2,0))
gplots::boxplot2(split(sample_1se_qual_vec, lassoAll_1se_conf_vec),
  outline=F, ylab = '',
  border=c('green', 'green', 'red', 'red'),
  ylim=c(0,1))
title('1se Model')

gplots::boxplot2(split(sample_min_qual_vec, lassoAll_min_conf_vec),
  outline=F, ylab = '',
  border=c('green', 'green', 'red', 'red'),
  ylim=c(0,1))
title('min lambda Model')
```

```
mtext(side=1, outer=T, cex=1.5, 'Classification - Truth:Predicted')
mtext(side=2, outer=T, cex=1.5, 'Quality Score')
mtext(side=3, outer=T, cex=1.5, 'Sample Quality vs Classification Outcome')
```

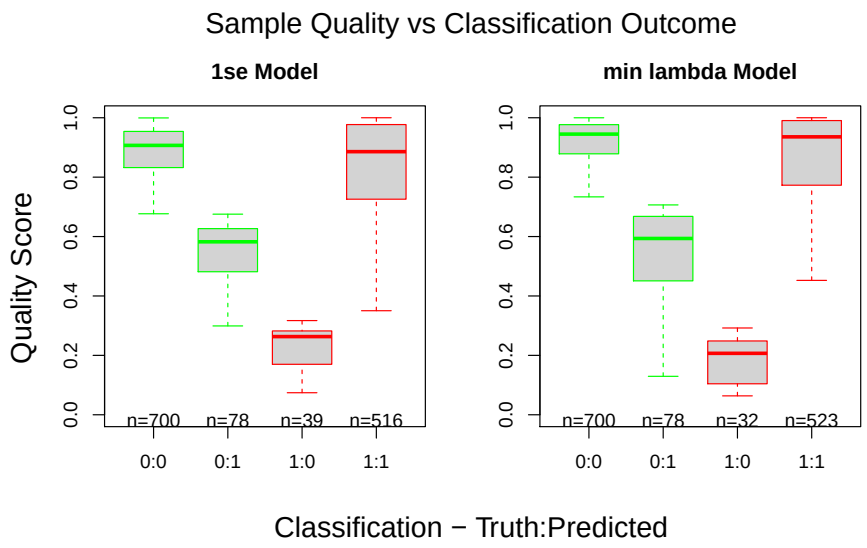

Figure 5.4: quality scores by classification - Control=0, HCC=1

This figure shows that for false positive cases (0:1 or classifying a control as an affected case), the algorithm is *less certain* of its predicted outcome than for the fals negative cases (1:0 or classifying an affected case as a control). ie. the misclassified HCC samples are quite similar to Controls, whereas there is more ambiguity in the misclassified Control samples.

We will use the minimum lambda model to provide the fitted probabilities used to compute quality scores, but we could have used either one.

```
sample_qual_vec <- sample_min_qual_vec
```

5.2 Selected feature list stability

Before moving on to the simulation, let's examine gene selection stability on the full data set. We have two sets of sellected features - one for the one standard deviation rule model, and one for the mimimum lambda model. We saw in Table **??** that the number of features selected by the minimum lambda models

had an IQR of $175.25 - 261$, while the one standard error rule models had an IQR of $102 - 132.5$.

Let's examine the stability of the gene lists across cv replicates.

```
### CLEAR CACHE


# 1se
lassoAll_coef_1se_lst <- lapply(cv_lassoAll_lst, function(cv_fit){
 cv_fit_coef <- coef(
 cv_fit,
 s = "lambda.1se"
 )
 cv_fit_coef@Dimnames[[1]][cv_fit_coef@i[-1]]
 })

# put into matrix
lassoAll_coef_1se_all <- Reduce(union, lassoAll_coef_1se_lst)
lassoAll_coef_1se_mtx <- sapply(lassoAll_coef_1se_lst,
  function(LL) is.element(lassoAll_coef_1se_all, LL)
)
rownames(lassoAll_coef_1se_mtx) <- lassoAll_coef_1se_all

genes_by_rep_1se_tbl <- table(rowSums(lassoAll_coef_1se_mtx))
barplot(
 genes_by_rep_1se_tbl,
 xlab='Number of Replicates',
 ylab='Number of features'

)
```

We see that 44 features are included in every cv replicate. These make up between 33% and 43% (Q1 and Q3) of the cv replicate one standard error rule models feature lists.

```
### CLEAR CACHE


# min
lassoAll_coef_min_lst <- lapply(cv_lassoAll_lst, function(cv_fit){
 cv_fit_coef <- coef(
 cv_fit,
 s = "lambda.min"
 )
 cv_fit_coef@Dimnames[[1]][cv_fit_coef@i[-1]]
```

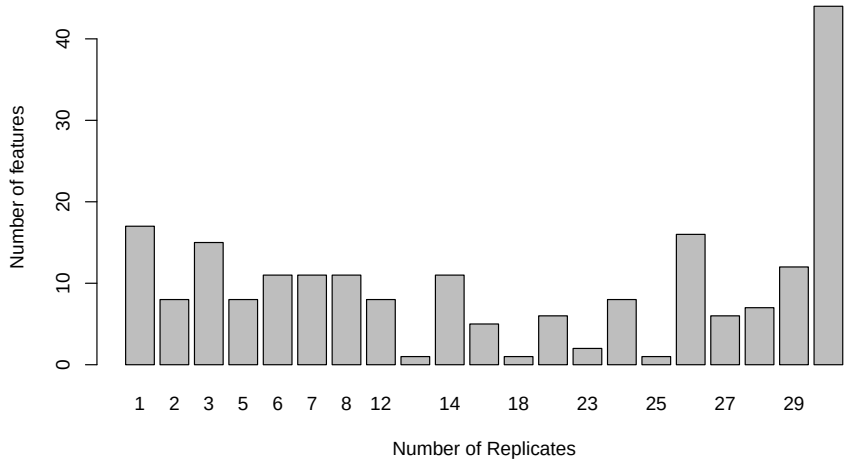

Figure 5.5: Feature list stability for one standard error rule models

```
 })

# put into matrix
lassoAll_coef_min_all <- Reduce(union, lassoAll_coef_min_lst)
lassoAll_coef_min_mtx <- sapply(lassoAll_coef_min_lst,
  function(LL) is.element(lassoAll_coef_min_all, LL)
)
rownames(lassoAll_coef_min_mtx) <- lassoAll_coef_min_all

genes_by_rep_min_tbl <- table(rowSums(lassoAll_coef_min_mtx))
barplot(
 genes_by_rep_min_tbl,
 xlab='Number of Replicates',
 ylab='Number of features'

)
```

We see that 106 features are included in every cv replicate. These make up between 41% and 60% (Q1 and Q3) of the cv replicate min feature lists. We will consider the genes that are selected in all cv replicates as a gene signature produced by each model.

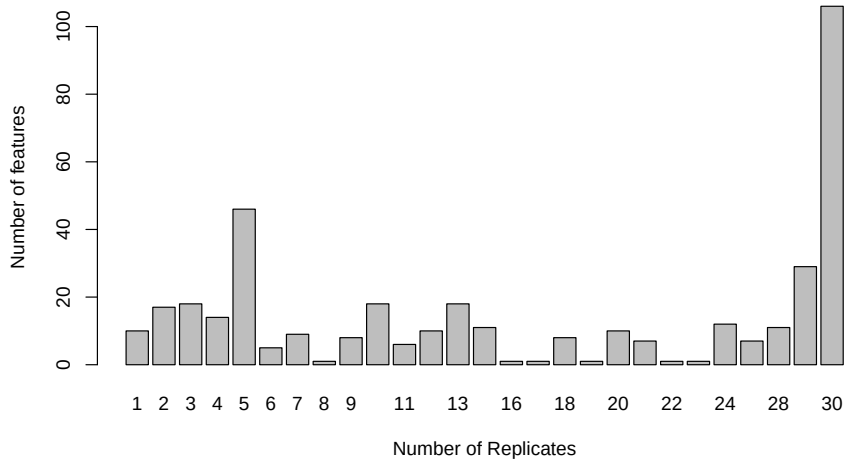

Figure 5.6: Feature list stability for minimum lambda models

```
lasso_gene_sign_1se_vec <- rownames(lassoAll_coef_1se_mtx)[rowSums(lassoAll_coef_1se_m
lasso_gene_sign_min_vec <- rownames(lassoAll_coef_min_mtx)[rowSums(lassoAll_coef_min_m
```

44 out of 44 of the genes in the 1se model gene signature are contained in the min lambda model gene signature.

5.3 Simulation Design

We are now ready to run the simulations.

```
SIM <- 30
SIZE <- c(25, 50, 100, 200, 300)
CV_REP <- 30
```

Simluation parameters:

- Number of simulations : SIM $= 30$
- Sample sizes: SIZE $= 25, 50, 100, 200, 300$
- Number of CV Replicates: CV_REP $= 30$

We will repeat the simulation process SIM = 30 times. For each simulation iteration, we will select 300 Control and 300 HCC samples at random. Models will be fitted and analyzed to balanced subsets of SIZE = $25, 50, 100, 200, 300$, in a `Matryoshka doll` manner to emulate a typical sample accrual process. Note that in this accural process there is no time effect - the accrual process is completely randomized. In practice, there could be significant time effects. For example, the first 25 HCC samples could come from Center A, while the next 25 could come from Center B. And affected and control samples could be acquired from different clinics or in different time interevals. In other words, there is no batch effect or shared variability in our simulation, while these are almost always present in real data, including batch effects that are associated with class labels - controls being in different batches than affected samples is an all too common occurence, for example. One should be especially watchful of potential batch effects when dealing with blood samples as blood is notoriously finicky in character [[16]; [17];]. Presented with results that look impressively good based on a small data set, one should definitely be skeptical of the promise of future equally good results.

For a given simulation and a given sample size, we will obtain CV_REP = 30 cross-validated lasso fits. From these fits, we can obtain 30 out-of-fold assessments of classification accuracy to get a sense if its variability. From each cv replicate, we also obtain an estimated model size and a set of selected features. We will want to examine how these stabilize as the sample size increases.

Note that we limit the simulations to a maximum of sample size of 300 in order to to have simulations with low overlap. With 300 randomly selected HCC samples, the expected overlap between two randomly selected sets of HCC samples is 29.2%. For Controls the expected overlap is 14.9%.

5.4 Setup simulation

To setup the simulation, we only need two master tables: one for the selection of Controls and one for the selection of HCC samples.

```
all_control_vec <- names(all_group_vec[all_group_vec=='Control'])
all_affected_vec <- names(all_group_vec[all_group_vec=='HCC'])
```

We have 778 control sample IDs stored in `all_control_vec` and 555 affected sample IDs stored in `all_affected_vec`. To create a suite of random samples from these, we only need to randomly select indices from each vector.

```
set.seed(12379)

sim_control_mtx <- sapply(
```

```
 1:SIM,
 function(dummy)
   sample(1:length(all_control_vec), size =  max(SIZE))
)


sim_affected_mtx <- sapply(
 1:SIM,
 function(dummy)
   sample(1:length(all_affected_vec), size =  max(SIZE))
)
```

Each simulation is specified by a given column of the simulation design matrices: `sim_control_mtx` and `sim_affected_mtx`, each with domensions $300, 30$. Within each simulation, we can run the analyses of size $25, 50, 100, 200, 300$ by simply selecting samples specified in the appropriate rows of each design matrix.

We can examine how much variability we have in the quality scores of the selected samples. Here we show results for the smalle sample sizes where variability will be the greatest.

```
### CLEAR CACHE

all_control_qual_vec <- sample_qual_vec[all_control_vec]
sim_control_qual_mtx <- sapply(
  1:ncol(sim_control_mtx),
  function(CC) all_control_qual_vec[sim_control_mtx[,CC]]
 )

all_affected_qual_vec <- sample_qual_vec[all_affected_vec]
sim_affected_qual_mtx <- sapply(
  1:ncol(sim_affected_mtx),
  function(CC) all_affected_qual_vec[sim_affected_mtx[,CC]]
 )

# ONLY LOOK AT SAMPLE SIZE == 50
NN <- 50

# PLOT
par(mfrow=c(2,1), mar = c(2,5,2,1))
# control
boxplot(
  sim_control_qual_mtx[1:NN,],
  outline = T,
  ylab = 'Quality Score',
  xaxt = 'n'
```

```
)
title("Control sample quality across simulations")

# affected
boxplot(
  sim_affected_qual_mtx[1:NN,],
  outline = T,
  ylab = 'Quality Score'
)
title("Affected sample quality across simulations")
```

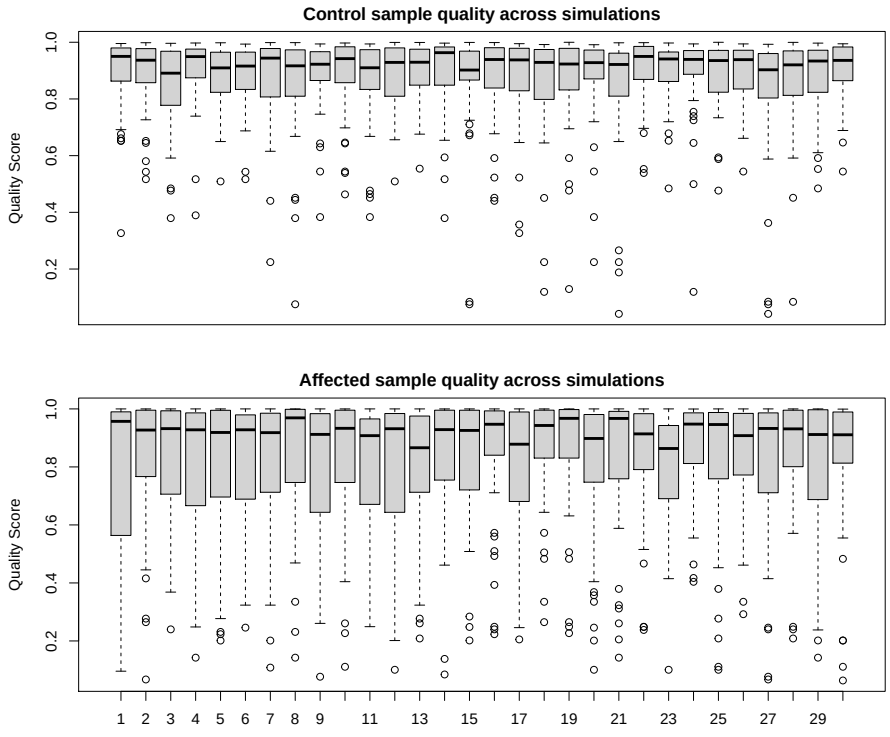

Figure 5.7: sample quality by simulation run for size = 50

In this figure, we are summarizing the quality measures of 50 samples per group acress 30 simulations, or random selections of control and affected samples. We see significant variability in sample quality, especially in the affected cases. This may lead an unwary observer to be overly optimistic, or overly pessimistic, in the early accrual stages of a study.

5.5 Run simulations

As these take a while to run, we will save the results of each similation to a different object and store to disk. These can be easily read from disk when needed for analysis.

The simulation results are saved to the file system and only needs to be run once. The simulation takes ≈ 8 to 10 minutes per iteration, or 4 to 5 hours of run time on a laptop. (Platform: x86_64-apple-darwin17.0 (64-bit) Running under: macOS Mojave 10.14.6)

```
start_time <- proc.time()

# Get stage from SIZE
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0, SIZE), include.lowest = T)

for (SIMno in 1:ncol(sim_control_qual_mtx)) {

  #cat("Running simulation ", SIMno, "\n")

  sim_cv_lst <- lapply(1:length(levels(stage_vec)), function(STGno) {
    Stage_rows_vec <- which(stage_vec %in% levels(stage_vec)[1:STGno])
    #cat("Stage ", STGno, "- analyzing", length(Stage_rows_vec), "paired samples.\n")

    sim_stage_samples_vec <- c(
      all_control_vec[sim_control_mtx[Stage_rows_vec, SIMno]],
      all_affected_vec[sim_affected_mtx[Stage_rows_vec, SIMno]]
    )
    sim_stage_lcpm_mtx <- all_lcpm_mtx[sim_stage_samples_vec, ]
    sim_stage_group_vec <- all_group_vec[sim_stage_samples_vec]
    #print(table(sim_stage_group_vec))

    sim_stage_cv_lst <- lapply(1:CV_REP, function(CV) {
      cv_fit <- glmnet::cv.glmnet(
        x = sim_stage_lcpm_mtx,
        y = sim_stage_group_vec,
        alpha = 1,
        family = "binomial",
        type.measure = "class",
        keep = T,
        nlambda = 30
      )

      # Extract 1se metrics from cv_fit
      ########################
      ndx_1se <- which(cv_fit$lambda == cv_fit$lambda.1se)
```

```
nzero_1se <- cv_fit$nzero[ndx_1se]
cvm_1se <- cv_fit$cvm[ndx_1se]

# test error
sim_stage_test_samples_vec <- setdiff(rownames(all_lcpm_mtx), sim_stage_samples_vec)
sim_stage_test_lcpm_mtx <- all_lcpm_mtx[sim_stage_test_samples_vec,]
sim_stage_test_group_vec <- all_group_vec[sim_stage_test_samples_vec]

test_pred_1se_vec <- predict(
 cv_fit,
 newx=sim_stage_test_lcpm_mtx,
 s="lambda.1se",
 type="class"
)
test_1se_error <- mean(test_pred_1se_vec != sim_stage_test_group_vec)

# genes
coef_1se <- coef(
  cv_fit,
  s = "lambda.1se"
)
genes_1se <- coef_1se@Dimnames[[1]][coef_1se@i[-1]]

# Extract min metrics from cv_fit
########################
ndx_min <- which(cv_fit$lambda == cv_fit$lambda.min)

nzero_min <- cv_fit$nzero[ndx_min]
cvm_min <- cv_fit$cvm[ndx_min]

# test error
sim_stage_test_samples_vec <- setdiff(rownames(all_lcpm_mtx), sim_stage_samples_vec)
sim_stage_test_lcpm_mtx <- all_lcpm_mtx[sim_stage_test_samples_vec,]
sim_stage_test_group_vec <- all_group_vec[sim_stage_test_samples_vec]

test_pred_min_vec <- predict(
 cv_fit,
 newx=sim_stage_test_lcpm_mtx,
 s="lambda.min",
 type="class"
)
test_min_error <- mean(test_pred_min_vec != sim_stage_test_group_vec)

# genes
coef_min <- coef(
```

```
        cv_fit,
        s = "lambda.min"
      )
      genes_min <- coef_min@Dimnames[[1]][coef_min@i[-1]]

      # return cv_fit summary metrics
      list(
       p_1se = nzero_1se,
       p_min = nzero_min,
       cv_1se = cvm_1se,
       cv_min = cvm_min,
       test_1se=test_1se_error,
       test_min=test_min_error,
       genes_1se = genes_1se,
       genes_min = genes_min)
    })
    sim_stage_cv_lst
  })

  # save  sim_cv_lst
  fName <- paste0("sim_", SIMno, "_cv_lst")
  assign(fName, sim_cv_lst)
  save(list = fName, file=file.path("RData", fName))

}
```

5.6 Simulation results

Recall the we have 30 simulations, or randomly selected sets of HCC and Control samples, analyzed in inreasing sizes of $25, 50, 100, 200, 300$, with 30 repeated cross-validated lasso fits:

- Sample sizes: SIZE $= 25, 50, 100, 200, 300$
- Number of CV Replicates: CV_REP $= 30$

First we extract simluation results and store into one big table (only showing the top of table shere):

```
### CLEAR CACHE
sim_files_vec <- list.files('RData', '^sim_')
```

```
# define extraction methods

# Each sumulation is a list of cv results
## nested in a list of replicates
################################################

# cvList2frm_f makes a frame out of the inner list
cvList2frm_f <- function(cv_lst) {
 frm1 <- as.data.frame(t(sapply(cv_lst, function(x) x)))
 frm2 <- data.frame(
  unlist(frm1[[1]]), unlist(frm1[[2]]),
  unlist(frm1[[3]]), unlist(frm1[[4]]),
  unlist(frm1[[5]]), unlist(frm1[[6]]),
  frm1[7], frm1[8])
  names(frm2) <- names(frm1)
  data.frame(Rep=1:nrow(frm2), frm2)}

# cv_lst_to_frm loop over replicates, concatenating the inner list frames
cv_lst_to_frm <- function(sim_cv_lst) {
 do.call('rbind', lapply(1:length(sim_cv_lst),
  function(JJ) {
    siz_frm <- cvList2frm_f(sim_cv_lst[[JJ]])
    data.frame(Size=SIZE[JJ], siz_frm)
  }))
}

# we loop across simulations to combine all results into one big table
lasso_sim_results_frm <- do.call('rbind', lapply(1:length(sim_files_vec),
 function(SIM_NO) {
  load(file=file.path('RData', sim_files_vec[SIM_NO]))
  assign('sim_cv_lst', get(sim_files_vec[SIM_NO]))
  rm(list=sim_files_vec[SIM_NO])

  data.frame(SimNo=paste0('Sim_',formatC(SIM_NO,width = 2,flag = 0)), cv_lst_to_frm(sim_cv_lst))
}
))
```

```
### CLEAR CACHE

knitr::kable(head(with(lasso_sim_results_frm, table(SimNo, Size))),
  caption = paste("Simulation Results - N Sim =", SIM)) %>%
   kableExtra::kable_styling(full_width = F)
```

Table 5.4: Simulation Results - N Sim = 30

	25	50	100	200	300
Sim_01	30	30	30	30	30
Sim_02	30	30	30	30	30
Sim_03	30	30	30	30	30
Sim_04	30	30	30	30	30
Sim_05	30	30	30	30	30
Sim_06	30	30	30	30	30

Table 5.5: Simulation Results - not showing genes column

SimNo	Size	Rep	p_1se	p_min	cv_1se	cv_min	test_1se	test_min
Sim_01	25	1	20	40	0.32	0.26	0.30	0.31
Sim_01	25	2	26	26	0.22	0.22	0.29	0.29
Sim_01	25	3	6	32	0.34	0.30	0.35	0.31
Sim_01	25	4	10	24	0.36	0.30	0.32	0.29
Sim_01	25	5	27	32	0.26	0.22	0.30	0.31
Sim_01	25	6	20	27	0.36	0.32	0.30	0.30

```
knitr::kable(head(lasso_sim_results_frm) %>% dplyr::select(-c(genes_1se, genes_min)),
    caption = paste("Simulation Results - not showing genes column"),
    digits=2) %>%
   kableExtra::kable_styling(full_width = F)
```

5.6.1 Simulation Results - look at one simulation

First examine results for one simulation run. In the figures that follow, each boxplot summarized 30 repreated cross validation runs performed on a fixed random selection of Control and Affeced samples. Recall that as we move from 25 to 50, etc., the sample sets are growing to emulate an accrual of samples over time.

```
### CLEAR CACHE

# get full model cv error ref
error_1se_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])
error_1se_q2 <- quantile(error_1se_vec, prob=1/2)

error_min_vec <- sapply(cv_lassoAll_lst,
```

```
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])
error_min_q2 <- quantile(error_min_vec, prob=1/2)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)


#SIM <- "Sim_01"

for(SIM in unique(lasso_sim_results_frm$SimNo)[1]){

SimNum <- as.numeric(sub('Sim_','',SIM))

simNo_results_frm <- lasso_sim_results_frm %>% dplyr::filter(SimNo==SIM)


# errors
par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
####################
# 1se
####################
cv_1se_lst <- with(simNo_results_frm,
 split(cv_1se, Size))
names(cv_1se_lst) <- paste0(stringr::str_pad(names(cv_1se_lst), width=3, pad='0'),'_cv')

test_1se_lst <- with(simNo_results_frm,
 split(test_1se, Size))
names(test_1se_lst) <- paste0(stringr::str_pad(names(test_1se_lst), width=3, pad='0'),'_cv')

error_1se_lst <- c(cv_1se_lst, test_1se_lst)
error_1se_lst <- error_1se_lst[order(names(error_1se_lst))]

boxplot(error_1se_lst,
  border=c('blue','green'),
  ylim=c(0.05, .4),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_1se_lst))[-1] - 0.5, col='grey')
abline(h= error_1se_q2, col = 'red')
```

```
legend('topright',
   #title='1se errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('one se lambda - error rates'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

# min
####################
cv_min_lst <- with(simNo_results_frm,
 split(cv_min, Size))
names(cv_min_lst) <- paste0(stringr::str_pad(names(cv_min_lst), width=3, pad='0'),'_cv

test_min_lst <- with(simNo_results_frm,
 split(test_min, Size))
names(test_min_lst) <- paste0(stringr::str_pad(names(test_min_lst), width=3, pad='0'),

error_min_lst <- c(cv_min_lst, test_min_lst)
error_min_lst <- error_min_lst[order(names(error_min_lst))]

boxplot(error_min_lst,
  border=c('blue','green'),
  ylim=c(0.05, .4),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
```

```
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_min_lst))[-1] - 0.5, col='grey')
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('min lambda - error rates'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP
mtext(side=3, outer=T, cex=1.25, paste('Sim =',  SIM))

} # for(SIM
```

In this one simulation, we see:

- Model accuracy increases with sample size, with minimal improvement going from N=200 to N=300.
- CV error rates tend to be pessimistic, expecially for the small sample sizes. This is odd and may be related to sample quality.
- There isn't much to chose from between the one standard error and the minimum lambda models. The latter may show lower propensity to produce optimistic cv error rates.

```
### CLEAR CACHE
```

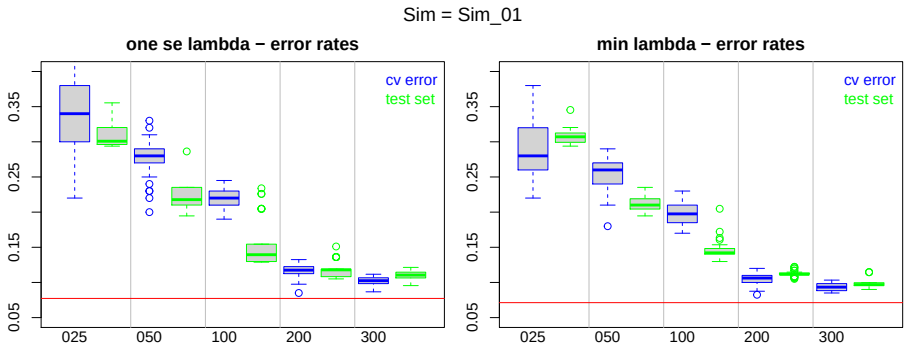

Figure 5.8: lasso Model Errors by Sample Size

```
# get full model nzero ref
nzero_1se_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se])
nzero_1se_q2 <- quantile(nzero_1se_vec, prob=c(2)/4)

nzero_min_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min])
nzero_min_q2 <- quantile(nzero_min_vec, prob=c(2)/4)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)


#SIM <- "Sim_01"

for(SIM in unique(lasso_sim_results_frm$SimNo)[1]){

SimNum <- as.numeric(sub('Sim_','',SIM))

simNo_results_frm <- lasso_sim_results_frm %>% dplyr::filter(SimNo==SIM)


par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
###################
# 1se
####################
```

```
# selected feature counts
p_1se_lst <- with(simNo_results_frm,
 split(p_1se, Size))
names(p_1se_lst) <- paste0(stringr::str_pad(names(p_1se_lst), width=3, pad='0'),'_p')

# get selected features that are part of lasso_gene_sign_1se_vec
# - the signature selected genes
sign_genes_1se_lst <- lapply(1:nrow(simNo_results_frm), function(RR)
    intersect(unlist(simNo_results_frm[RR, 'genes_1se']), lasso_gene_sign_1se_vec))

sign_p_1se_lst <- split(sapply(sign_genes_1se_lst, length), simNo_results_frm$Size)
names(sign_p_1se_lst) <- paste0(stringr::str_pad(names(sign_p_1se_lst), width=3, pad='0'),'_signP


p_singP_1se_lst <- c(p_1se_lst, sign_p_1se_lst)
p_singP_1se_lst <- p_singP_1se_lst[order(names(p_singP_1se_lst))]

boxplot(p_singP_1se_lst,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_1se_lst))[-1] - 0.5, col='grey')
#abline(h= nzero_1se_q2, col = 'red')
legend('topleft',
   #title='1se errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('one se lambda - selected gene counts'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  round(control_qual_vec, 2)
```

```
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

####################
# min
####################
# selected feature counts
p_min_lst <- with(simNo_results_frm,
 split(p_min, Size))
names(p_min_lst) <- paste0(stringr::str_pad(names(p_min_lst), width=3, pad='0'),'_p')

# get selected features that are part of lasso_gene_sign_min_vec
# - the signature selected genes
sign_genes_min_lst <- lapply(1:nrow(simNo_results_frm), function(RR)
    intersect(unlist(simNo_results_frm[RR, 'genes_min']), lasso_gene_sign_min_vec))

sign_p_min_lst <- split(sapply(sign_genes_min_lst, length), simNo_results_frm$Size)
names(sign_p_min_lst) <- paste0(stringr::str_pad(names(sign_p_min_lst), width=3, pad='0


p_singP_min_lst <- c(p_min_lst, sign_p_min_lst)
p_singP_min_lst <- p_singP_min_lst[order(names(p_singP_min_lst))]

boxplot(p_singP_min_lst,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_min_lst))[-1] - 0.5, col='grey')
#abline(h= nzero_min_q2, col = 'red')
legend('topleft',
   #title='min errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
```

```
 )
title(paste('min lambda - selected gene counts'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

mtext(side=3, outer=T, cex=1.25, paste('Sim =',  SIM))

} # for(SIM
```

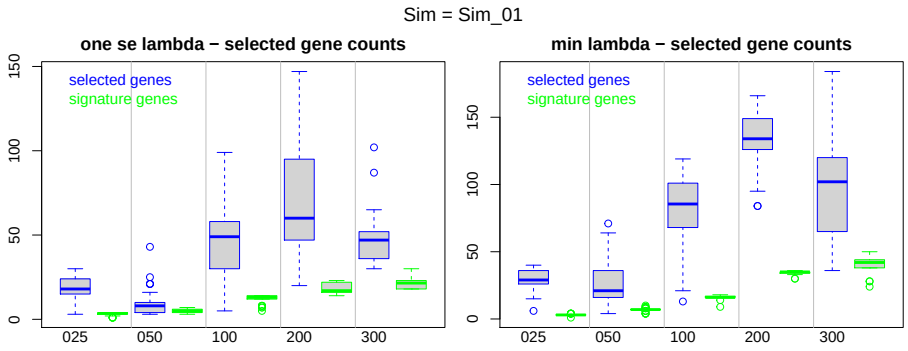

Figure 5.9: lasso Models Selected Features by Sample Size

In this one simulation, we see:

- The selected number of features in the smaller sample size analyses are low, with few feaures belonging to the core signature identified in the full data set.

- As the sample size increases the number of features selected in the minimum lambda model remains variable, but the number of core signature features selected in the samples of sizes 200 and 300 is stable and between 40 and 50.

5.6.2 Summarize results across simulation runs.

Now look acoss all simulations. In the figures that follow, each boxplot summarizes the results of 30 simulations. For a give sample size and a given simulation, each data point is the median across 30 repeated cv runs.

```
### CLEAR CACHE

# get full model cv error ref
error_1se_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])
error_1se_q2 <- quantile(error_1se_vec, prob=1/2)

error_min_vec <- sapply(cv_lassoAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])
error_min_q2 <- quantile(error_min_vec, prob=1/2)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)

par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
# 1se
########################################
## cv
cv_1se_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_cv_1se_lst <- with(sizeVal_results_frm, split(cv_1se, SimNo))
 sapply(sizeVal_cv_1se_lst, median)
})
names(cv_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_cv')

## test
test_1se_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_test_1se_lst <- with(sizeVal_results_frm, split(test_1se, SimNo))
 sapply(sizeVal_test_1se_lst, median)
```

```
})
names(test_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_test')


error_1se_Bysize_lst <- c(cv_1se_Bysize_lst, test_1se_Bysize_lst)
error_1se_Bysize_lst <- error_1se_Bysize_lst[order(names(error_1se_Bysize_lst))]

boxplot(error_1se_Bysize_lst,
  col=0,
  border=c('blue','green'),
  ylim=c(0.05, .5),
  outline=F,
  xaxt='n'
)
for(JJ in 1:length(error_1se_Bysize_lst))
points(
   x=jitter(rep(JJ, length(error_1se_Bysize_lst[[JJ]])), amount=0.25),
   y=error_1se_Bysize_lst[[JJ]],
   col=ifelse(grepl('cv', names(error_1se_Bysize_lst)[JJ]),'blue', 'green')
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_1se_Bysize_lst))[-1] - 0.5, col='grey')
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('one se lambda - error rates'))


# min
########################################
## cv
cv_min_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_cv_min_lst <- with(sizeVal_results_frm, split(cv_min, SimNo))
 sapply(sizeVal_cv_min_lst, median)
```

```
})
names(cv_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_cv')

## test
test_min_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_test_min_lst <- with(sizeVal_results_frm, split(test_min, SimNo))
 sapply(sizeVal_test_min_lst, median)
})
names(test_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_test')


error_min_Bysize_lst <- c(cv_min_Bysize_lst, test_min_Bysize_lst)
error_min_Bysize_lst <- error_min_Bysize_lst[order(names(error_min_Bysize_lst))]

boxplot(error_min_Bysize_lst,
  col=0,
  border=c('blue','green'),
  ylim=c(0.05, .5),
  outline=F,
  xaxt='n'
)
for(JJ in 1:length(error_min_Bysize_lst))
points(
   x=jitter(rep(JJ, length(error_min_Bysize_lst[[JJ]])), amount=0.25),
   y=error_min_Bysize_lst[[JJ]],
   col=ifelse(grepl('cv', names(error_min_Bysize_lst)[JJ]),'blue', 'green')
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_min_Bysize_lst))[-1] - 0.5, col='grey')
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('min lambda - error rates'))
```

```
mtext(side=3, outer=T, cex=1.25, paste('lasso fit error rates summarized across simulations'))
```

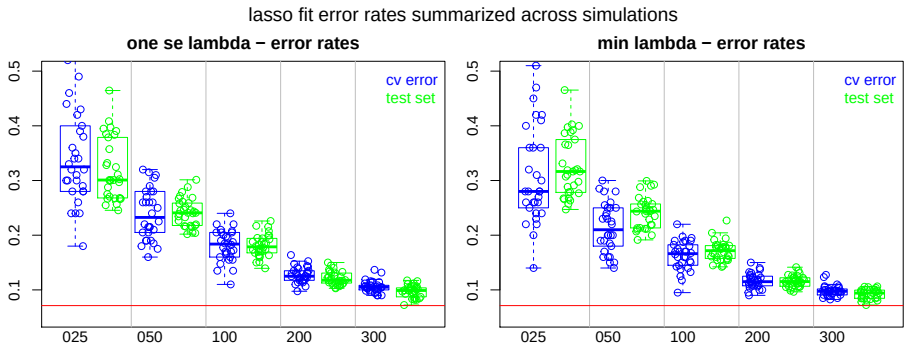

Figure 5.10: lasso Model Errors by Sample Size

```
### CLEAR CACHE

error_1se_Bysize_sum_frm <- t(sapply(error_1se_Bysize_lst, function(LL) quantile(LL, prob=(1:3)/4
colnames(error_1se_Bysize_sum_frm) <- paste0('1se_', colnames(error_1se_Bysize_sum_frm))

error_min_Bysize_sum_frm <- t(sapply(error_min_Bysize_lst, function(LL) quantile(LL, prob=(1:3)/4
colnames(error_min_Bysize_sum_frm) <- paste0('min_', colnames(error_min_Bysize_sum_frm))


knitr::kable(cbind(`1se`=error_1se_Bysize_sum_frm, min=error_min_Bysize_sum_frm),
      caption = paste("lasso error rates by sample size across simulations"),
    digits=2) %>%
   kableExtra::kable_styling(full_width = F)
```

- For the smaller samples sizes, cv error rates for the minimum lambda models tend to be optimistic.
- For lower sample sizes, assess performance is quite variable.

Now look at feature selection.

Table 5.6: lasso error rates by sample size across simulations

	1se_25%	1se_50%	1se_75%	min_25%	min_50%	min_75%
025_cv	0.28	0.32	0.40	0.25	0.28	0.36
025_test	0.27	0.30	0.37	0.28	0.32	0.37
050_cv	0.21	0.23	0.28	0.18	0.21	0.25
050_test	0.22	0.24	0.26	0.21	0.24	0.26
100_cv	0.16	0.18	0.20	0.15	0.17	0.18
100_test	0.17	0.18	0.19	0.16	0.17	0.18
200_cv	0.12	0.12	0.14	0.11	0.12	0.12
200_test	0.11	0.12	0.13	0.11	0.12	0.12
300_cv	0.10	0.10	0.11	0.09	0.10	0.10
300_test	0.09	0.10	0.10	0.08	0.09	0.10

```
### CLEAR CACHE

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)

par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
# 1se
########################################
# selected features
p_1se_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_p_1se_lst <- with(sizeVal_results_frm, split(p_1se, SimNo))
 sapply(sizeVal_p_1se_lst, median)
})
names(p_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_p')

# selected signatue features
sign_p_1se_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)


 sizeVal_sign_genes_1se_lst <- lapply(1:nrow(sizeVal_results_frm), function(RR)
    intersect(unlist(sizeVal_results_frm[RR, 'genes_1se']), lasso_gene_sign_1se_vec))

 sizeVal_sign_p_1se_lst <- split(sapply(sizeVal_sign_genes_1se_lst, length),
    sizeVal_results_frm$SimNo)
```

```
 sapply(sizeVal_sign_p_1se_lst, median)
})
names(sign_p_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_signP')


p_singP_1se_Bysize_lst <- c(p_1se_Bysize_lst, sign_p_1se_Bysize_lst)
p_singP_1se_Bysize_lst <- p_singP_1se_Bysize_lst[order(names(p_singP_1se_Bysize_lst))]

boxplot(p_singP_1se_Bysize_lst,
  col=0,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
for(JJ in 1:length(p_singP_1se_Bysize_lst))
points(
   x=jitter(rep(JJ, length(p_singP_1se_Bysize_lst[[JJ]])), amount=0.25),
   y=p_singP_1se_Bysize_lst[[JJ]],
   col=ifelse(grepl('_p', names(p_singP_1se_Bysize_lst)[JJ]),'blue', 'green')
)

LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_1se_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_1se_Bysize_lst))[-1] - 0.5, col='grey')
#abline(h= nzero_1se_q2, col = 'red')
legend('topleft',
   #title='1se errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('one se lamdba - selected gene counts'))


# min
##########################################
# selected features
p_min_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_p_min_lst <- with(sizeVal_results_frm, split(p_min, SimNo))
```

```
 sapply(sizeVal_p_min_lst, median)
})
names(p_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_p')

# selected signatue features
sign_p_min_Bysize_lst <- lapply(unique(lasso_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- lasso_sim_results_frm %>% dplyr::filter(Size==SizeVal)


 sizeVal_sign_genes_min_lst <- lapply(1:nrow(sizeVal_results_frm), function(RR)
    intersect(unlist(sizeVal_results_frm[RR, 'genes_min']), lasso_gene_sign_min_vec))

 sizeVal_sign_p_min_lst <- split(sapply(sizeVal_sign_genes_min_lst, length),
    sizeVal_results_frm$SimNo)

 sapply(sizeVal_sign_p_min_lst, median)
})
names(sign_p_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(lasso_sim_results_frm$Size), width=3, pad='0'), '_signP')


p_singP_min_Bysize_lst <- c(p_min_Bysize_lst, sign_p_min_Bysize_lst)
p_singP_min_Bysize_lst <- p_singP_min_Bysize_lst[order(names(p_singP_min_Bysize_lst))]

boxplot(p_singP_min_Bysize_lst,
  col=0,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
for(JJ in 1:length(p_singP_min_Bysize_lst))
points(
   x=jitter(rep(JJ, length(p_singP_min_Bysize_lst[[JJ]])), amount=0.25),
   y=p_singP_min_Bysize_lst[[JJ]],
   col=ifelse(grepl('_p', names(p_singP_min_Bysize_lst)[JJ]),'blue', 'green')
)

LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_min_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_min_Bysize_lst))[-1] - 0.5, col='grey
```

```
#abline(h= nzero_min_q2, col = 'red')
legend('topleft',
   #title='min errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('min lambda - selected gene counts'))

mtext(side=3, outer=T, cex=1.25, paste('lasso fit feature selection summarized across simulations
```

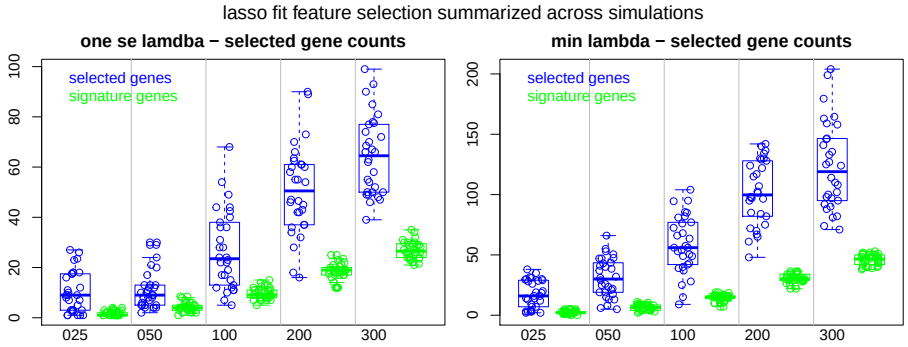

Figure 5.11: lasso Models Selected Features by Sample Size

```
### CLEAR CACHE

p_sing_1se_Bysize_sum_frm <- t(sapply(p_singP_1se_Bysize_lst, function(LL) quantile(LL, prob=(1:3
colnames(p_sing_1se_Bysize_sum_frm) <- paste0('1se_', colnames(p_sing_1se_Bysize_sum_frm))

p_sing_min_Bysize_sum_frm <- t(sapply(p_singP_min_Bysize_lst, function(LL) quantile(LL, prob=(1:3
colnames(p_sing_min_Bysize_sum_frm) <- paste0('min_', colnames(p_sing_min_Bysize_sum_frm))

knitr::kable(cbind(p_sing_1se_Bysize_sum_frm, p_sing_min_Bysize_sum_frm),
    caption = paste("lasso feature selection by sample size across simulations"),
    digits=2) %>%
   kableExtra::kable_styling(full_width = F)
```

- The number of selected features incerase with sample size.

Table 5.7: lasso feature selection by sample size across simulations

	1se_25%	1se_50%	1se_75%	min_25%	min_50%	min_75%
025_p	3.00	9.0	17.50	7.00	16.00	28.88
025_signP	1.00	1.0	2.00	2.00	2.00	3.00
050_p	5.00	9.0	13.00	19.00	30.00	43.38
050_signP	3.00	4.0	5.00	4.50	6.50	8.38
100_p	13.50	23.5	37.50	42.25	56.00	76.75
100_signP	8.00	9.0	11.00	14.00	15.00	16.00
200_p	38.25	50.5	61.00	82.50	99.75	127.00
200_signP	17.00	19.0	20.00	29.00	30.00	33.75
300_p	50.00	64.5	76.25	95.50	119.00	146.38
300_signP	24.00	26.5	29.38	42.25	46.50	49.75

- The number of selected features is quite variable, even for larger sample sizes.
- The number of core signature features among selected features is stable for larger sample sizes and represenst 30 to 40% of the selected features (for N=200 and 300 respectively).

5.7 Effect of sample quality

To see the effect of sample quality on classification results, we will focus on the performance of the small sample fits where variability is the greatest, and the context where investigators should be most aware of the effect of sample selection over and beyond sample size concerns.

```
control_samp25_qual_vec <- apply(sim_control_qual_mtx[1:25, ], 2, median)
affected_samp25_qual_vec <- apply(sim_affected_qual_mtx[1:25, ], 2, median)

samp25_qual_error_mtx <- cbind(
   `1se_cv` = error_1se_Bysize_lst[['025_cv']],
   `1se_test` = error_1se_Bysize_lst[['025_test']],
   `control_Q` = control_samp25_qual_vec,
   `affected_Q` = affected_samp25_qual_vec)


# Correlation panel
panel.cor <- function(x, y){
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    r <- round(cor(x, y), digits=2)
```

```
    txt <- paste0("R = ", r)
    cex.cor <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex = 1.5) ###cex.cor * r)
}

pairs(samp25_qual_error_mtx,
 lower.panel = panel.cor)
```

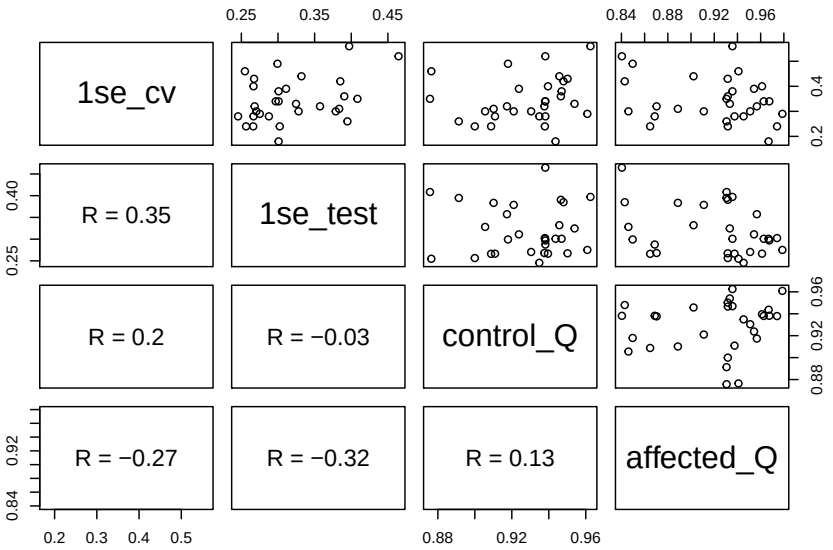

Figure 5.12: sample quality vs classifier performance

We see that although the sample quality effect is not strong, the quality of the affected sample cohorts does have a measurable impact on the 1se cv and test set error rates (cor = -0.32 and -0.27, respectively).

Chapter 6

Conclusions

We have found that ...

Other questions ...

Appendix 1 - Sample size in elastic net fits

Repeat the analyses from Section **??**, but using the elastic net as classfication model.

```
set.seed(1)

start_time <-  proc.time()

cv_enetAll_lst <- lapply(1:30, function(REP) {
glmnet::cv.glmnet(
 x = all_lcpm_mtx,
 y = factor(all_group_vec,levels = c('Control', 'HCC')),
 alpha = 0.5,
 family = 'binomial',
 type.measure  =  "class",
 keep = T,
 nlambda = 100
)
}
)

message("enetAll time: ", round((proc.time() - start_time)[3],2),"s")
```

Examine the fits.

```
### CLEAR CACHE
plot(
 log(cv_enetAll_lst[[1]]$lambda),
 cv_enetAll_lst[[1]]$cvm,
 lwd=2,
 xlab='log(Lambda)', ylab='CV Misclassification Error', type='l', ylim=c(0, .5)
)
```

```
for(JJ in 2:length(cv_enetAll_lst))
 lines(
  log(cv_enetAll_lst[[JJ]]$lambda),
  cv_enetAll_lst[[JJ]]$cvm,
  lwd=2
)
```

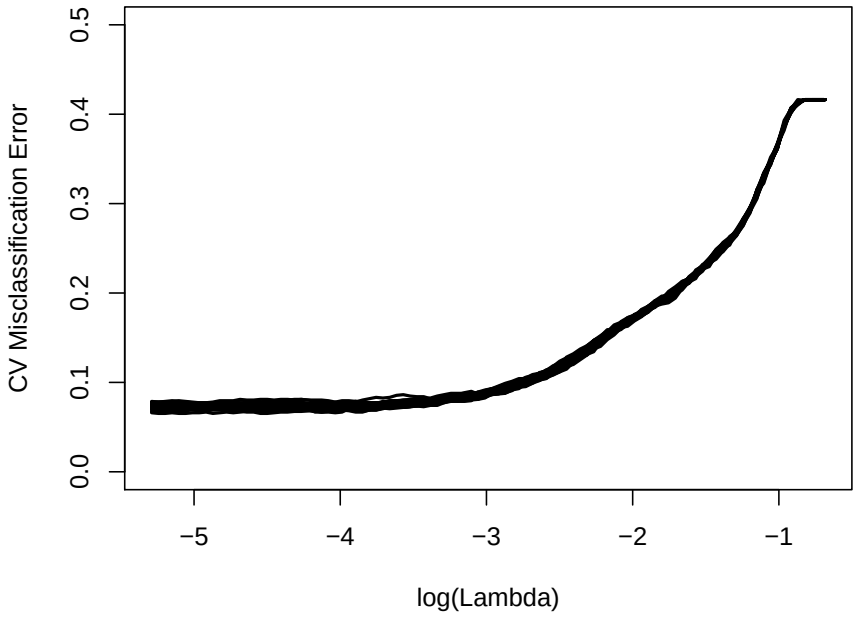

Figure 6.1: Repeated cv enet models fitted to all samples

These cv curves are again remarkably consistent meaning that the determination of the size or sparsity of the model through cross validation is fairly precise:

```
library(magrittr)

par(mfrow=c(1,2), mar=c(3,4, 2, 1))

# nzero
nzero_1se_vec <- sapply(cv_enetAll_lst,
```

```
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se])

nzero_min_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min])

boxplot(list(`1se`=nzero_1se_vec, min = nzero_min_vec), ylab="Full Model cv Summary")

# error
error_1se_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])

error_min_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])

boxplot(
 list(`1se`=error_1se_vec, min = error_min_vec),
 ylab=cv_enetAll_lst[[1]]$name,
 ylim=c(0.06, .10)
)
```

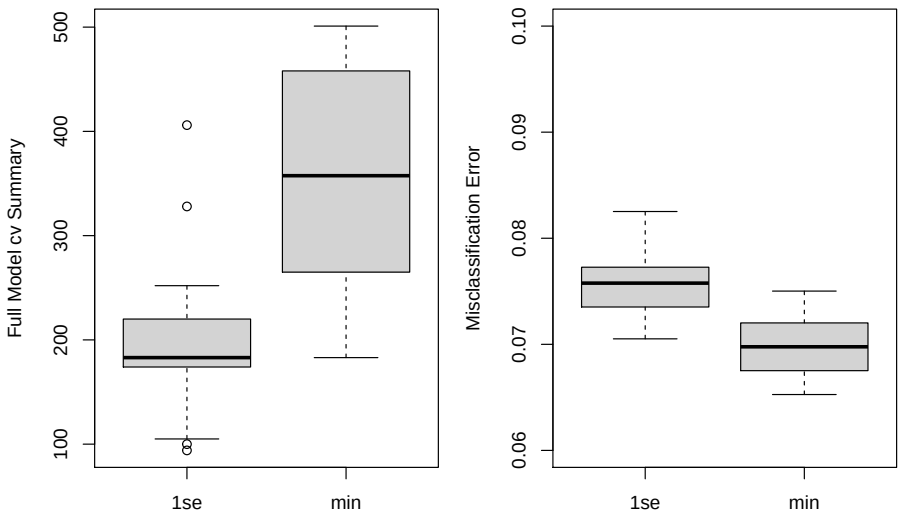

Figure 6.2: Feature selection and estimated error by repeated cv enet models

```
# tabular format
tmp <- data.frame(rbind(
```

Table 6.1: Number of selected features

	Min.	X1st.Qu.	Median	X3rd.Qu.	Max.
features_1se	94.0	174.0	183.0	215.0	406.0
features_min	183.0	271.5	357.5	457.2	501.0
features:min-1se	23.0	81.5	168.5	237.5	331.0
cv_error_1se	7.1	7.4	7.6	7.7	8.3
cv_error_min	6.5	6.8	7.0	7.2	7.5
cv_error:1se-min	0.2	0.5	0.6	0.7	0.9

```
 `features_1se` = summary(nzero_1se_vec),
 features_min = summary(nzero_min_vec),
 `features:min-1se` = summary(nzero_min_vec - nzero_1se_vec),
 `cv_error_1se` = summary(100*error_1se_vec),
 cv_error_min = summary(100*error_min_vec),
 `cv_error:1se-min` = summary(100*(error_1se_vec-error_min_vec))
))

knitr::kable(tmp %>% dplyr::select(-Mean),
  caption = "Number of selected features",
  digits=1) %>%
   kableExtra::kable_styling(full_width = F)
```

The number of features selected by the minimum lambda model is larger than the number selected by the "one standard error" rule by a median of 168.5 and results on a median reduction in cv error rates of 0.6%.

The cv error rates observed in this set are comparable to the rates oberved in the enet models fitted to the training sample set which consisted of 80% of the samples in this set. See Table **??**. It's not clear at this point whether the minimum lambda model is better than the "one standard error" rule model. We would need and external validation set to make this determination. We can compare the two sets of out-of-fold predicted values, averaged across cv replicates, to see if there is a meaningful difference between the two.

```
# predicted probs - 1se
enetAll_predResp_1se_mtx <- sapply(cv_enetAll_lst, function(cv_fit) {
  ndx_1se <- match(cv_fit$lambda.1se,cv_fit$lambda)
  logistic_f(cv_fit$fit.preval[,ndx_1se])
 })
enetAll_predResp_1se_vec <- rowMeans(enetAll_predResp_1se_mtx)

# predicted probs - min
enetAll_predResp_min_mtx <- sapply(cv_enetAll_lst, function(cv_fit) {
```

```
  ndx_min <- match(cv_fit$lambda.min,cv_fit$lambda)
  logistic_f(cv_fit$fit.preval[,ndx_min])
 })
enetAll_predResp_min_vec <- rowMeans(enetAll_predResp_min_mtx)

# plot
par(mfrow=c(1,2), mar=c(5,5,2,1))
tmp <- c(
 `1se` = split(enetAll_predResp_1se_vec, all_group_vec),
 min = split(enetAll_predResp_min_vec, all_group_vec)
)
names(tmp) <- sub('\\.', '\n', names(tmp))

boxplot(
 tmp,
 ylab='Predicted oof probability',
 border=c('green', 'red'),
 xaxt='n'
)
axis(side=1, at=1:length(tmp), tick=F, names(tmp))


# compare the two
plot(
 x = enetAll_predResp_1se_vec, xlab='1se model oof Prob',
 y = enetAll_predResp_min_vec, ylab='min lambda model oof Prob',
 col = ifelse(all_group_vec == 'HCC', 'red', 'green')
)

# Add referecne lines at 10% false positive
thres_1se <- quantile(enetAll_predResp_1se_vec[all_group_vec == 'Control'], prob=.9)
thres_min <- quantile(enetAll_predResp_min_vec[all_group_vec == 'Control'], prob=.9)
abline(v = thres_1se, h = thres_min, col='grey')
```

We see that there isn't a big difference in out-of-fold predicted probabilities between the one-standard-error rule ans minimum lamda models. One way to quantify the difference in classification errors is to classify samples according to each vector of predicted probabilities, setting the thresholds to achieve a fixed false positive rate, 10% say. These thresholds are indicated by the grey lines in the scatter plot on the right side of Figure **??**.

```
enetAll_predClass_1se_vec <- ifelse(
 enetAll_predResp_1se_vec > thres_1se, 'HCC', 'Control')

enetAll_predClass_min_vec <- ifelse(
```

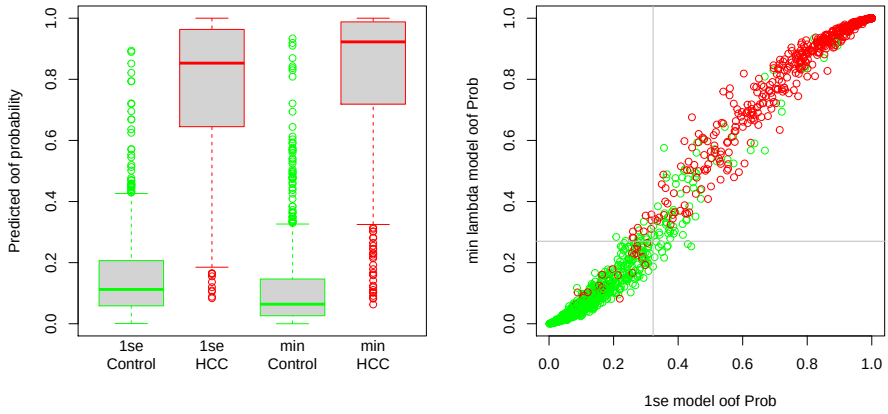

Figure 6.3: Predicted probabilities - averaged over cv replicates

Table 6.2: Classifications: rows are truth

	1se-Control	1se-HCC	min-Control	min-HCC
Control	700	78	700	78
HCC	34	521	25	530

```
 enetAll_predResp_min_vec > thres_min, 'HCC', 'Control')

tmp <- cbind(
 table(truth=all_group_vec, `1se-pred`=enetAll_predClass_1se_vec),
 table(truth=all_group_vec, `min-pred`=enetAll_predClass_min_vec)
)
# Hack for printing
colnames(tmp) <- c('1se-Control', '1se-HCC', 'min-Control', 'min-HCC')

knitr::kable(tmp,
  caption = "Classifications: rows are truth",
  digits=1) %>%
   kableExtra::kable_styling(full_width = F)
```

When we fix the false positive rate at 10%, the `1se` model makes 39 false negative calls whereas the minimum lambda model makes 32. A difference of 1.3%

Selected feature list stability

Before moving on to the simulation, let's examine gene selection stability on the full data set. We have two sets of sellected features - one for the one standard deviation rile model, and one for the mimimum lambda model. We saw in Table **??** that the number of features selected by the minimum lambda models had an IQR of $271.5 - 457.25$, while the one standard error rule models had an IQR of $174 - 215$.

Let's examine the stability of the gene lists across cv replicates.

```
### CLEAR CACHE


# 1se
enetAll_coef_1se_lst <- lapply(cv_enetAll_lst, function(cv_fit){
 cv_fit_coef <- coef(
 cv_fit,
 s = "lambda.1se"
 )
 cv_fit_coef@Dimnames[[1]][cv_fit_coef@i[-1]]
 })

# put into matrix
enetAll_coef_1se_all <- Reduce(union, enetAll_coef_1se_lst)
enetAll_coef_1se_mtx <- sapply(enetAll_coef_1se_lst,
  function(LL) is.element(enetAll_coef_1se_all, LL)
)
rownames(enetAll_coef_1se_mtx) <- enetAll_coef_1se_all

genes_by_rep_1se_tbl <- table(rowSums(enetAll_coef_1se_mtx))
barplot(
 genes_by_rep_1se_tbl,
 xlab='Number of Replicates',
 ylab='Number of features'

)
```

We see that 76 features are included in every cv replicate. These make up between 35% and 44% (Q1 and Q3) of the cv replicate one standard error rule models feature lists.

```
### CLEAR CACHE


# min
```

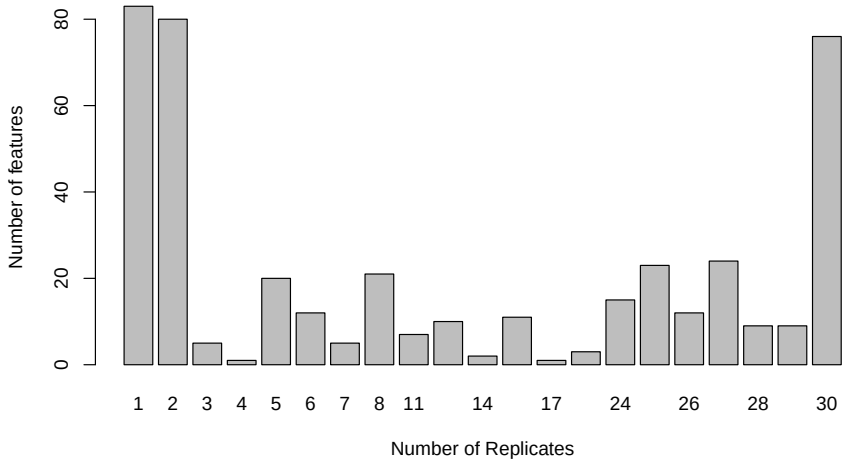

Figure 6.4: Feature list stability for one standard error rule models

```
enetAll_coef_min_lst <- lapply(cv_enetAll_lst, function(cv_fit){
 cv_fit_coef <- coef(
 cv_fit,
 s = "lambda.min"
 )
 cv_fit_coef@Dimnames[[1]][cv_fit_coef@i[-1]]
 })

# put into matrix
enetAll_coef_min_all <- Reduce(union, enetAll_coef_min_lst)
enetAll_coef_min_mtx <- sapply(enetAll_coef_min_lst,
  function(LL) is.element(enetAll_coef_min_all, LL)
)
rownames(enetAll_coef_min_mtx) <- enetAll_coef_min_all

genes_by_rep_min_tbl <- table(rowSums(enetAll_coef_min_mtx))
barplot(
 genes_by_rep_min_tbl,
 xlab='Number of Replicates',
 ylab='Number of features'

)
```

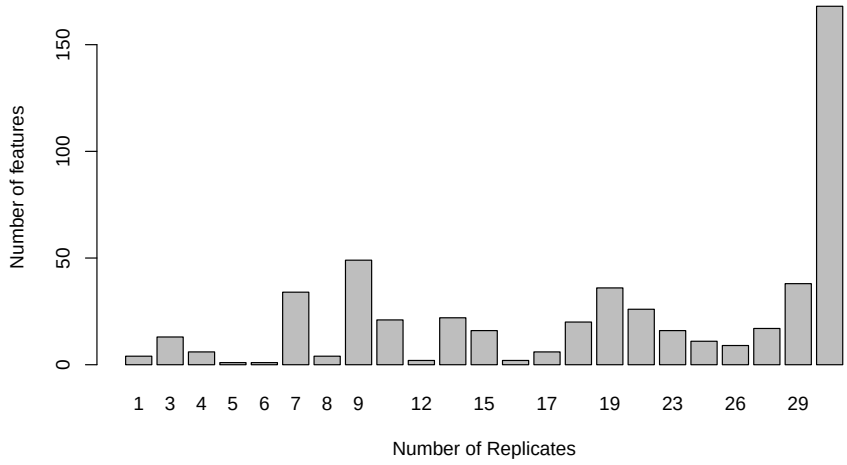

Figure 6.5: Feature list stability for minimum lambda models

We see that 168 features are included in every cv replicate. These make up between 37% and 62% (Q1 and Q3) of the cv replicate min feature lists. We will consider the genes that are selected in all cv replicates as a gene signature produced by each model.

```
enet_gene_sign_1se_vec <- rownames(enetAll_coef_1se_mtx)[rowSums(enetAll_coef_1se_mtx)==30]
enet_gene_sign_min_vec <- rownames(enetAll_coef_min_mtx)[rowSums(enetAll_coef_min_mtx)==30]
```

76 out of 76 of the genes in the 1se model gene signature are contained in the min lambda model gene signature.

Run simulations - enet

As these make take a while to run, we will save the results of each similation to a different object and store to disk. These can be easily read from disk when needed for analysis.

The simulation saves results to the file system and only needs to be run once. The simulation takes ≈ 8 minutes per iteration, or 4 hours of run time on a laptop. (Platform: x86_64-apple-darwin17.0 (64-bit) Running under: macOS Mojave 10.14.6)

```
### CLEAR CACHE
start_time <- proc.time()

# Get stage from SIZE
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0, SIZE), include.lowest = T)

# ran in two runs 1:7, 8:ncol
for (SIMno in 8:ncol(sim_control_qual_mtx)) {

  #cat("Running simulation ", SIMno, "\n")

  sim_cv_lst <- lapply(1:length(levels(stage_vec)), function(STGno) {
    Stage_rows_vec <- which(stage_vec %in% levels(stage_vec)[1:STGno])
    #cat("Stage ", STGno, "- analyzing", length(Stage_rows_vec), "paired samples.\n")

    sim_stage_samples_vec <- c(
      all_control_vec[sim_control_mtx[Stage_rows_vec, SIMno]],
      all_affected_vec[sim_affected_mtx[Stage_rows_vec, SIMno]]
    )
    sim_stage_lcpm_mtx <- all_lcpm_mtx[sim_stage_samples_vec, ]
    sim_stage_group_vec <- all_group_vec[sim_stage_samples_vec]
    #print(table(sim_stage_group_vec))

    sim_stage_cv_lst <- lapply(1:CV_REP, function(CV) {
      cv_fit <- glmnet::cv.glmnet(
        x = sim_stage_lcpm_mtx,
        y = sim_stage_group_vec,
        alpha = 1,
        family = "binomial",
        type.measure = "class",
        keep = T,
        nlambda = 30
      )

      # Extract 1se metrics from cv_fit
      ########################
      ndx_1se <- which(cv_fit$lambda == cv_fit$lambda.1se)

      nzero_1se <- cv_fit$nzero[ndx_1se]
      cvm_1se <- cv_fit$cvm[ndx_1se]

      # test error
      sim_stage_test_samples_vec <- setdiff(rownames(all_lcpm_mtx), sim_stage_samples_
      sim_stage_test_lcpm_mtx <- all_lcpm_mtx[sim_stage_test_samples_vec,]
      sim_stage_test_group_vec <- all_group_vec[sim_stage_test_samples_vec]
```

```
test_pred_1se_vec <- predict(
 cv_fit,
 newx=sim_stage_test_lcpm_mtx,
 s="lambda.1se",
 type="class"
)
test_1se_error <- mean(test_pred_1se_vec != sim_stage_test_group_vec)

# genes
coef_1se <- coef(
  cv_fit,
  s = "lambda.1se"
)
genes_1se <- coef_1se@Dimnames[[1]][coef_1se@i[-1]]

# Extract min metrics from cv_fit
########################
ndx_min <- which(cv_fit$lambda == cv_fit$lambda.min)

nzero_min <- cv_fit$nzero[ndx_min]
cvm_min <- cv_fit$cvm[ndx_min]

# test error
sim_stage_test_samples_vec <- setdiff(rownames(all_lcpm_mtx), sim_stage_samples_vec)
sim_stage_test_lcpm_mtx <- all_lcpm_mtx[sim_stage_test_samples_vec,]
sim_stage_test_group_vec <- all_group_vec[sim_stage_test_samples_vec]

test_pred_min_vec <- predict(
 cv_fit,
 newx=sim_stage_test_lcpm_mtx,
 s="lambda.min",
 type="class"
)
test_min_error <- mean(test_pred_min_vec != sim_stage_test_group_vec)

# genes
coef_min <- coef(
  cv_fit,
  s = "lambda.min"
)
genes_min <- coef_min@Dimnames[[1]][coef_min@i[-1]]

# return cv_fit summary metrics
list(
 p_1se = nzero_1se,
```

```
      p_min = nzero_min,
      cv_1se = cvm_1se,
      cv_min = cvm_min,
      test_1se=test_1se_error,
      test_min=test_min_error,
      genes_1se = genes_1se,
      genes_min = genes_min)
    })
    sim_stage_cv_lst
  })

  # save  sim_cv_lst
  fName <- paste0("enet_sim_", SIMno, "_cv_lst")
  assign(fName, sim_cv_lst)
  save(list = fName, file=file.path("RData", fName))

}
  message("simulation time: ", round((proc.time() - start_time)[3], 2), "s")
```

enet Simulation results

Simulation Results - look at one simulation

First examine results for one simulation run.

```
### CLEAR CACHE

# get full model cv error ref
error_1se_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])
error_1se_q2 <- quantile(error_1se_vec, prob=1/2)

error_min_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])
error_min_q2 <- quantile(error_min_vec, prob=1/2)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)


#SIM <- "Sim_01"
```

```
for(SIM in unique(enet_sim_results_frm$SimNo)[1]){

SimNum <- as.numeric(sub('Sim_','',SIM))

simNo_results_frm <- enet_sim_results_frm %>% dplyr::filter(SimNo==SIM)


# errors
par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
###################
# 1se
####################
cv_1se_lst <- with(simNo_results_frm,
 split(cv_1se, Size))
names(cv_1se_lst) <- paste0(stringr::str_pad(names(cv_1se_lst), width=3, pad='0'),'_cv')

test_1se_lst <- with(simNo_results_frm,
 split(test_1se, Size))
names(test_1se_lst) <- paste0(stringr::str_pad(names(test_1se_lst), width=3, pad='0'),'_cv')

error_1se_lst <- c(cv_1se_lst, test_1se_lst)
error_1se_lst <- error_1se_lst[order(names(error_1se_lst))]

boxplot(error_1se_lst,
  border=c('blue','green'),
  ylim=c(0.05, .4),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_1se_lst))[-1] - 0.5, col='grey')
abline(h= error_1se_q2, col = 'red')
legend('topright',
   #title='1se errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('one se lambda - error rates'))

SKIP  <- function() {
# Add qual annotation
```

```
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

# min
####################
cv_min_lst <- with(simNo_results_frm,
 split(cv_min, Size))
names(cv_min_lst) <- paste0(stringr::str_pad(names(cv_min_lst), width=3, pad='0'),'_cv

test_min_lst <- with(simNo_results_frm,
 split(test_min, Size))
names(test_min_lst) <- paste0(stringr::str_pad(names(test_min_lst), width=3, pad='0'),

error_min_lst <- c(cv_min_lst, test_min_lst)
error_min_lst <- error_min_lst[order(names(error_min_lst))]

boxplot(error_min_lst,
  border=c('blue','green'),
  ylim=c(0.05, .4),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_min_lst))[-1] - 0.5, col='grey')
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
```

```
title(paste('min lambda - error rates'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP
mtext(side=3, outer=T, cex=1.25, paste('Sim =',  SIM))

} # for(SIM
```

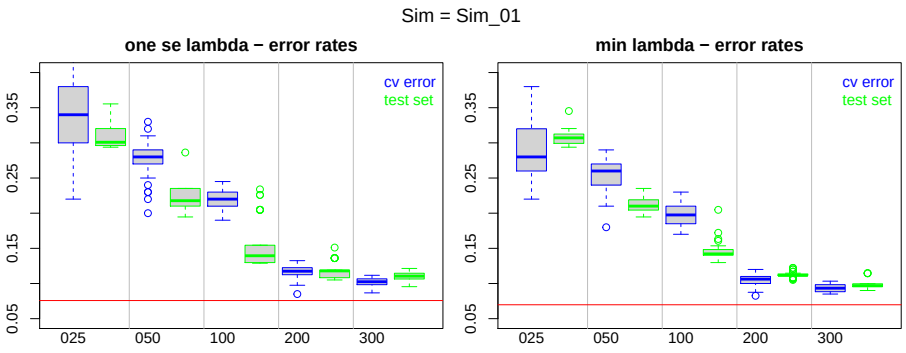

Figure 6.6: enet Model Errors by Sample Size

```
### CLEAR CACHE

# get full model nzero ref
nzero_1se_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.1se])
nzero_1se_q2 <- quantile(nzero_1se_vec, prob=c(2)/4)
```

```
nzero_min_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$nzero[cv_fit$lambda == cv_fit$lambda.min])
nzero_min_q2 <- quantile(nzero_min_vec, prob=c(2)/4)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)


#SIM <- "Sim_01"

for(SIM in unique(enet_sim_results_frm$SimNo)[1]){

SimNum <- as.numeric(sub('Sim_','',SIM))

simNo_results_frm <- enet_sim_results_frm %>% dplyr::filter(SimNo==SIM)


par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
####################
# 1se
####################
# selected feature counts
p_1se_lst <- with(simNo_results_frm,
 split(p_1se, Size))
names(p_1se_lst) <- paste0(stringr::str_pad(names(p_1se_lst), width=3, pad='0'),'_p')

# get selected features that are part of enet_gene_sign_1se_vec
# - the signature selected genes
sign_genes_1se_lst <- lapply(1:nrow(simNo_results_frm), function(RR)
    intersect(unlist(simNo_results_frm[RR, 'genes_1se']), enet_gene_sign_1se_vec))

sign_p_1se_lst <- split(sapply(sign_genes_1se_lst, length), simNo_results_frm$Size)
names(sign_p_1se_lst) <- paste0(stringr::str_pad(names(sign_p_1se_lst), width=3, pad='0


p_singP_1se_lst <- c(p_1se_lst, sign_p_1se_lst)
p_singP_1se_lst <- p_singP_1se_lst[order(names(p_singP_1se_lst))]

boxplot(p_singP_1se_lst,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
LL <- -1
```

```
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_1se_lst))[-1] - 0.5, col='grey')
#abline(h= nzero_1se_q2, col = 'red')
legend('topleft',
   #title='1se errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('one se lamdba - selected gene counts'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_1se_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

####################
# min
####################
# selected feature counts
p_min_lst <- with(simNo_results_frm,
 split(p_min, Size))
names(p_min_lst) <- paste0(stringr::str_pad(names(p_min_lst), width=3, pad='0'),'_p')

# get selected features that are part of enet_gene_sign_min_vec
# - the signature selected genes
sign_genes_min_lst <- lapply(1:nrow(simNo_results_frm), function(RR)
    intersect(unlist(simNo_results_frm[RR, 'genes_min']), enet_gene_sign_min_vec))

sign_p_min_lst <- split(sapply(sign_genes_min_lst, length), simNo_results_frm$Size)
names(sign_p_min_lst) <- paste0(stringr::str_pad(names(sign_p_min_lst), width=3, pad='0'),'_signP
```

```
p_singP_min_lst <- c(p_min_lst, sign_p_min_lst)
p_singP_min_lst <- p_singP_min_lst[order(names(p_singP_min_lst))]

boxplot(p_singP_min_lst,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_min_lst))[-1] - 0.5, col='grey')
#abline(h= nzero_min_q2, col = 'red')
legend('topleft',
   #title='min errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('min lambda - selected gene counts'))

SKIP  <- function() {
# Add qual annotation
control_qual_vec <- sapply(split(sim_control_qual_mtx[,SimNum], stage_vec), median)
affected_qual_vec <- sapply(split(sim_affected_qual_mtx[,SimNum], stage_vec), median)
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  round(control_qual_vec, 2)
 )
LL <- LL + 1
axis(side=1, tick=F, line = LL,
  at =  match(paste0(SIZE0,'_p'),names(p_singP_min_lst)),
  round(affected_qual_vec, 2)
 )
}#SKIP

mtext(side=3, outer=T, cex=1.25, paste('Sim =',  SIM))

} # for(SIM
```

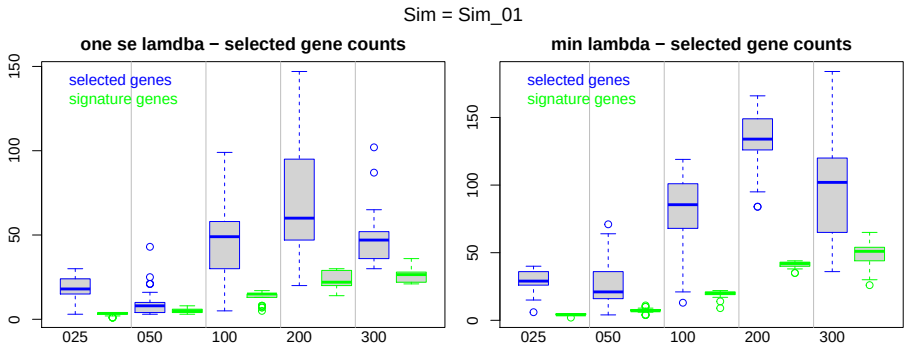

Figure 6.7: enet Models Selected Features by Sample Size

Summarize results across simulation runs

```
### CLEAR CACHE

# get full model cv error ref
error_1se_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se])
error_1se_q2 <- quantile(error_1se_vec, prob=1/2)

error_min_vec <- sapply(cv_enetAll_lst,
 function(cv_fit) cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min])
error_min_q2 <- quantile(error_min_vec, prob=1/2)

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)

par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
# 1se
##########################################
## cv
cv_1se_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_cv_1se_lst <- with(sizeVal_results_frm, split(cv_1se, SimNo))
 sapply(sizeVal_cv_1se_lst, median)
})
```

```
names(cv_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_cv')

## test
test_1se_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_test_1se_lst <- with(sizeVal_results_frm, split(test_1se, SimNo))
 sapply(sizeVal_test_1se_lst, median)
})
names(test_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_test')


error_1se_Bysize_lst <- c(cv_1se_Bysize_lst, test_1se_Bysize_lst)
error_1se_Bysize_lst <- error_1se_Bysize_lst[order(names(error_1se_Bysize_lst))]

boxplot(error_1se_Bysize_lst,
  col=0,
  border=c('blue','green'),
  ylim=c(0.05, .5),
  outline=F,
  xaxt='n'
)
for(JJ in 1:length(error_1se_Bysize_lst))
points(
   x=jitter(rep(JJ, length(error_1se_Bysize_lst[[JJ]])), amount=0.25),
   y=error_1se_Bysize_lst[[JJ]],
   col=ifelse(grepl('cv', names(error_1se_Bysize_lst)[JJ]),'blue', 'green')
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_1se_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_1se_Bysize_lst))[-1] - 0.5, col='grey')
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('one se lambda - error rates'))
```

```
# min
########################################
## cv
cv_min_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_cv_min_lst <- with(sizeVal_results_frm, split(cv_min, SimNo))
 sapply(sizeVal_cv_min_lst, median)
})
names(cv_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_cv')

## test
test_min_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_test_min_lst <- with(sizeVal_results_frm, split(test_min, SimNo))
 sapply(sizeVal_test_min_lst, median)
})
names(test_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_test')


error_min_Bysize_lst <- c(cv_min_Bysize_lst, test_min_Bysize_lst)
error_min_Bysize_lst <- error_min_Bysize_lst[order(names(error_min_Bysize_lst))]

boxplot(error_min_Bysize_lst,
  col=0,
  border=c('blue','green'),
  ylim=c(0.05, .5),
  outline=F,
  xaxt='n'
)
for(JJ in 1:length(error_min_Bysize_lst))
points(
   x=jitter(rep(JJ, length(error_min_Bysize_lst[[JJ]])), amount=0.25),
   y=error_min_Bysize_lst[[JJ]],
   col=ifelse(grepl('cv', names(error_min_Bysize_lst)[JJ]),'blue', 'green')
)
LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_cv'),names(error_min_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_cv'),names(error_min_Bysize_lst))[-1] - 0.5, col='grey')
```

```
abline(h= error_min_q2, col = 'red')
legend('topright',
   #title='min errors', title.col = 'black',
   text.col = c('blue','green'),
   legend = c('cv error', 'test set'),
   bty='n'
 )
title(paste('min lambda - error rates'))


mtext(side=3, outer=T, cex=1.25, paste('enet fit error rates summarized across simulati
```

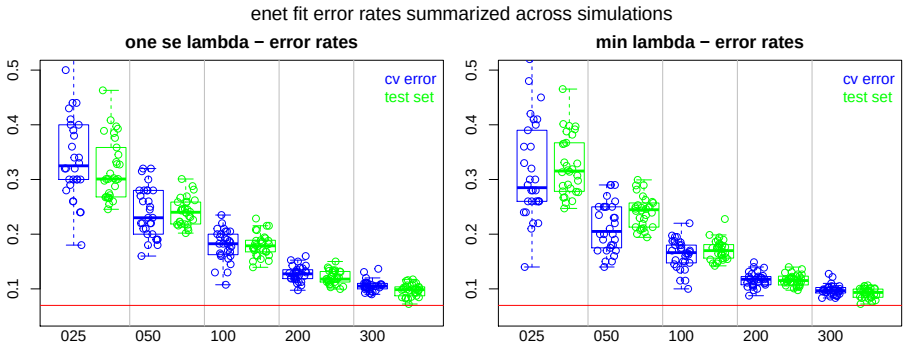

Figure 6.8: enet Model Errors by Sample Size

```
### CLEAR CACHE

error_1se_Bysize_sum_frm <- t(sapply(error_1se_Bysize_lst, function(LL) quantile(LL, pr
colnames(error_1se_Bysize_sum_frm) <- paste0('1se_', colnames(error_1se_Bysize_sum_frm)


error_min_Bysize_sum_frm <- t(sapply(error_min_Bysize_lst, function(LL) quantile(LL, pr
colnames(error_min_Bysize_sum_frm) <- paste0('min_', colnames(error_min_Bysize_sum_frm)


knitr::kable(cbind(`1se`=error_1se_Bysize_sum_frm, min=error_min_Bysize_sum_frm),
      caption = paste("elastic net error rates by sample size across simulations"),
    digits=2) %>%
   kableExtra::kable_styling(full_width = F)
```

Table 6.3: elastic net error rates by sample size across simulations

	1se_25%	1se_50%	1se_75%	min_25%	min_50%	min_75%
025_cv	0.30	0.32	0.40	0.26	0.29	0.38
025_test	0.27	0.30	0.36	0.28	0.32	0.37
050_cv	0.20	0.23	0.28	0.18	0.21	0.25
050_test	0.22	0.24	0.26	0.21	0.24	0.26
100_cv	0.16	0.18	0.20	0.15	0.17	0.18
100_test	0.17	0.18	0.19	0.16	0.17	0.18
200_cv	0.12	0.13	0.13	0.11	0.12	0.12
200_test	0.11	0.12	0.13	0.11	0.12	0.12
300_cv	0.10	0.10	0.11	0.09	0.10	0.10
300_test	0.09	0.10	0.10	0.08	0.09	0.10

Now look at feature selection.

```
### CLEAR CACHE

# Utility objects
SIZE0 <- stringr::str_pad(SIZE, width=3, pad='0')
stage_vec <- cut(1:nrow(sim_control_qual_mtx), c(0,SIZE), include.lowest = T)

par(mfrow=c(1,2), mar=c(4, 2, 2, 1), oma=c(0,0,2,0))
# 1se
#########################################
# selected features
p_1se_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_p_1se_lst <- with(sizeVal_results_frm, split(p_1se, SimNo))
 sapply(sizeVal_p_1se_lst, median)
})
names(p_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_p')

# selected signatue features
sign_p_1se_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)


 sizeVal_sign_genes_1se_lst <- lapply(1:nrow(sizeVal_results_frm), function(RR)
    intersect(unlist(sizeVal_results_frm[RR, 'genes_1se']), enet_gene_sign_1se_vec))
```

```
 sizeVal_sign_p_1se_lst <- split(sapply(sizeVal_sign_genes_1se_lst, length),
    sizeVal_results_frm$SimNo)

 sapply(sizeVal_sign_p_1se_lst, median)
})
names(sign_p_1se_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_signP')


p_singP_1se_Bysize_lst <- c(p_1se_Bysize_lst, sign_p_1se_Bysize_lst)
p_singP_1se_Bysize_lst <- p_singP_1se_Bysize_lst[order(names(p_singP_1se_Bysize_lst))]

boxplot(p_singP_1se_Bysize_lst,
  col=0,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
for(JJ in 1:length(p_singP_1se_Bysize_lst))
points(
   x=jitter(rep(JJ, length(p_singP_1se_Bysize_lst[[JJ]])), amount=0.25),
   y=p_singP_1se_Bysize_lst[[JJ]],
   col=ifelse(grepl('_p', names(p_singP_1se_Bysize_lst)[JJ]),'blue', 'green')
)

LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_1se_Bysize_lst)),
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_1se_Bysize_lst))[-1] - 0.5, col='grey
#abline(h= nzero_1se_q2, col = 'red')
legend('topleft',
   #title='1se errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('one se lamdba - selected gene counts'))


# min
########################################
# selected features
p_min_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
```

```
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)
 sizeVal_p_min_lst <- with(sizeVal_results_frm, split(p_min, SimNo))
 sapply(sizeVal_p_min_lst, median)
})
names(p_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_p')

# selected signatue features
sign_p_min_Bysize_lst <- lapply(unique(enet_sim_results_frm$Size),
function(SizeVal) {
 sizeVal_results_frm <- enet_sim_results_frm %>% dplyr::filter(Size==SizeVal)


 sizeVal_sign_genes_min_lst <- lapply(1:nrow(sizeVal_results_frm), function(RR)
    intersect(unlist(sizeVal_results_frm[RR, 'genes_min']), enet_gene_sign_min_vec))

 sizeVal_sign_p_min_lst <- split(sapply(sizeVal_sign_genes_min_lst, length),
    sizeVal_results_frm$SimNo)

 sapply(sizeVal_sign_p_min_lst, median)
})
names(sign_p_min_Bysize_lst) <- paste0(
 stringr::str_pad(unique(enet_sim_results_frm$Size), width=3, pad='0'), '_signP')


p_singP_min_Bysize_lst <- c(p_min_Bysize_lst, sign_p_min_Bysize_lst)
p_singP_min_Bysize_lst <- p_singP_min_Bysize_lst[order(names(p_singP_min_Bysize_lst))]

boxplot(p_singP_min_Bysize_lst,
  col=0,
  border=c('blue','green'),
  #ylim=c(0, 300),
  xaxt='n'
)
for(JJ in 1:length(p_singP_min_Bysize_lst))
points(
   x=jitter(rep(JJ, length(p_singP_min_Bysize_lst[[JJ]])), amount=0.25),
   y=p_singP_min_Bysize_lst[[JJ]],
   col=ifelse(grepl('_p', names(p_singP_min_Bysize_lst)[JJ]),'blue', 'green')
)

LL <- -1
axis(side=1, tick=F, line = LL,
  at = match(paste0(SIZE0,'_p'),names(p_singP_min_Bysize_lst)),
```

```
  SIZE0
 )
abline(v= match(paste0(SIZE0,'_p'),names(p_singP_min_Bysize_lst))[-1] - 0.5, col='grey
#abline(h= nzero_min_q2, col = 'red')
legend('topleft',
   #title='min errors', title.col = 'black',
   text.col = c('blue', 'green'),
   legend= c('selected genes','signature genes'),
   bty='n'
 )
title(paste('min lambda - selected gene counts'))

mtext(side=3, outer=T, cex=1.25, paste('enet fit feature selection summarized across si
```

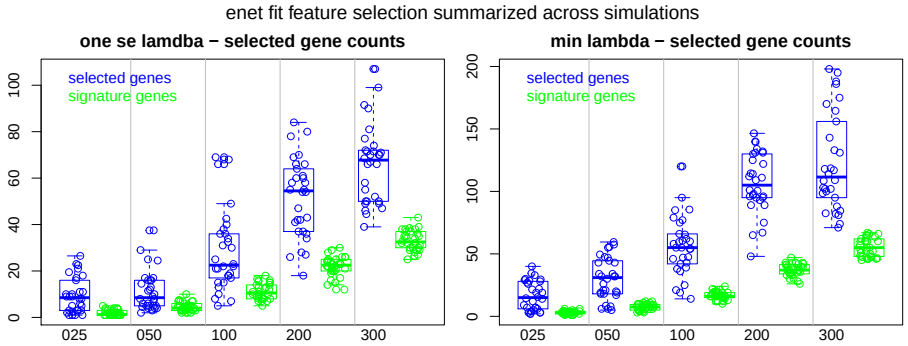

Figure 6.9: enet Models Selected Features by Sample Size

```
### CLEAR CACHE

p_sing_1se_Bysize_sum_frm <- t(sapply(p_singP_1se_Bysize_lst, function(LL) quantile(LL
colnames(p_sing_1se_Bysize_sum_frm) <- paste0('1se_', colnames(p_sing_1se_Bysize_sum_fr

p_sing_min_Bysize_sum_frm <- t(sapply(p_singP_min_Bysize_lst, function(LL) quantile(LL
colnames(p_sing_min_Bysize_sum_frm) <- paste0('min_', colnames(p_sing_min_Bysize_sum_fr

knitr::kable(cbind(p_sing_1se_Bysize_sum_frm, p_sing_min_Bysize_sum_frm),
    caption = paste("elastic net feature selection by sample size across simulations")
    digits=2) %>%
   kableExtra::kable_styling(full_width = F)
```

1. Gai, W., and Sun, K. Epigenetic biomarkers in cell-free dna and applications

Table 6.4: elastic net feature selection by sample size across simulations

	1se_25%	1se_50%	1se_75%	min_25%	min_50%	min_75%
025_p	3.00	8.50	14.75	6.25	15.0	27.75
025_signP	1.00	1.25	2.75	2.00	3.0	3.75
050_p	5.00	8.50	16.00	18.25	31.0	44.12
050_signP	3.00	4.00	5.75	5.00	7.5	9.38
100_p	17.25	22.50	36.00	42.25	55.0	65.88
100_signP	8.12	10.50	14.00	15.25	16.0	19.00
200_p	38.00	54.50	63.25	95.00	105.0	128.75
200_signP	20.00	22.50	24.88	34.00	37.0	41.38
300_p	50.00	67.75	71.88	95.88	111.5	152.75
300_signP	30.00	32.50	36.75	48.12	55.0	61.50

in liquid biopsy. Genes *10*, 32. Available at: https://pubmed.ncbi.nlm.nih.gov/30634483.

2. Cai, J., Chen, L., Zhang, Z., Zhang, X., Lu, X., Liu, W., Shi, G., Ge, Y., Gao, P., and Yang, Y. *et al.* Genome-wide mapping of 5-hydroxymethylcytosines in circulating cell-free dna as a non-invasive approach for early detection of hepatocellular carcinoma. Gut, gutjnl–2019–318882. Available at: http://gut.bmj.com/content/early/2019/07/28/gutjnl-2019-318882.abstract.

3. Li, W., Zhang, X., Lu, X., You, L., Song, Y., Luo, Z., Zhang, J., Nie, J., Zheng, W., and Xu, D. *et al.* DNA 5-hydroxymethylcytosines from cell-free circulating dna as diagnostic biomarkers for human cancers. bioRxiv, 163204. Available at: http://biorxiv.org/content/early/2017/07/13/163204.abstract.

4. Song, C.-X., Yin, S., Ma, L., Wheeler, A., Chen, Y., Zhang, Y., Liu, B., Xiong, J., Zhang, W., and Hu, J. *et al.* (2017). 5-hydroxymethylcytosine signatures in cell-free dna provide information about tumor types and stages. Cell Research *27*, 1231–1242. Available at: https://doi.org/10.1038/cr.2017.106.

5. Collin, F., Ning, Y., Phillips, T., McCarthy, E., Scott, A., Ellison, C., Ku, C.-J., Guler, G.D., Chau, K., and Ashworth, A. *et al.* Detection of early stage pancreatic cancer using 5-hydroxymethylcytosine signatures in circulating cell free dna. bioRxiv, 422675. Available at: http://biorxiv.org/content/early/2018/09/26/422675.abstract.

6. Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. J Stat Softw *33*, 1–22.

7. Hastie, T., and Tibshirani, R. (2017). Extended comparisons of best subset selection, forward stepwise selection, and the lasso. arXiv: Methodology.

8. Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., and Tibshirani, R.J. Strong rules for discarding predictors in lasso-type problems.

Journal of the Royal Statistical Society. Series B, Statistical methodology *74*, 245–266. Available at: https://pubmed.ncbi.nlm.nih.gov/25506256.

9. Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for cox's proportional hazards model via coordinate descent. J Stat Softw *39*, 1–13.

10. Simon, N., Friedman, J., and Hastie, T. (2013). A blockwise descent algorithm for group-penalized multiresponse and multinomial regression. arXiv preprint arXiv:1311.6529.

11. Xiang, G., Keller, C.A., Giardine, B., An, L., Li, Q., Zhang, Y., and Hardison, R.C. (2020). S3norm: Simultaneous normalization of sequencing depth and signal-to-noise ratio in epigenomic data. Nucleic Acids Research *48*, e43–e43. Available at: https://doi.org/10.1093/nar/gkaa105.

12. Lozoya, O.A., Santos, J.H., and Woychik, R.P. A leveraged signal-to-noise ratio (lstnr) method to extract differentially expressed genes and multivariate patterns of expression from noisy and low-replication rnaseq data. Frontiers in genetics *9*, 176–176. Available at: https://pubmed.ncbi.nlm.nih.gov/29868123.

13. Simonsen, A.T., Hansen, M.C., Kjeldsen, E., Møller, P.L., Hindkjær, J.J., Hokland, P., and Aggerholm, A. (2018). Systematic evaluation of signal-to-noise ratio in variant detection from single cell genome multiple displacement amplification and exome sequencing. BMC Genomics *19*, 681. Available at: https://doi.org/10.1186/s12864-018-5063-5.

14. Rapaport, F., Khanin, R., Liang, Y., Pirun, M., Krek, A., Zumbo, P., Mason, C.E., Socci, N.D., and Betel, D. (2013). Comprehensive evaluation of differential gene expression analysis methods for rna-seq data. Genome biology *14*, R95–R95. Available at: https://pubmed.ncbi.nlm.nih.gov/24020486.

15. Bertsimas, D., King, A., and Mazumder, R. (2016). Best subset selection via a modern optimization lens. Ann. Statist. *44*, 813–852. Available at: https://projecteuclid.org:443/euclid.aos/1458245736.

16. Huang, L.-H., Lin, P.-H., Tsai, K.-W., Wang, L.-J., Huang, Y.-H., Kuo, H.-C., and Li, S.-C. The effects of storage temperature and duration of blood samples on dna and rna qualities. PloS one *12*, e0184692–e0184692. Available at: https://pubmed.ncbi.nlm.nih.gov/28926588.

17. Permenter, J., Ishwar, A., Rounsavall, A., Smith, M., Faske, J., Sailey, C.J., and Alfaro, M.P. (2015). Quantitative analysis of genomic dna degradation in whole blood under various storage conditions for molecular diagnostic testing. Molecular and Cellular Probes *29*, 449–453. Available at: http://www.sciencedirect.com/science/article/pii/S0890850815300207.

18. Law, C., Alhamdoosh, M., Su, S., Dong, X., Tian, L., Smyth, G., and Ritchie, M. (2018). RNA-seq analysis is easy as 1-2-3 with limma, glimma and edgeR [version 3; peer review: 3 approved]. F1000Research *5*. Available at: https://dx.doi.org/10.12688%2Ff1000research.9005.3.

19. Ritchie, M.E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., and Smyth, G.K. Limma powers differential expression analyses for rna-sequencing and microarray studies. Nucleic acids research *43*, e47–e47. Available at: https://pubmed.ncbi.nlm.nih.gov/25605792.

20. Peixoto, L., Risso, D., Poplawski, S.G., Wimmer, M.E., Speed, T.P., Wood, M.A., and Abel, T. How data analysis affects power, reproducibility and biological insight of rna-seq studies in complex datasets. Nucleic acids research *43*, 7664–7674. Available at: https://pubmed.ncbi.nlm.nih.gov/26202970.

21. Gandolfo, L.C., and Speed, T.P. RLE plots: Visualizing unwanted variation in high dimensional data. PloS one *13*, e0191629–e0191629. Available at: https://pubmed.ncbi.nlm.nih.gov/29401521.

22. Risso, D., Ngai, J., Speed, T.P., and Dudoit, S. (2014). Normalization of rna-seq data using factor analysis of control genes or samples. Nature Biotechnology *32*, 896–902. Available at: https://doi.org/10.1038/nbt.2931.

23. McCarthy, D.J., and Smyth, G.K. Testing significance relative to a fold-change threshold is a treat. Bioinformatics *25*, 765–771. Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2654802/.

24. Lockhart, R., Taylor, J., Tibshirani, R.J., and Tibshirani, R. A significance test for the lasso. Ann Stat *42*, 413–468.

25. Wasserman, L. (2014). Discussion: "A significance test for the lasso". Ann. Statist. *42*, 501–508. Available at: https://projecteuclid.org:443/euclid.aos/1400592166.

26. Engebretsen, S., and Bohlin, J. Statistical predictions with glmnet. Clinical epigenetics *11*, 123–123. Available at: https://pubmed.ncbi.nlm.nih.gov/31443682.

27. Höfling, H., and Tibshirani, R. (2008). A study of pre-validation. *2*, 643–664. Available at: http://www.jstor.org/stable/30244221.