DNA Hydroxymethylation in Hepatocellular Carcinoma

Francois Collin

2020-09-08

Contents

Preamble

This vignette offers some exploratory data analyses of data available from the NCBI GEO web site.

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 2 we provide some background to our modeling approach.
- In Section 3 we preprocess the data that we will use for the classification analysis and perform some light QC analyses.
- In Section 4 we explore some glmnet fits that discriminate between early stage HCC and control samples.
- In Section 5 we examine the results of fitting a suite of models to investigate the effect of sample size on model performance.
- In Section 6 look at the question of assessing variable importance.
- Concluding remarks are in Section 7.

Chapter 2

Modeling - Background

Refer to first pass study for relevant exploratory data analyis results.

2.1 Predictive modeling for genomic data

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.

2.1.1 caret for model evaluation

The `caret` Package provide a 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 only mention `caret` here because it is an extremely useful tool for anyone interested in comparing many predictive models. We have done that in the past and have found that regularized regression models perform as well as any in the context of classification based on genomic scale data and will focus on the particular set of tools for fitting and analyzing regularized regression models provided by the `glmnet` R package.

2.2 glmnet

In this investigation we will focus on models that can be analyzed with the the `glmnet` R package [6]. Several factors favor this choice:

- 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.*

2.2.1 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 od variance explained is defined as

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

.

This is maximizd 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

Lasso vs Best Subset

- Best subset selection

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

- lasso

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

- Bertsimas et al. (2016) [15]
  - presented a mixed integer optimization (MIO) formulation for the best subset selection problem
  - Using these MIO solvers, can solve problems with p in the hundreds and even thousands
  - demonstrated that best subset selection generally gives superior prediction accuracy compared to forward stepwise selection and the lasso, over a variety of problem setups.
- Hastie et al. (2017) [7]
  - 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 throughout;

  - the relaxed lasso is the overall winner, performing just about as well as the lasso in low SNR scenarios, and as well as best subset selection in high SNR scenarios. We conclude that it 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.

- relaxed lasso

$$\hat{\beta}^{relax}(\lambda, \gamma) = \gamma\beta^{lasso}(\lambda) + (1 - \gamma)(\beta^{LS}(\lambda)$$

- shrunken relaxed lasso (aka the blended fit)

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

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

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

The de-biasing will potentially improve prediction performance, and CV 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

Preprocessing

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. For the Cai et al. [2] model fitting and analysis, samples were separated into `Train` and `Val-1` subsets. `Val-2` was 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))

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`

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

Table 3.2: Samples used in this analysis

group	Freq
Control	778
HCC	555

```
# 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)
)
```

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)
```

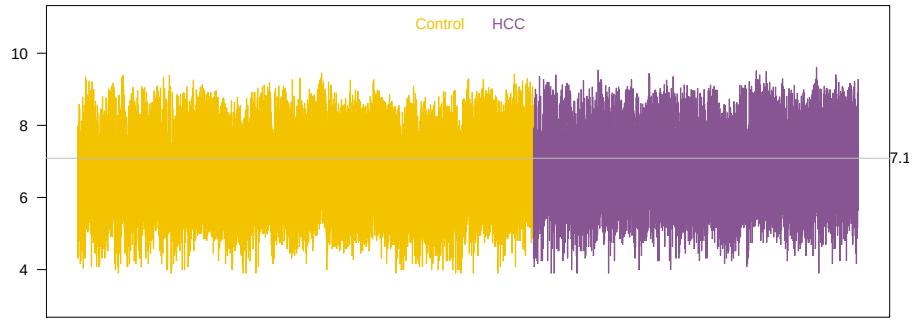

Figure 3.1: Sample log2 count boxplots

```
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,
  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.

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

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")
```

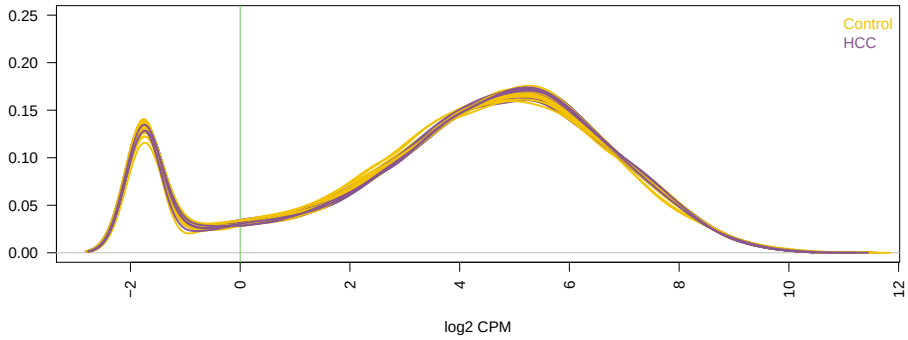

Figure 3.2: Sample log_2 CPM densities

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")
```

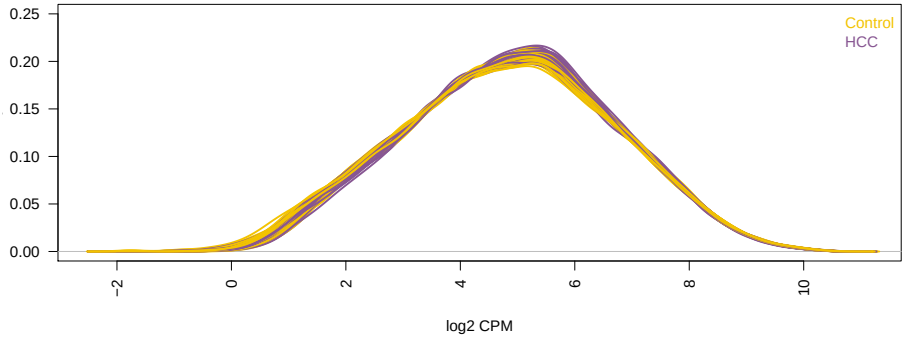

Figure 3.3: Sample log_2 CPM densities after removing weak genes

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)
```

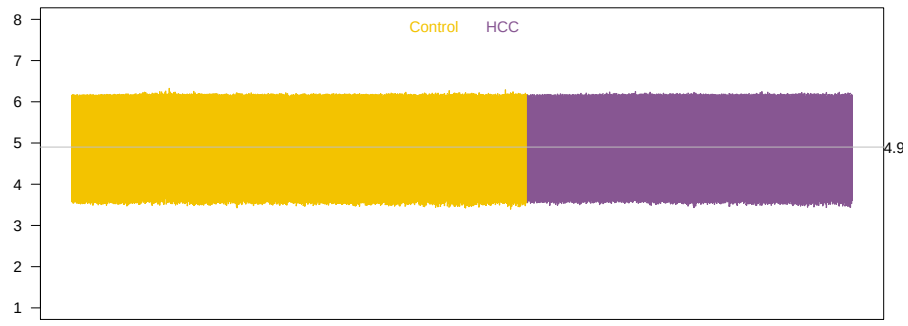

Figure 3.4: Sample log2 count boxplots

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
)
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
)
```

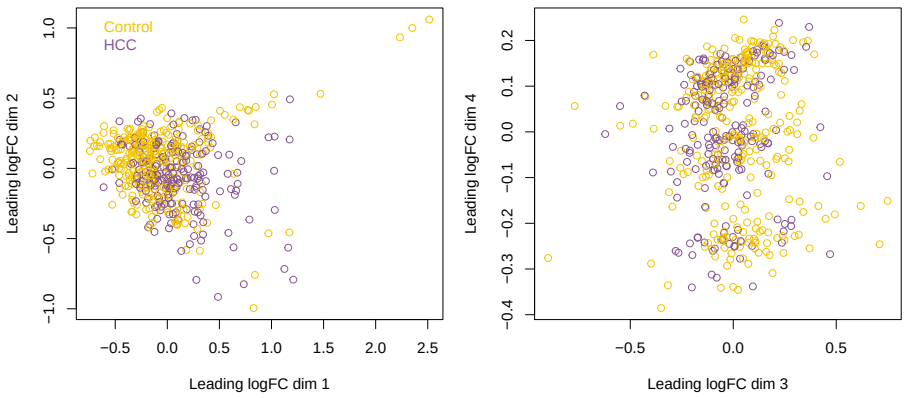

Figure 3.5: MDS plots of log-CPM values

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.

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 3.4 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) )
```

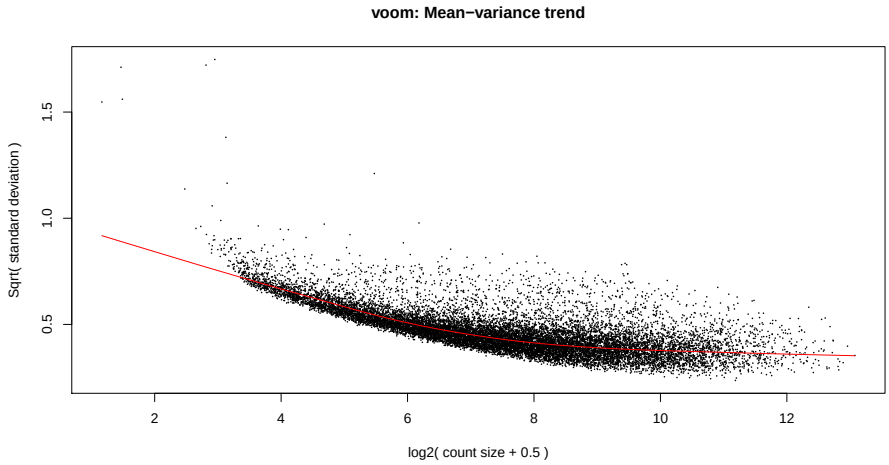

Figure 3.6: Removing heteroscedascity

Table 3.4: DE Results at FDR = 0.05

	Down	NotSig	Up
HCCvsControl	5214	5280	5258

```
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)
```

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)
```

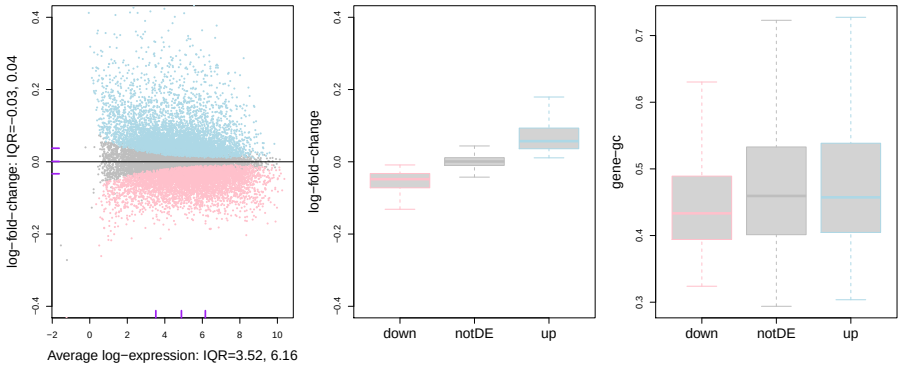

Figure 3.7: HCC vs Control - Genes Identified at FDR = 0,05

```
 #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)
```

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

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


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

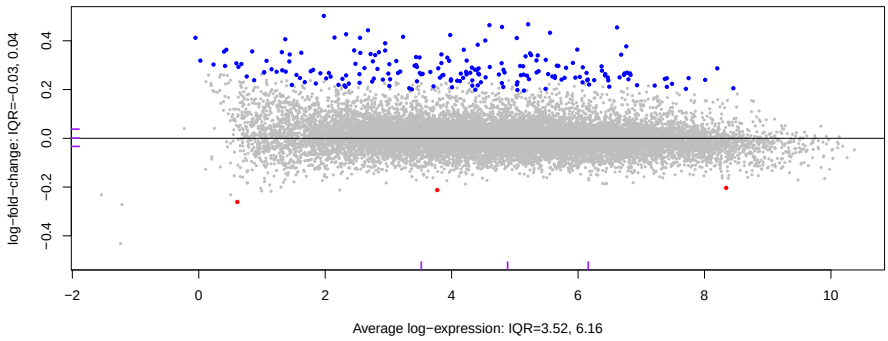

Figure 3.8: HCC vs Control - Identified Genes at FDR = 0,05 and logFC > 10%

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
```

Table 3.6: SNR Quantiles

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

```
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
  )
```

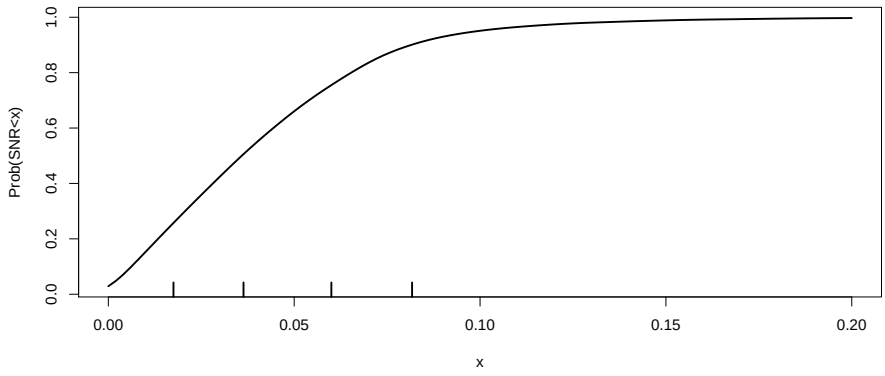

Figure 3.9: Cumulative Distribution of SNR - rug = 25, 50, 75 and 90th percentile

```
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.

4.1 CV analysis setup

```
K_FOLD <- 10
trainP <- 0.8
EPS <- 0.05    # Have no idea what "small" epsilon means
```

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"

- Consider models:
  - lasso: $\alpha = 1.0$ - sparse model
  - ridge $\alpha = 0$ - shrunken coefficients model
  - elastic net: $\alpha = 0.5$ - semi sparse model
- Does the relaxed lasso improve performance?
- Does the shrunken relaxed lasso (aka the blended mix) improve performance
- How sparse is the model undelying best 5hmC classifier for Early HCC vs Control?
- Is the degree of sparsity, or the size of the model, a stable feature of the problem and data set?

Table 4.3: training samples fold composition

	1	2	3	4	5	6	7	8	9	10
Control	62	62	62	63	62	62	63	62	63	62
HCC	45	44	44	45	44	45	44	45	44	44

In this analysis, we will only evaluate models in terms of model size, 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.

Next 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)

knitr::kable(sapply(split(train_group_vec, train_foldid_vec),
  table), 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.

4.2 Fit and compare models

- cross-validated accuracy
- test set accuracy

- sparsity
  - for lasso, enet , examine number of selected variables

Although "the one standard error rule" can produce a model with fewer predictors, it usually results in increased MSE and more biased parameter estimates (see Engebretsen et al. (2019) [26] for example). We will look at both the minimum cv error and the one standard error rule model preformance.

4.2.1 Logistic regression in `glmnet`

`glmnet` provides functionality to extract various predicted of fitted values from calibrated models. Note in passing 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. 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))
```

It is important to note that all *predicted* values extracted from `glmnet` fitted models by the **predict()** extraction method yield **fitted** values for design points that are part of the training data set. This includes the predicted class for training data which ae used to estimate misclassification error rates. As a result, the cv error rates quoted in various `glmnet` summaries are generally optimistic. `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 - ie. predictive of performance in unseen data - than

assessments based on values produced from the full fit, which by default is what `glmnet` extraction methods provide. See Höfling and Tibshirani (2008) [27] for a description of the use of pre-validation in model assessment.

Because the `keep=T` option will store predicted values for all models evaluated in the cross-validation process, we will limit the number of models tested by setting **nlambda=30** when calling the fitting functions. This has no effect on performance in this data set.

```
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: 12.62s
```

```
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: 104.46s
```

```
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: 12.26s
```

The ridge regression model takes over 10 times longer to compute.

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

 library(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]

 # Get oof error
 ndx_1se <- match(cv_fit$lambda.1se,cv_fit$lambda)
 train_oofPred_1se_vec <- ifelse(
  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(
  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: cv =", round(100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se], 1),
 "oof =", round(100*train_oofPred_1se_error, 1),
 "test =", round(100*test_pred_1se_error, 1)
))
LL <- LL+1
mtext(side=1, outer=F, line = LL, paste(
 #ifelse(Nzero, paste("min p =", lambda.min_p),''),
 "min: cv =", round(100*cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min], 1),
 "oof =", round(100*train_oofPred_min_error, 1),
 "test =", round(100*test_pred_min_error, 1)
))

cbind(
 error_1se = c(
  train_cv = cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.1se],
  train_oof = train_oofPred_1se_error,
```

```
   test = test_pred_1se_error),
  error_min = c(
   train_cv = cv_fit$cvm[cv_fit$lambda == cv_fit$lambda.min],
   train_oof = train_oofPred_min_error,
   test = test_pred_min_error)
  )

}
```

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
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 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)
```

```
errors_frm <- data.frame(
  lasso = lasso_errors_mtx, ridge = rifge_errors_mtx, enet = enet_errors_mtx
)

knitr::kable(t(errors_frm)*100,
 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. These is very little difference between the min lambda and the the one standard error rule

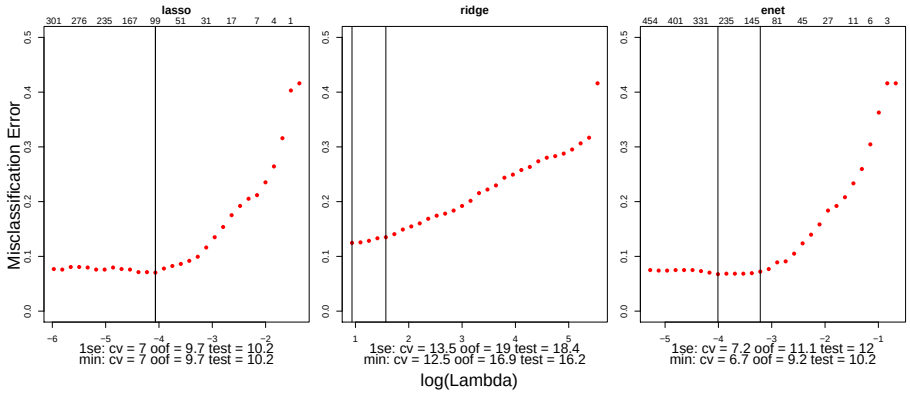

Figure 4.1: compare fits

Table 4.4: Misclassifiaction error rates

	train_cv	train_oof	test
lasso.error_1se	7.0	9.7	10.2
lasso.error_min	7.0	9.7	10.2
ridge.error_1se	13.5	19.0	18.4
ridge.error_min	12.5	16.9	16.2
enet.error_1se	7.2	11.1	12.0
enet.error_min	6.7	9.2	10.2

lambda models (the two are the same for the lasso in this data set). We also see that the training data out-of-fold estimates of misclassification error rates are much closer to the test set estimates than are the cv estimated rates. This has been our experience with regularized regression models fitted to genomic scale data. It should also be noted that the cv estimates of misclassification rates become more biased as the sample size decreases, as we will show in Section 5.

4.3 Relaxed lasso and blended mix

Next we look at the so-called `relaxed lasso` and the `blended mix` which is an optimized shrinkage between the relaxed lasso and the regular lasso.

```
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
# Get relaxed lasso (gamma=0) oof error
train_oofPred_relaxed_1se_vec <- ifelse(
  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
```

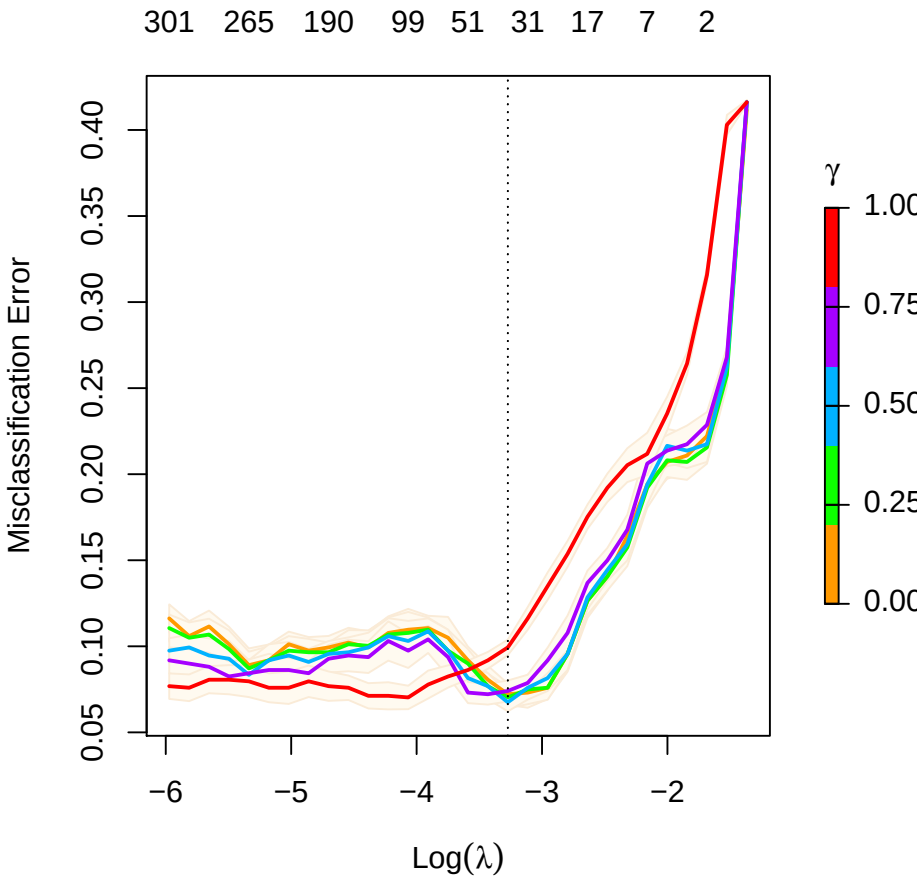

Figure 4.2: lassoR fit

```r
# blended mix (gamma=0.5)
train_oofPred_blended_1se_vec <- ifelse(
  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_sum["1se", "Measure"]
cv_lassoR_min_error <- cv_lassoR_sum["min", "Measure"]

knitr::kable(t(data.frame(
  train_lassoR_cv = cv_lassoR_1se_error,
  train_blended_oof = train_oofPred_blended_1se_error,
  train_relaxed_oof = train_oofPred_relaxed_1se_error,
  test_blended_oof = test_pred_blended_1se_error,
  test_relaxed_oof = test_pred_relaxed_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 quite optimistic, while out-of-fold error rates continue to be good indicators of unseen

Table 4.5: Relaxed lasso and blended mix error rates

train_lassoR_cv	6.7
train_blended_oof	10.2
train_relaxed_oof	11.1
test_blended_oof	10.2
test_relaxed_oof	10.9

data error rates.

4.4 Examination of sensitivity vs specificity

In the results above we reported error rates without inspecting the sensitivity versus specificity trade off. Here we look at this question with the help of ROC curves.

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_lassoR_oofProb_vec <- logistic_f(cv_lassoR$fit.preval[['g:0']][,ndx_1se])
train_lassoR_roc <- pROC::roc(
 response = as.numeric(train_group_vec=='HCC'),
 predictor = train_lassoR_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_lassoR_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('lassoR =', round(train_lassoR_roc[['auc']],3)),
  paste('blended =', round(train_blended_roc[['auc']],3))
 ),
 text.col = col_vec[1:4],
 bty='n'
)
```

Compare thresholds for 90% Specificity:

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

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

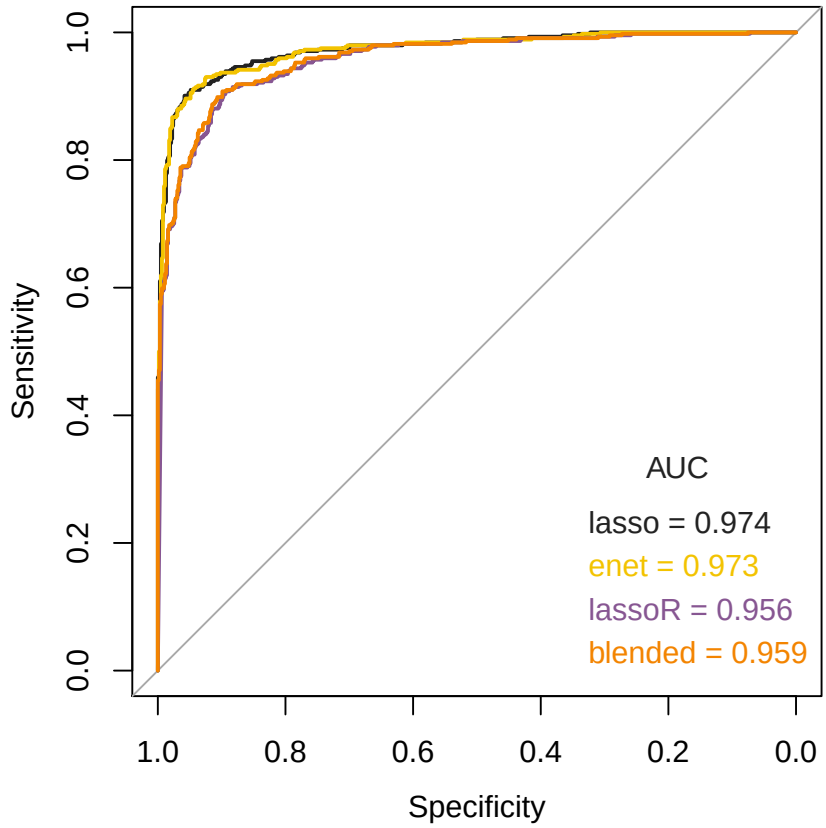

Figure 4.3: Train data out-of-sample ROCs

Table 4.6: Specificity = .90 Coordinates

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

```
lassoR_ndx <- with(as.data.frame(pROC::coords(train_lassoR_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,],
 lassoR=as.data.frame(pROC::coords(train_lassoR_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_lassoR_oofProb_vec[train_group_vec == "Control"]),
  xlim = c(0, 1), main = "", xlab = "", ylab = "", col = "green"
)
lines(density(train_lassoR_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 arr 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_lassoR_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)),
    paste("enet =", round(test_enet_roc[["auc"]], 3)),
```

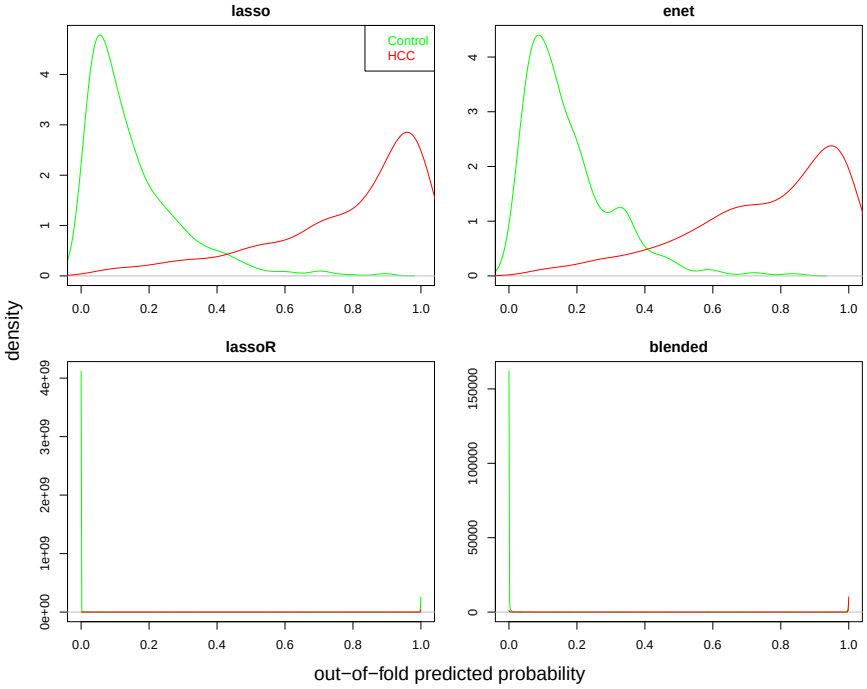

Figure 4.4: Train data out-of-fold predicted probabilities

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

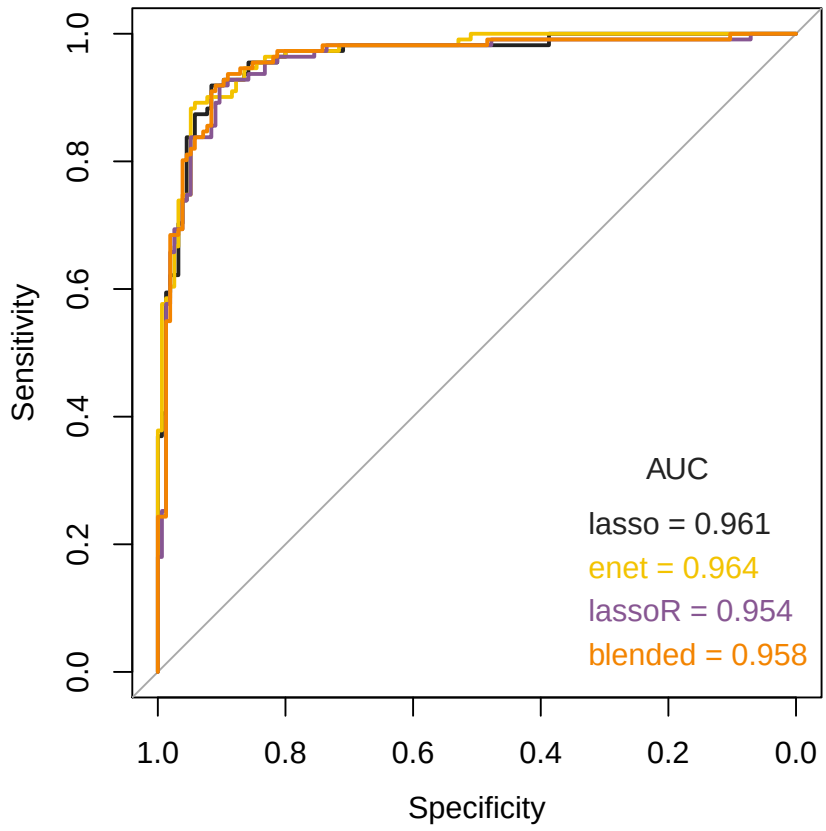

Figure 4.5: Test data out-of-sample ROCs

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")

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

#sapply(split(test_lassoR_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)


# Define plotting function
bxpPredProb_f <- function(cv_fit, Gamma=NULL) {
```

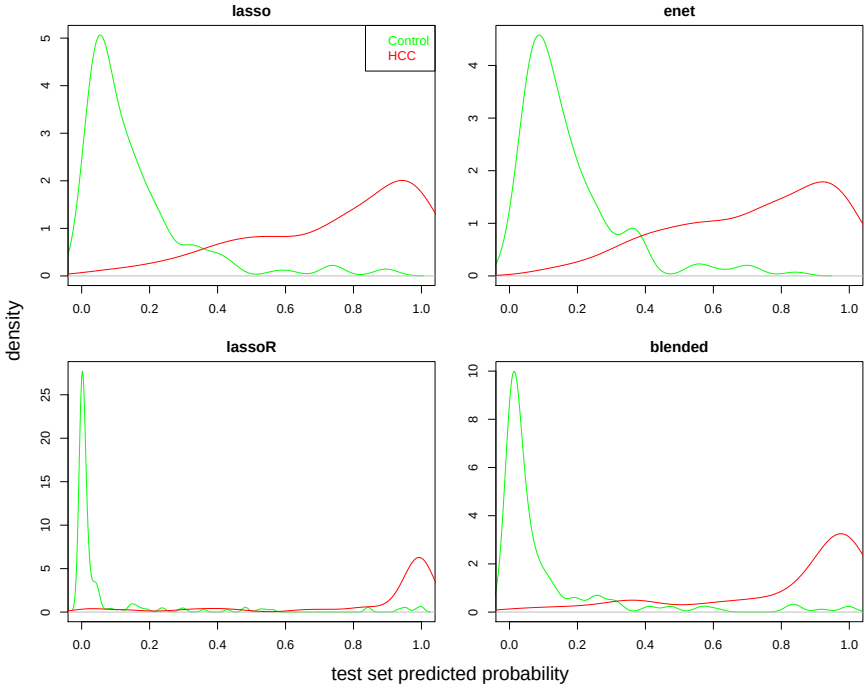

Figure 4.6: Test data out-of-fold predicted probabilities

```
  # 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('lassoR')

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

We have seen above that assessments of model performance based on the out-of-fold predicted values are close to the test set assessments, and that assessments based on prediction extracted from glmnet object are optimistic. Here we look at confusion matrices to see how this affects the classification results.

Here we us a threshold of 0.5 to dichotomize the predicted probabilities into a class prediction, as is done in the glmnet predictions.

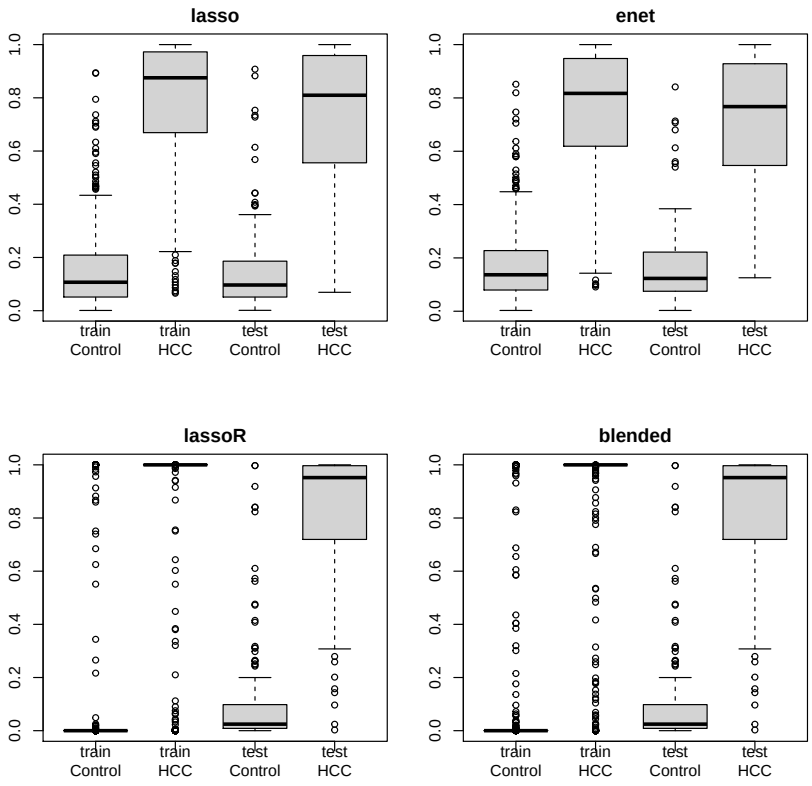

Figure 4.7: Predicted Probabilities - Train and Test

```
# lasso
##########################
# train - cv predicted
train_lasso_predClass_vec <- predict(
 cv_lasso,
 newx=train_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

# 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')

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

# enet
##########################
# train - cv predicted
train_enet_predClass_vec <- predict(
 cv_enet,
 newx=train_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

# train - oof
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')

# test
test_enet_predClass_vec <- predict(
 cv_enet,
 newx=test_lcpm_mtx,
 s='lambda.1se',
```

```
 type='class'
)


# lasso - relaxed (gamma=0)
############################
# train - cv predicted
train_lassoR_predClass_vec <- predict(
 cv_lassoR,
 g=0,
 newx=train_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

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

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


# blended mix (gamma=0.5)
##################################
# train - cv predicted
train_blended_predClass_vec <- predict(
 cv_lassoR,
 g=0.5,
 newx=train_lcpm_mtx,
 s='lambda.1se',
 type='class'
)

# train - oof
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_oofClass_vec <- ifelse(
   train_blended_oofProb_vec > 0.5, 'HCC', 'Control')

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

# put it all together
########################
all_models_confustion_mtx <- rbind(
 train_lasso_cv = as.vector(table(train_lasso_predClass_vec, train_group_vec)),
 train_lasso_oof = as.vector(table(train_lasso_oofClass_vec, train_group_vec)),
 test_lasso = as.vector(table(test_lasso_predClass_vec, test_group_vec)),
 train_enet_cv = as.vector(table(train_enet_predClass_vec, train_group_vec)),
 train_enet_oof = as.vector(table(train_enet_oofClass_vec, train_group_vec)),
 test_enet = as.vector(table(test_enet_predClass_vec, test_group_vec)),
 train_lassoR_cv = as.vector(table(train_lassoR_predClass_vec, train_group_vec)),
 train_lassoR_oof = as.vector(table(train_lassoR_oofClass_vec, train_group_vec)),
 test_lassoR = as.vector(table(test_lassoR_predClass_vec, test_group_vec)),
 train_blended_cv = as.vector(table(train_blended_predClass_vec, train_group_vec)),
 train_blended_oof = as.vector(table(train_blended_oofClass_vec, train_group_vec)),
 test_blended = as.vector(table(test_blended_predClass_vec, test_group_vec))
)
colnames(all_models_confustion_mtx) <- c('C:C','C:H','H:C', 'H:H')


all_models_confustionRates_mtx <- sweep(
 all_models_confustion_mtx, 1, rowSums(all_models_confustion_mtx), '/')

all_models_confustionRates_mtx <- cbind(all_models_confustionRates_mtx,
  error = rowSums(all_models_confustionRates_mtx[,2:3]))

knitr::kable(100*all_models_confustionRates_mtx,
  caption="confusion: Columns are Truth:Predicted",
  digits=1) %>%
  kableExtra::kable_styling(full_width = F)
```

Table 4.7: confusion: Columns are Truth:Predicted

	C:C	C:H	H:C	H:H	error
train_lasso_cv	57.6	0.7	2.9	38.7	3.7
train_lasso_oof	56.7	1.7	5.3	36.3	7.0
test_lasso	55.6	2.6	7.5	34.2	10.2
train_enet_cv	57.6	0.7	3.8	37.8	4.6
train_enet_oof	57.1	1.3	5.9	35.7	7.2
test_enet	55.3	3.0	9.0	32.7	12.0
train_lassoR_cv	56.7	1.7	3.0	38.6	4.7
train_lassoR_oof	53.8	4.6	6.4	35.2	11.0
test_lassoR	54.1	4.1	6.8	35.0	10.9
train_blended_cv	57.1	1.3	3.4	38.2	4.7
train_blended_oof	54.3	4.1	6.2	35.4	10.3
test_blended	54.9	3.4	6.8	35.0	10.2

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.

```
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_lassoR_oofClass_vec)[train_lassoR_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_lassoR_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)
```

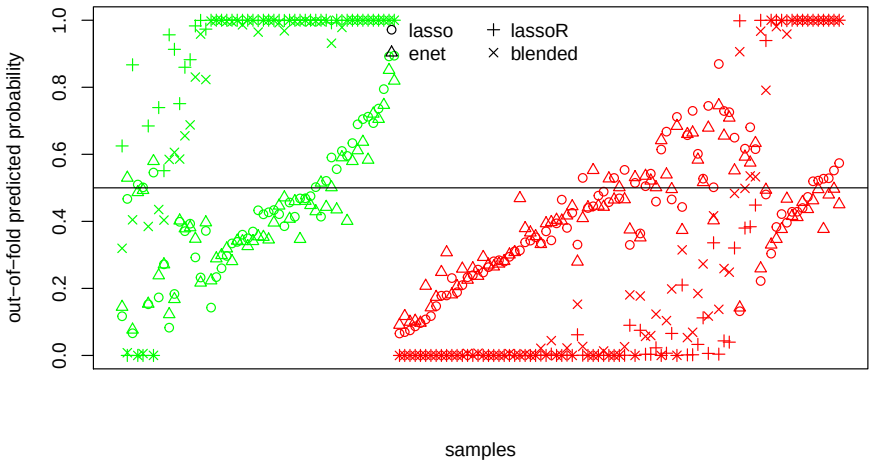

Figure 4.8: out-of-fold predicted probabilities at miscassified samples

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.

```
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_lassoR_predClass_vec[,1])[test_lassoR_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_lassoR_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)
```

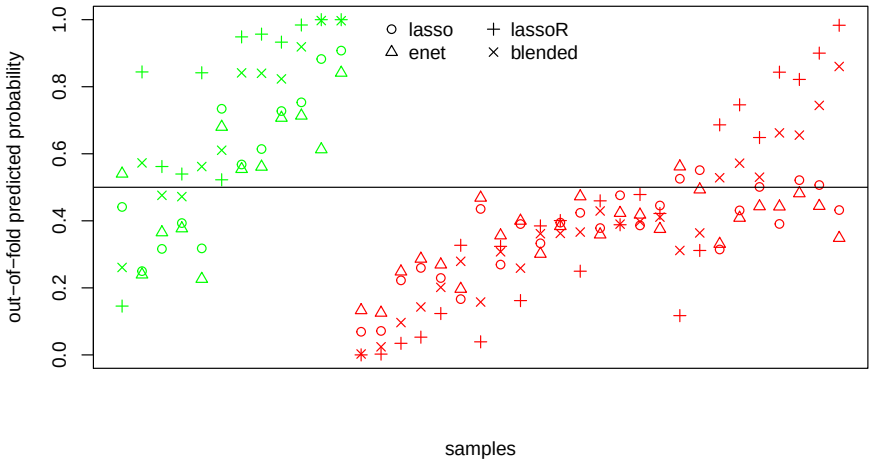

Figure 4.9: Test data predicted probabilities at miscassified samples

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)

# lasso - relaxed (gamma=0)
##########################
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,
 y = base::merge(
     x = enet_coef_frm,
     y = base::merge(
         x = lassoR_coef_frm,
         y = blended_coef_frm,
         by='gene', all=T),
     by='gene', all=T),
 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')
}
```

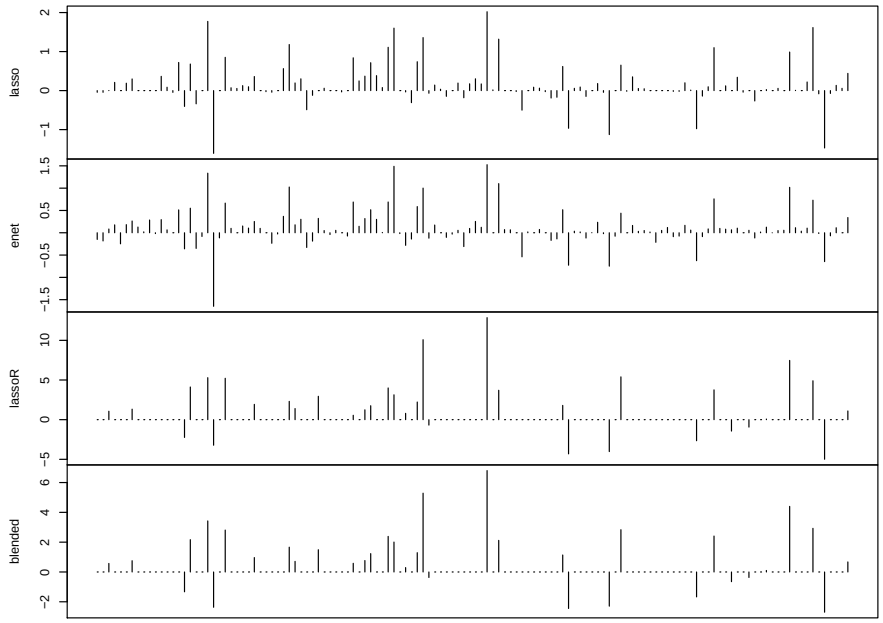

Figure 4.10: Coefficient Profiles

Coefficients in the relaxed lasso fit are much larger than those in the lasso fit, or zero. As a consequence, the blended fit coefficients look like a shrunken version of the relaxed lasso fit coefficients.

We can also examine these with a scatter plot matrix.

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

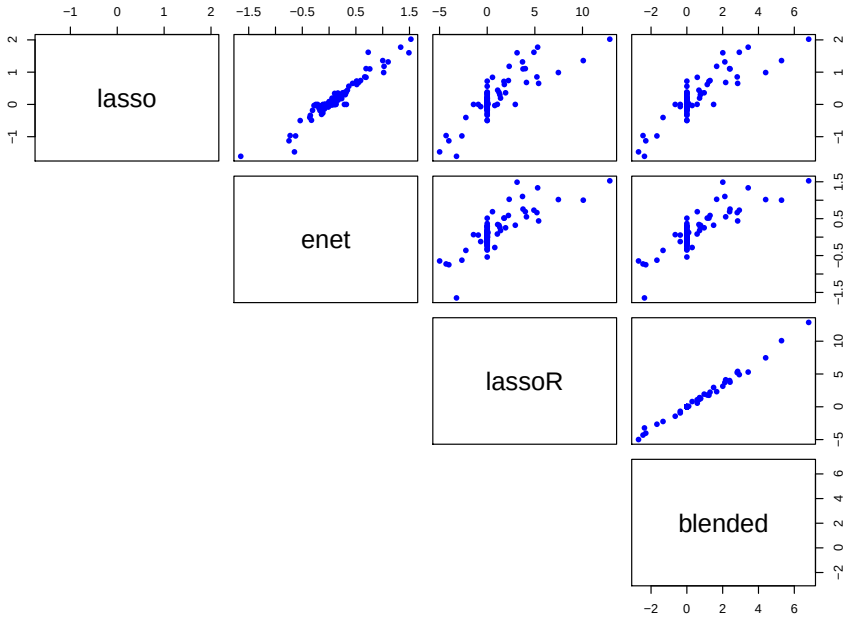

Figure 4.11: Coefficients from fits

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 stabi
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.

```
 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))
```

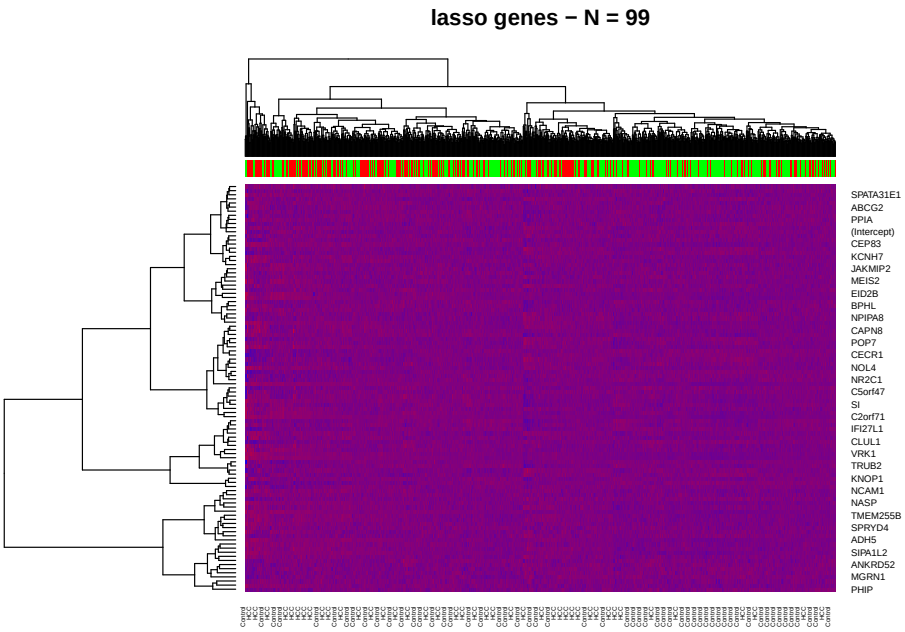

Figure 4.12: Lasso Model Genes

```
 suppressPackageStartupMessages(require(gplots))

# train - cv predicted
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))
```

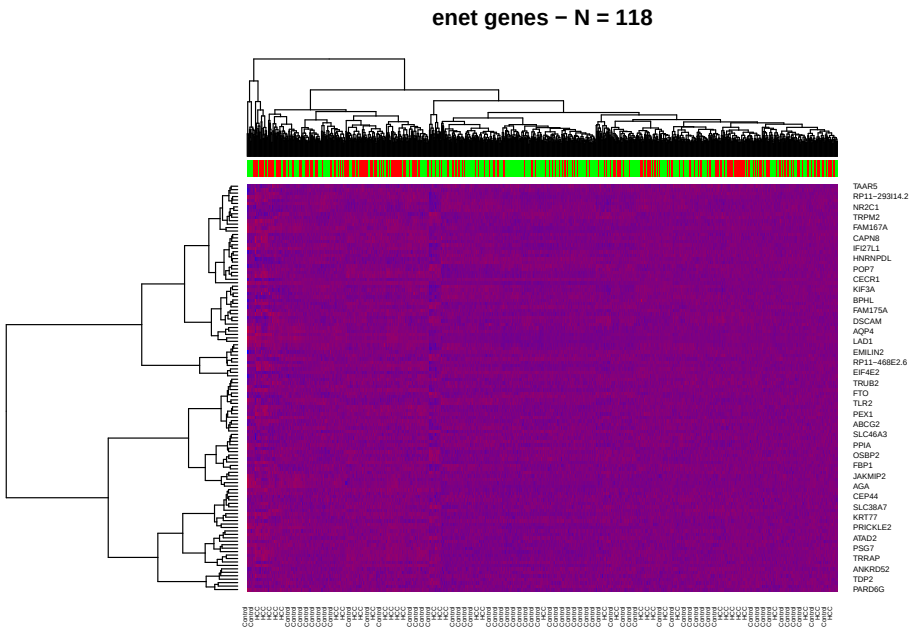

Figure 4.13: Enet Model Genes

Chapter 5

Fitted Model Suite

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

- assessed accuracy
  - out-of-fold estimates of precision and variability and cv assessed accuracy bias.
- 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 and stable selected feature profiles.

We will attempt to separate variability which is due to sample size from variability due to sample composition. To to this we will track sample quality. 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 likelihood as a sample quality score. To derive the 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.

5.1 Sample quality scores

5.2 Model suite

5.2.1 Simulation setup

Chapter 6

Variable importance

Cai et al. [2] used the fequency of selection across bootstrap replicates to select features form their final model. We sus[ect that this simply selects features with large coefficients which would correspond the the variable importance metric used in the `caret` package. In this section we measure variable importance as the loss in performance when the variable is left out of the model. Depending on the correlation structure, it is quite possible for a feature to have a large coefficient and be un-important, in our sense of the word. It is also possible for a feature to be frequently selected in bootstrap samples, and still not un-important. In a sense, this measure of importance is really a measure of `single importance`. We should therefore also have a measure of `group importance`.

Other questions ...

Chapter 7

Conclusions

We have found that ...

Other questions ...

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