# Comparing clustering algorithms

Hin-Tak Leung

April 15, 2011

## 1 Introduction

This document is the accumulation of a few somewhat unrelated threads of development:

- During the WTCCC study (phase 1, for which `chopsticks` was written), some multi-panel colourised cluster-plot drawing code in R, mainly to verify the validity of cluster across cohorts. e.g. a 3x3 panel works very well for 7 common diseases plus 2 controls. This becomes `snp.clust.plot` function described later.
- When I noticed from looking at the cluster plots of some Illumina data I was working on that Illuminus is buggy and plainly wrong when calling the sex chromosomes, I wrote some simple low-level composite C/R code to catalogue the differences between Illuminus and GenTrain 1 to find out on which SNP they behaves most differently. This eventually becomes the `snp.compare` function described later.
- Almost 4 years after GenTrain 1 stabilised (in January 2006), Illumina debuted an alternative genotype-calling algorithm called "GenTrain 2" in December 2009 with the release of GenomeStudio 2009.2; "GenTrain 2" becomes the default with fresh installation of the next release, GenomeStudio 2010.1 (March 2010). `SpringOnion` added the ability to run "GenTrain 2" very soon afterwards.

Given that "GenTrain 2" is now the default, obviously Illumina staff must believe that it does a better job than "GenTrain 1" (besides its being marginally about 5% faster). The question is, in what way? We have a way of running the same raw data through two different algorithms, a way of cataloguing differences, and a way of displaying them side-by-side. So that's what the rest of the document does.

Similar techniques can be used for studying differences between any two algorithms.

## 2 Finding the largest differences

Using the 1550 samples of 99 SNPs of the Illuminus example set (the data is neither typical of BeadArray nor very good, but "real" data requires scrubbing and obfuscating). Reading the genotypes in:

```
> library(chopsticks)
> load(system.file("data/Genotypes.GenTrain1.RData", package = "chopsticks"))
> load(system.file("data/Genotypes.GenTrain2.RData", package = "chopsticks"))
```

Reading the raw cluster data in:

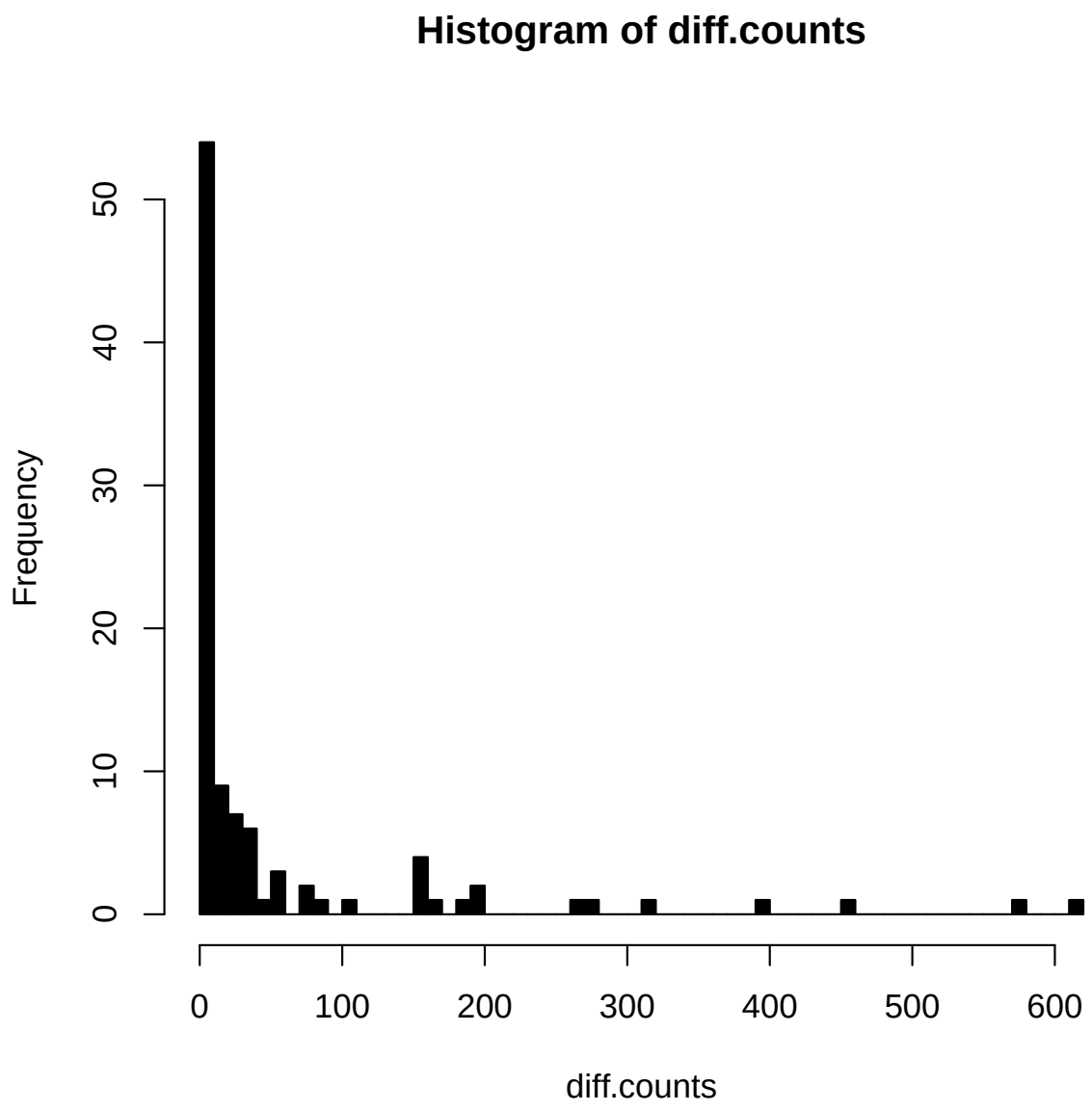
```
> ab.signals <- read.wtccc.signals(system.file("extdata/example-new.txt",
+     package = "chopsticks"), paste("rs", 1:99, sep = ""))

Reading /private/tmp/RtmpwMCGLj/Rinst3406aae1/chopsticks/extdata/example-new.txt ...
Can take a while...

...Done
```

We'll just quickly run chopsticks's SNP summary function to verify:

```
> col.summary(GenTrain2)[1:3, 2:7]

    Call.rate       MAF       P.AA      P.AB      P.BB       z.HWE
rs1 0.9257106 0.4228890 0.16957432 0.5066294 0.3237962  1.43643827
rs2 0.9728682 0.1377822 0.01925631 0.2370518 0.7436919 -0.08897442
rs3 0.9773902 0.4372108 0.19035030 0.4937211 0.3159286  0.12694323
```

Calculate the largest difference count of the two genotype objects, and plot a histogram:

```
> result <- snp.compare(GenTrain1, GenTrain2)
> diff.counts <- result$count
> hist(diff.counts, breaks = 50, col = "black")
```




More than half of the SNPs shows zero or close to zero differences between GenTrain 1 and GenTrain 2.

**Lesson 1: For most SNPs (which are well-behaved and have 3 well-separated clusters), any half-decent calling algorithms do almost exactly the same thing.**

**Corollary 1: One must look at where two algorithms differ to see which one is "correct" more often.**

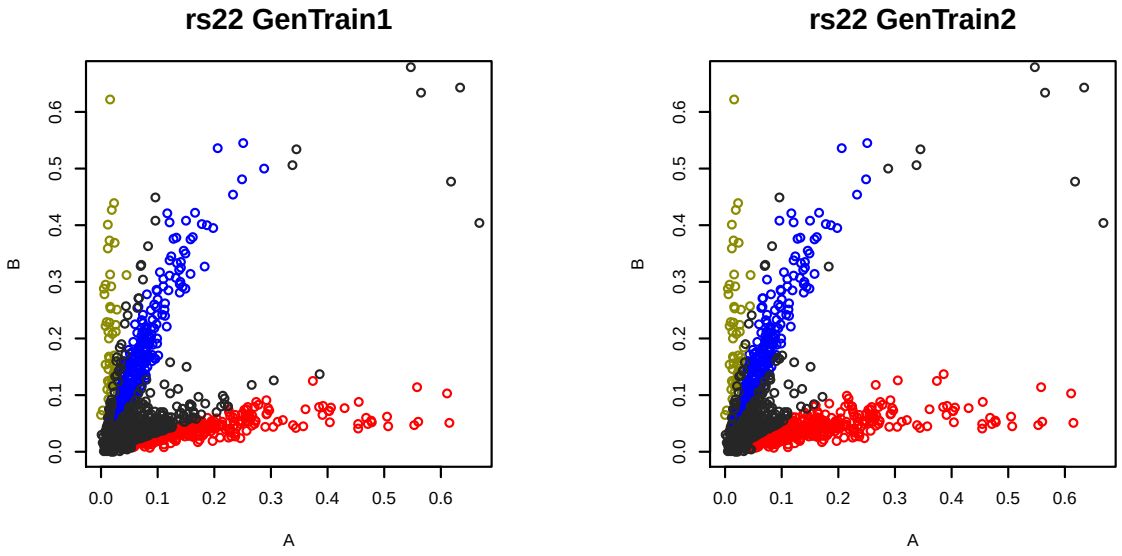
**Corollary 2: If one can make sure that where two algorithms differ, one algorithm is more "correct" than the other, then that's what matters.**

```
> dim(GenTrain1)

[1] 1550   99
```

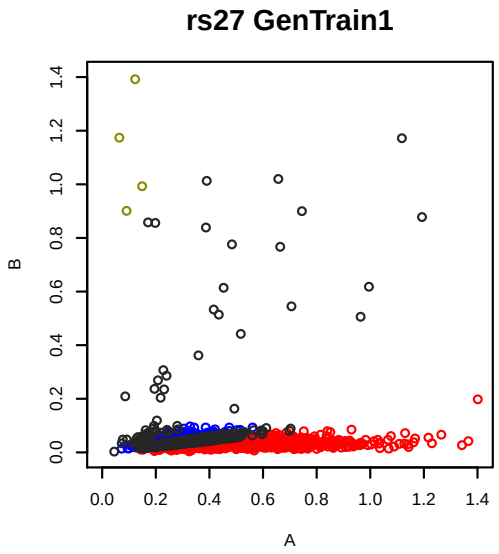
```
> diff.counts[diff.counts > 1550 * 0.1]

 rs5  rs8 rs13 rs22 rs24 rs27 rs37 rs62 rs71 rs77 rs80 rs94
 192  263  160  320  166  576  187  617  195  280  454  398

> diff.counts[diff.counts > 1550 * 0.05]

 rs5  rs8 rs13 rs19 rs22 rs24 rs27 rs36 rs37 rs53 rs56 rs59 rs62 rs71 rs77 rs80
 192  263  160  103  320  166  576  151  187   82  151  154  617  195  280  454
rs94
 398

> worst.snps <- names(diff.counts[diff.counts > 1550 * 0.2])
> worst.snps

[1] "rs22" "rs27" "rs62" "rs80" "rs94"
```

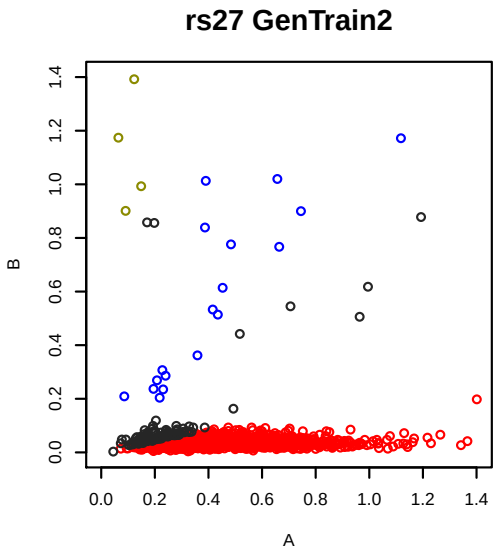
There are 5 SNPs with more than 20% difference. Let's plot these 5 SNPs:

```
> par(mfrow = c(1, 2), plt = c(0.25, 0.9, 0.15, 0.8), col.main = "black",
+     col.axis = "black", cex = 0.5, cex.main = 1.7)
> snp.clust.plot(ab.signals[["rs22"]], GenTrain1[, "rs22"], title = "rs22 GenTrain1")
> snp.clust.plot(ab.signals[["rs22"]], GenTrain2[, "rs22"], title = "rs22 GenTrain2")
```




GenTrain 2 does a better job at calling more of the homozygous AA for rs22.

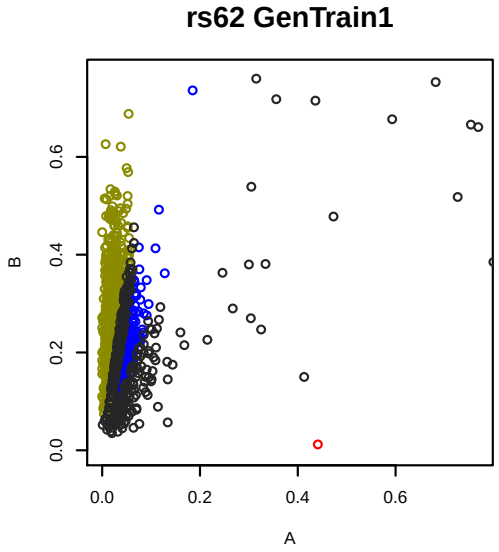


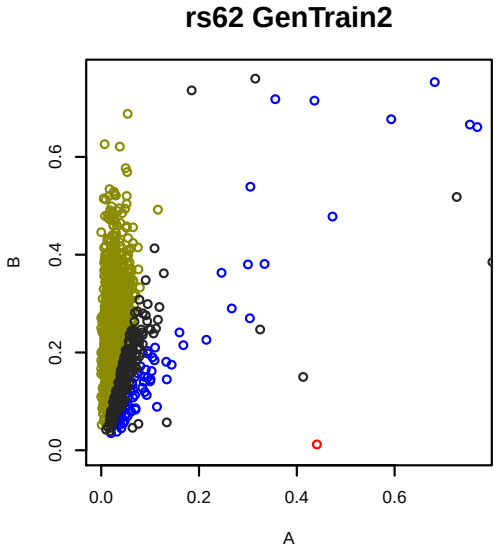



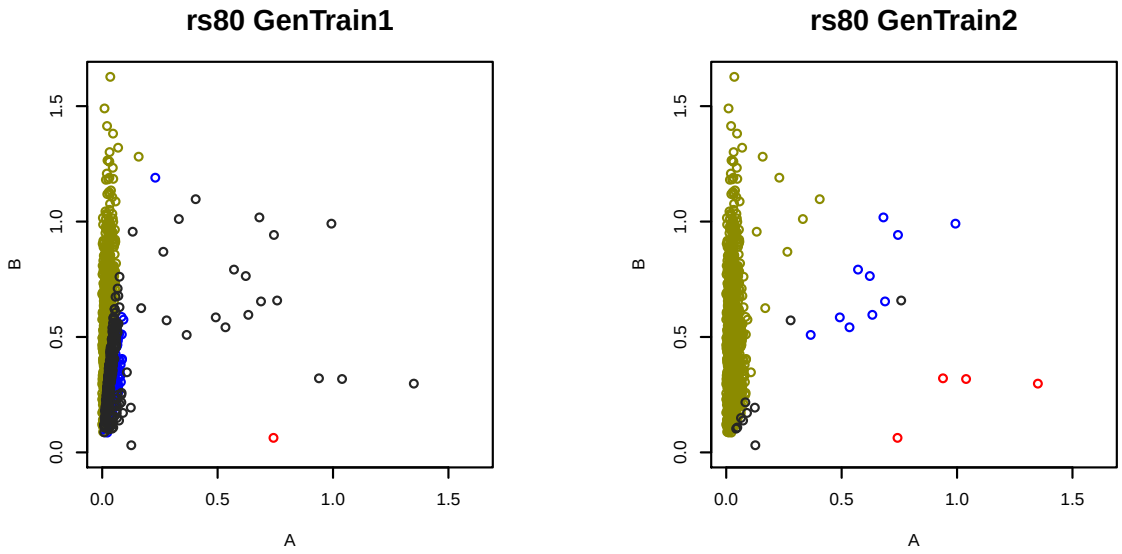

SNP rs27 is almost monomorphic — while one is inclined to say GenTrain 2 is better, GenTrain 2 also mis-behaves more than GenTrain 1 in calling some of the outliers (which were not called by GenTrain 1).

There are often wishful thinking along the line that higher call rates are always better. This bias often comes from the idea that since one already spent the effort for the samples to be processed, it is nicer to get *a* genotype than not getting one.

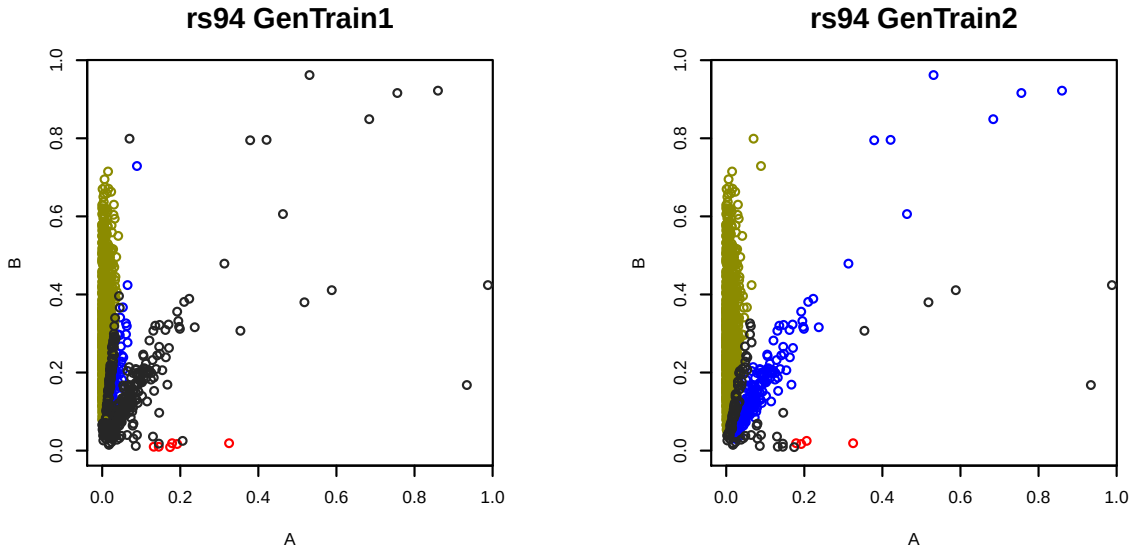
**Lesson 2: Being optimistic (calling more) can be wrong.**






SNP rs80 is somewhat similar to rs27 in that the SNP is near monomorphic and GenTrain 2 decides (somewhat arbitrarily) to classify outliers into minor clusters.

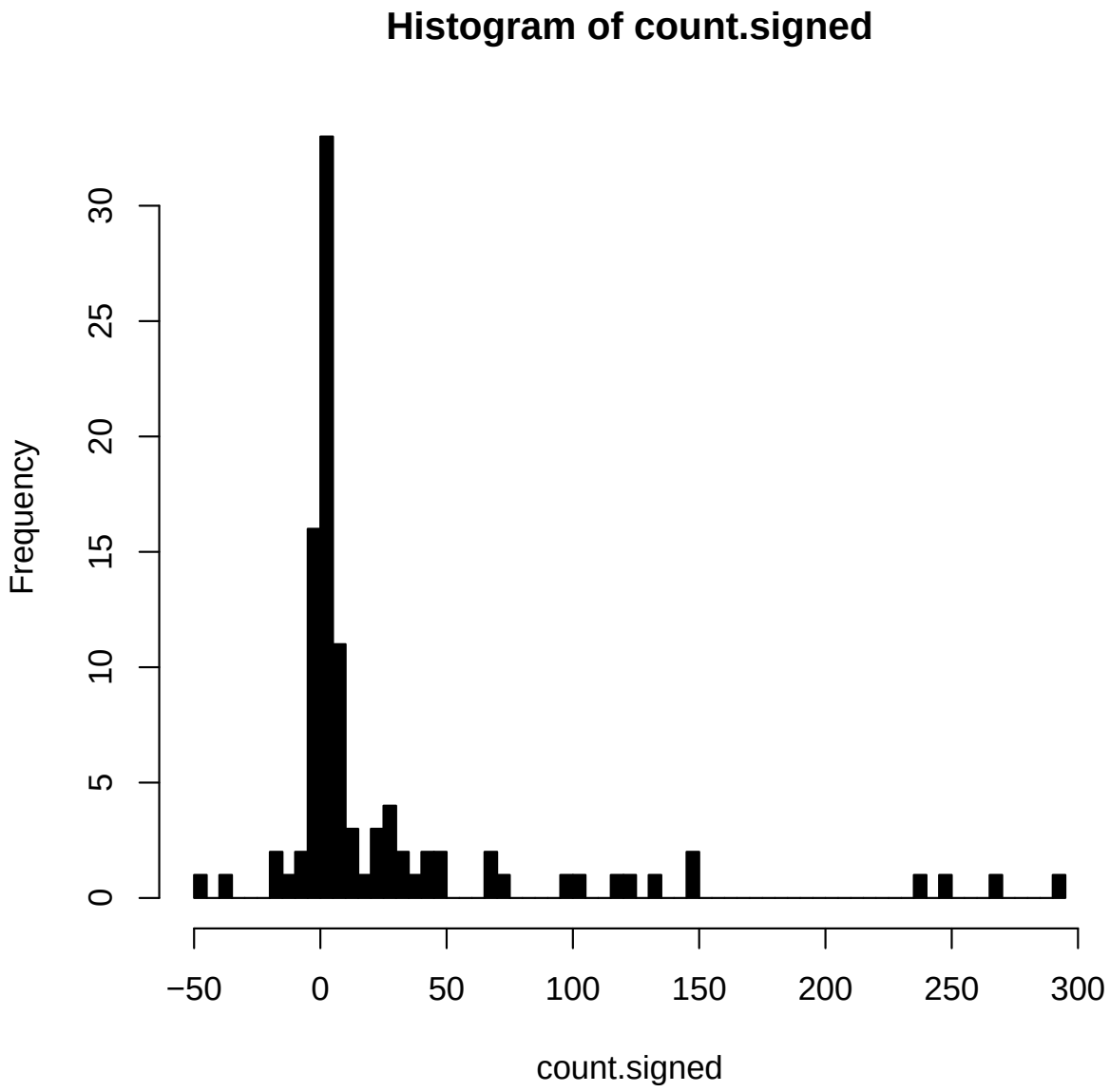



GenTrain 2 works very well on SNP rs94.

## 2.1 Does GenTrain 2 ever work less well than GenTrain1?

GenTrain 2 gives almost uniformly higher call rate across all SNPs:

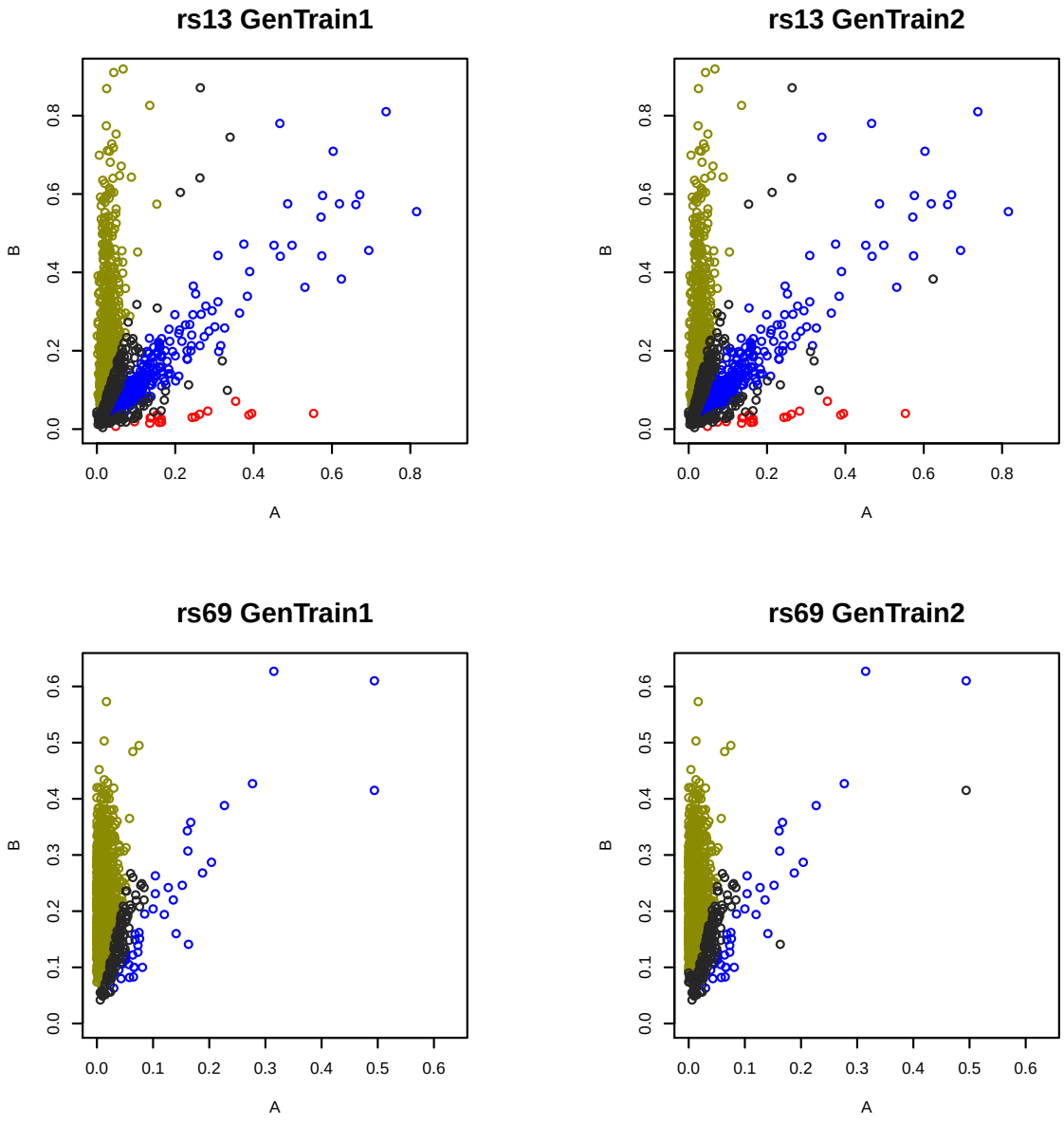
```
> count.signed <- result$count.signed
> hist(count.signed, breaks = 50, col = "black")
```




The two SNPs which give somewhat lower call rates under GenTrain 2 are:

```
> result$count.signed[result$count.signed < -30]

rs13 rs69
 -36  -46
```

The differences are some what subtle when the cluster plots are examined:




## 3 Call rates and HWE

Let's have a look at the call rates, etc of those 5 worst SNPs again:

```
> col.summary(GenTrain1)[worst.snps, 2:7]
```

```
     Call.rate        MAF          P.AA       P.AB         P.BB     z.HWE
rs22 0.4547804 0.31321023 0.4474431818 0.47869318 0.073863636  2.989549
rs27 0.7571059 0.10878840 0.7858361775 0.21075085 0.003412969  2.973807
rs62 0.7622739 0.13898305 0.0025423729 0.27288136 0.724576271  4.815062
```

```
rs80 0.8049096 0.06581059 0.0008025682 0.13001605 0.869181380  2.025882
rs94 0.7855297 0.04934211 0.0057565789 0.08717105 0.907072368 -2.469540
```

```
> col.summary(GenTrain2)[worst.snps, 2:7]
```

```
     Call.rate         MAF        P.AA        P.AB        P.BB      z.HWE
rs22 0.6149871 0.231092437 0.598739496 0.340336134 0.060924370  -1.305913
rs27 0.9315245 0.009015257 0.984743412 0.012482663 0.002773925 -11.445045
rs62 0.8585271 0.031226486 0.002257336 0.057938299 0.939804364  -1.545210
rs80 0.9935401 0.005851756 0.002600780 0.006501951 0.990897269 -17.301685
rs94 0.9386305 0.071576050 0.003441156 0.136269787 0.860289057   0.964802
```

In all cases, GenTrain 2 gives higher rates. We also see that GenTrain 2 allows HWE to deviate from normal a lot. Looking back at the cluster plots, we can see how that happens: GenTrain 2 calls more outliers when MAF is low and the SNP is near monomorphic. Or, conversely, near-monomorphic SNPs (which could also be clustering errors, i.e. where two of the clusters are mis-clustered as one, and lone outliers are classified as the 3rd cluster) shows up as low call-rates under GenTrain 1 (which tends to put an arbitrary cut in the middle), while GenTrain 2 draws in outliers and the SNP shows up with strong deviation from HWE.

Under the conventional filtering rules: a sufficiently high call rate (say 95%) and good HWE ($|\text{HWE}| < 5$), all 5 are rather poor under GenTrain 1, but rs94 becomes almost acceptable (as is shown in the cluster plot) under GenTrain 2.

In any case, as a final remark: GenTrain 2 is the new default — it doesn't necessarily mean it is any good, but 3rd-party alternative needs to compared well with it under "fair" conditions where both are run optimally.