

Analysis of Illumina microarray data

Mark Dunning

mark.dunning@cruk.cam.ac.uk

Cancer Research UK
Cambridge Institute
Robinson Way
Cambridge

30th January 2014

Outline

1 Introduction

2 Pre-processing

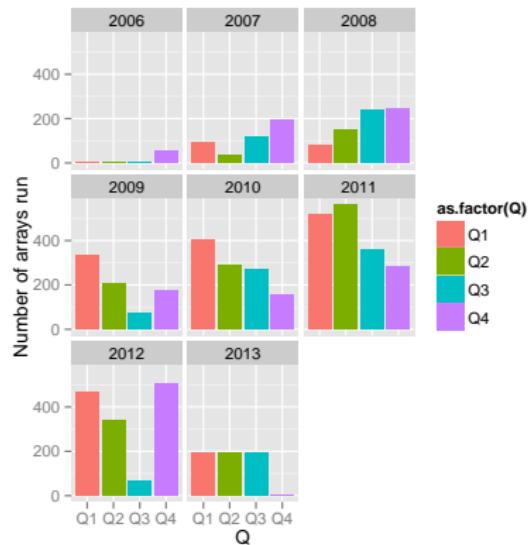
3 Probe Annotation

A brief history of Illumina

Now best-known as a sequencing company, Illumina are just one of several companies offering whole-genome microarrays.

- Hapmap (genotyping)
- GeneVar (expression)
- Metabric (expression)
- TCGA (methylation, genotyping)
- iCOGS (genotyping)

They have been the technology of choice at CRI since 2006



Are they still relevant?

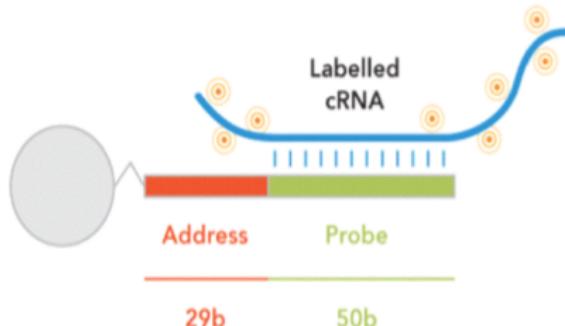
- Wealth of data available online e.g. on [GEO](#)
- Useful as a validation platform
- Methods are established and well-understood
- Cheaper? And easier access to equipment
- Some of the lessons learnt from microarray analysis can be applied to sequencing data
- Useful introduction to the world of Bioconductor

Microarray recap

- Microarrays use *hybridisation* to measure abundance of mRNA transcripts
- Pre-designed *probes* attached to solid surface
- Use *labels* to measure hybridization intensity
- *Scanned* image translated into probe intensities

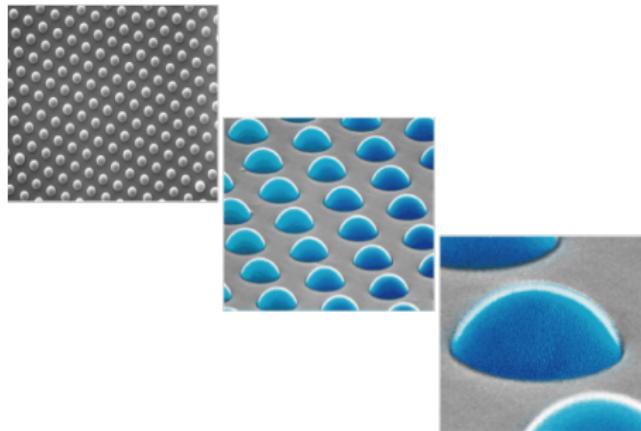
Technology overview

- Single-channel microarrays
- 50 base-pair 'probes' with the same sequence are attached to a bead
- Beads that have the same probe sequence are known as a *bead-type*
- Beads are identified by Illumina using a decoding sequence



Randomly-arranged arrays

- Arrays of randomly-arranged beads are constructed
- The number of replicates of each bead-type follows a normal distribution
- Around 20 replicates on most-recent arrays



Combinations of arrays and nomenclature

- 'Refseq' chips with 24,000 bead-types per array, 8 arrays per chip
- WG- chips with 48,000 bead-types per array, 6 arrays per chip
- HT-12 chips with 48,000 bead-types per array, 12 arrays per chip
- Each chip also has an annotation 'version' (v1, v2, v3, v4)



Example

Observations for Gene X on Illumina and older array technologies

Illumina



Array 1



Array 2



Array 3



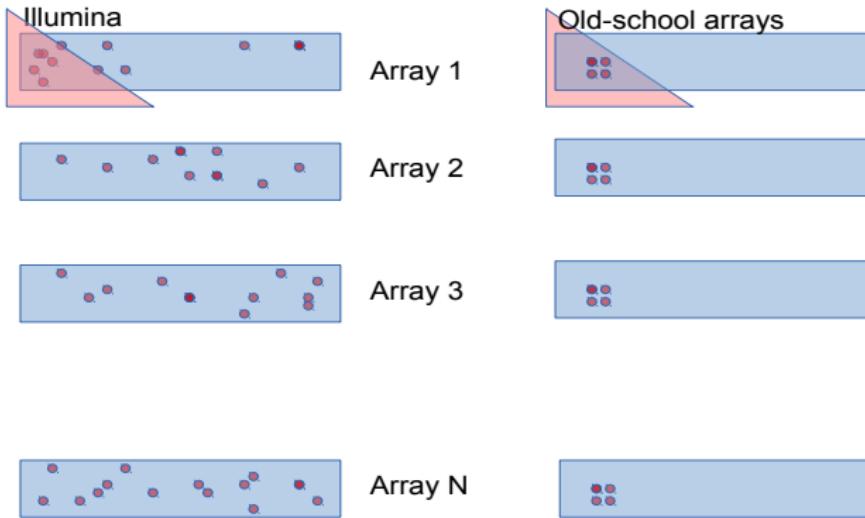
Array N

Old-school arrays



Example

Consider a technical failure on one of the arrays



Output formats

TIFF image representation of the array surface



Output formats

A text description of the bead locations, unique to each array.

```
Code....Grn..GrnX...GrnY
0010008 0934 1649.4 3650.0
0010008 0504 1866.5 3583.5
0010008 0607 0368.3 3075.2
0010008 1052 1908.5 5868.4
0010008 0437 0340.1 1465.9
0010008 0640 1974.0 7747.5
0010008 0798 1779.6 5615.7
0010008 0589 1952.5 5658.5
0010008 0508 1978.4 1144.0
0010008 1018 1691.3 4703.5
0010008 0701 0865.5 2197.7
0010008 0831 1572.3 7449.7
0010008 0700 0579.5 2328.0
0010010 0105 1184.2 4347.6
0010010 0108 1850.7 1129.3
.....
.....
Over 1,000,000 Lines
.....
.....
7650767 0300 0641.4 2720.7
7650767 0337 0298.4 1518.4
7650767 0256 0306.3 2944.1
7650767 0319 0848.0 1184.5
7650767 0198 1589.5 0399.0
```

Know as *bead-level* data.

Output formats

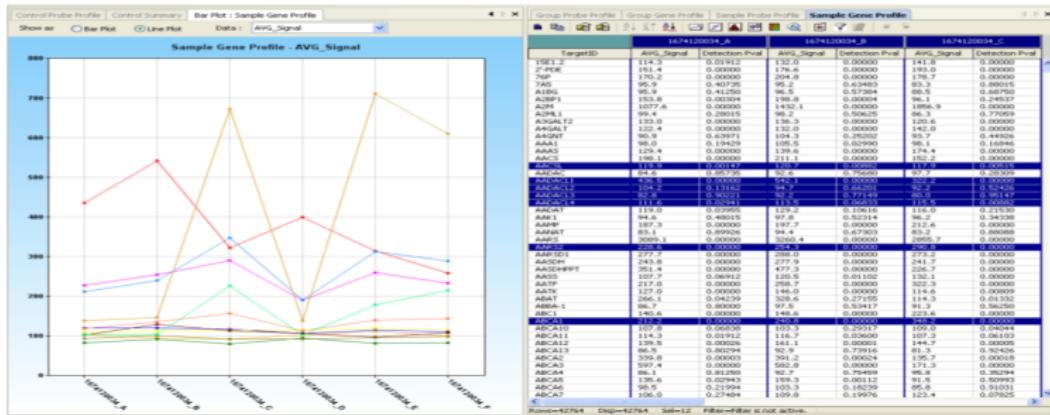
annotation file from Illumina that gives the probe sequences.
Shared for experiments using the same chip-type.

```
##  
  
##      Code          Sequence Symbol  
## 1 7650767 ACGACCTCCTGTTGGAC..... LAYN
```

We will revisit these later...

Summary-level data

Additionally, BeadStudio or GenomeStudio software can be used on the scanned data

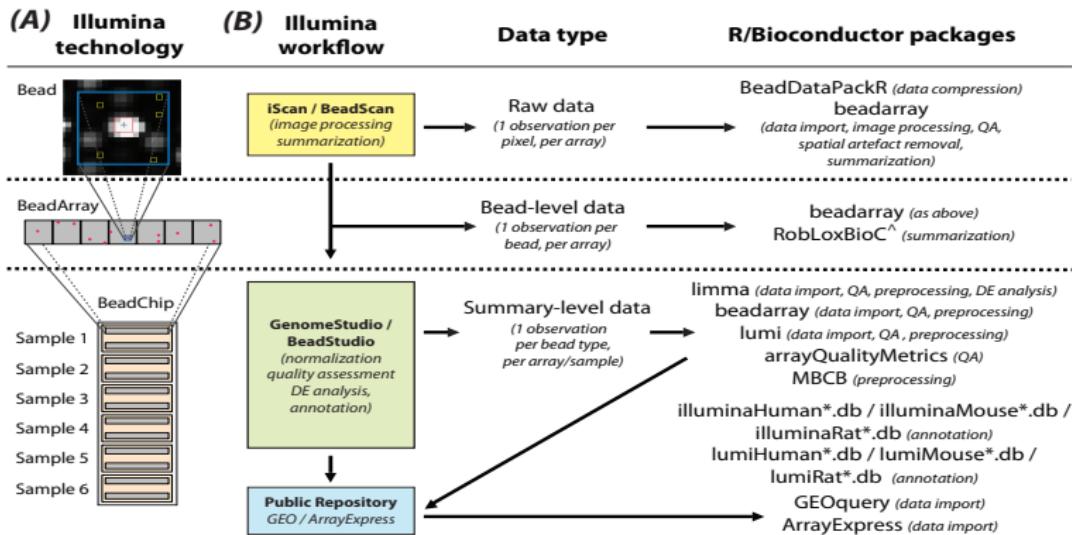


Starting point for analysis

The analyst may have a choice over what data to take as a starting point

- Raw images
- Bead-level data - vary number of observations for a bead-type on each array and pre-computed intensities
- Bead-summary - One value for each bead-type on each array. i.e. an 'Expression Matrix'
- Starting with raw, or bead-level data gives more flexibility in analysis
- Bead-summary data fit in with standard Bioconductor tools

Workflow Overview



From [Ritchie et al](#)

Scanner Metrics

- The scanner can give lots of useful quality control information
- P95 = 95th intensity percentile (x-axis)
- P05 = 5th intensity percentile (y-axis)
- P95:P05 Ratio or signal to noise ratio, red line = ideal

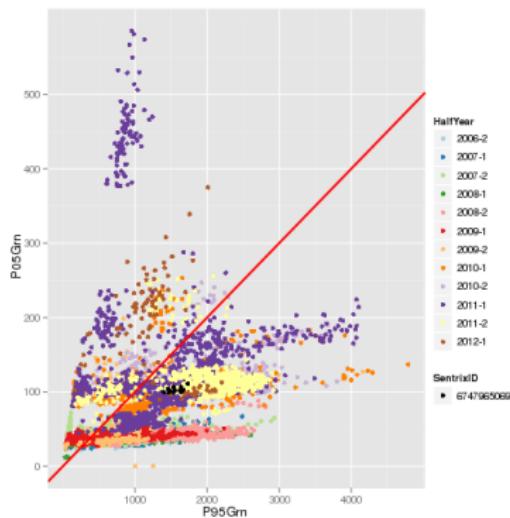
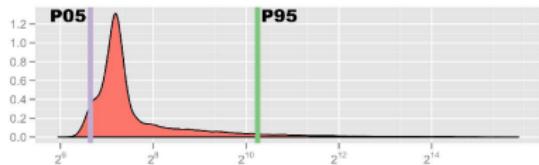
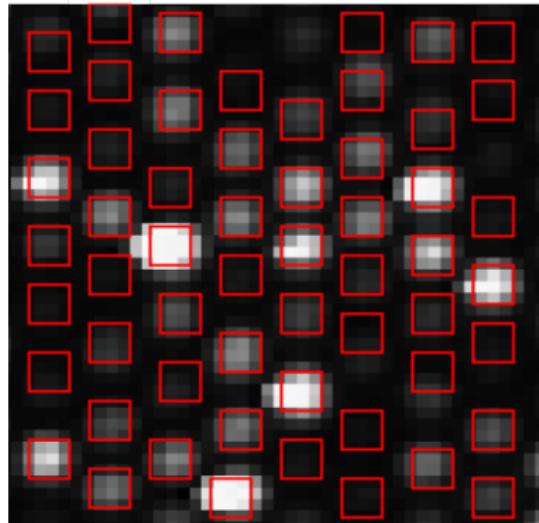


Image Processing

- Calculate foreground & background
- Background correct
- The resulting values are found in the text files
- The steps can be repeated in beadarray
- *Usually we start with these values*

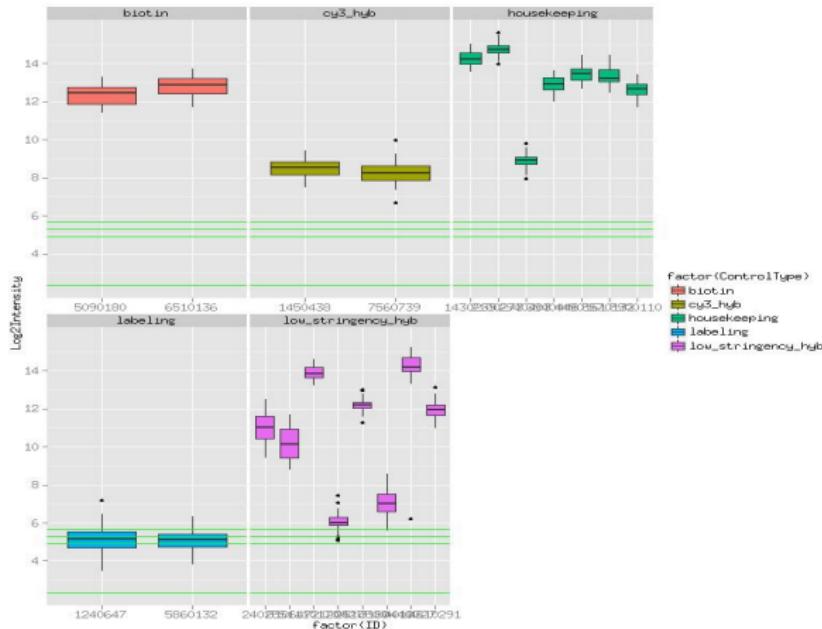
Full, gory, details in [Smith et al](#)



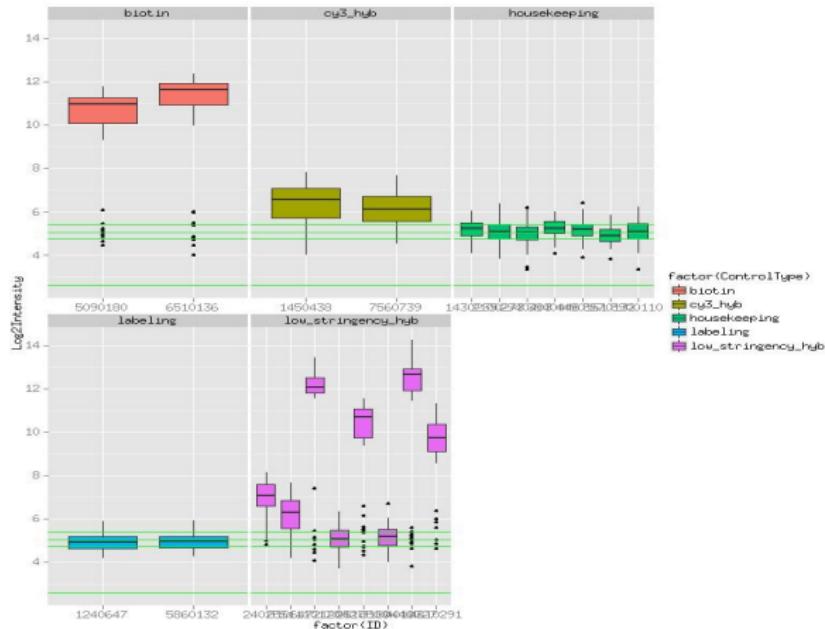
Control Plots

- Illumina design probes for housekeeping genes **Positive controls**
- They also have probes that should not hybridise to the target genome **Negative controls**
- The difference between these controls is a measure of signal level and the potential to detect differential expression
- The negative controls are also used to assess whether a given gene-probe is expressed / detected.

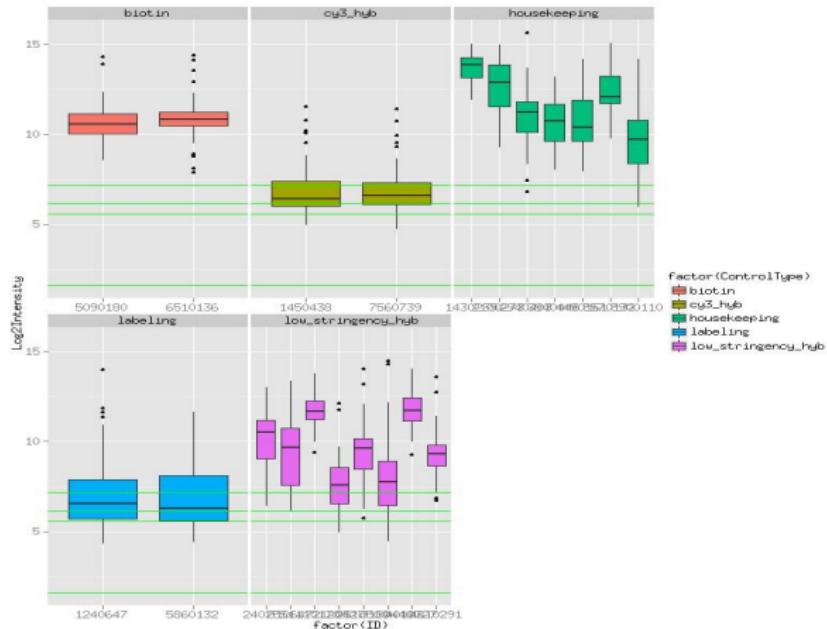
The good..



The bad..

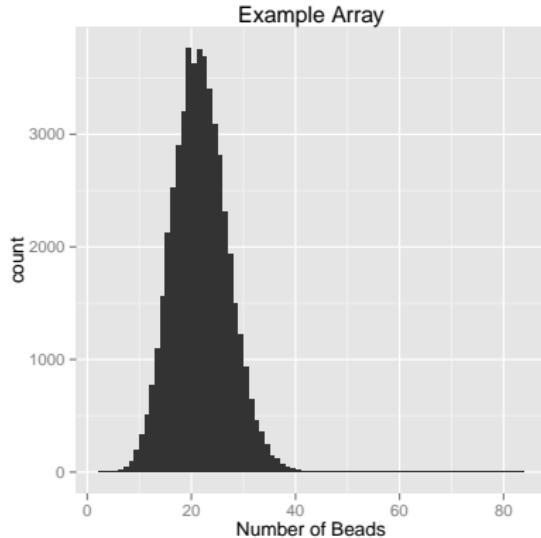


and the ugly..



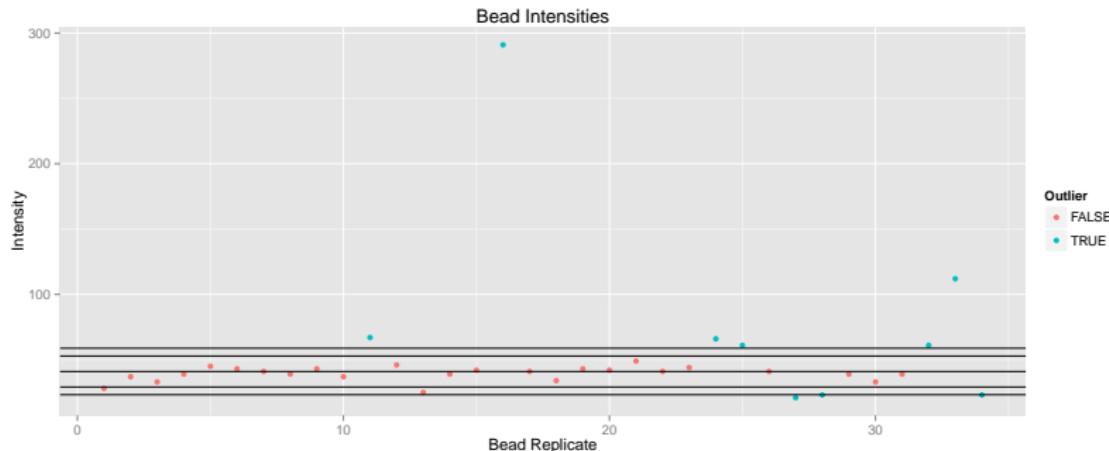
Dealing with multiple beads

- Having many replicates of each bead-type is one of the strengths of the BeadArray platform
- As is random array design
- But ultimately we want to compare one observation for each bead type between arrays



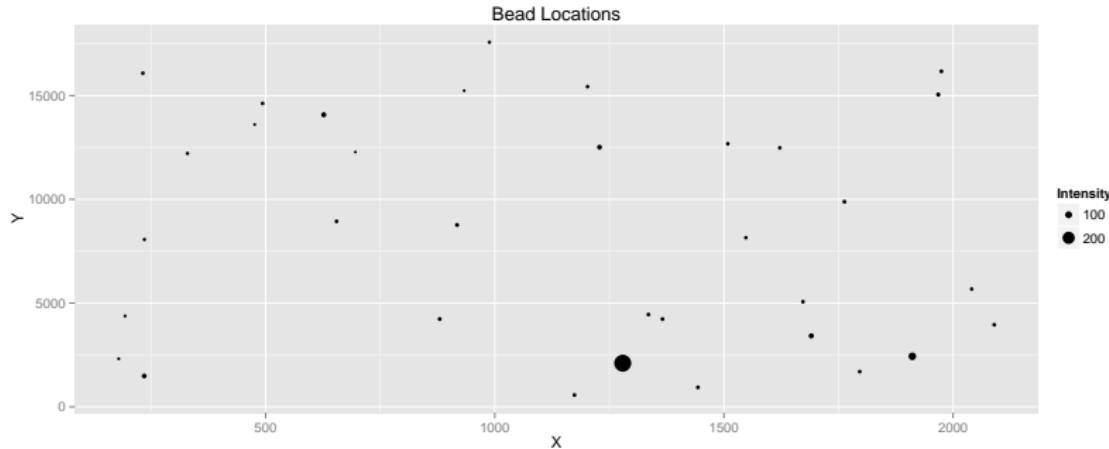
Illumina outlier removal

Illumina use an algorithm for removing unreliable beads using difference from the un-logged median. We prefer to use this method on the \log_2 scale



Illumina outlier removal

Where the outliers are located could also give clues about potential artefacts



Spatial artefacts

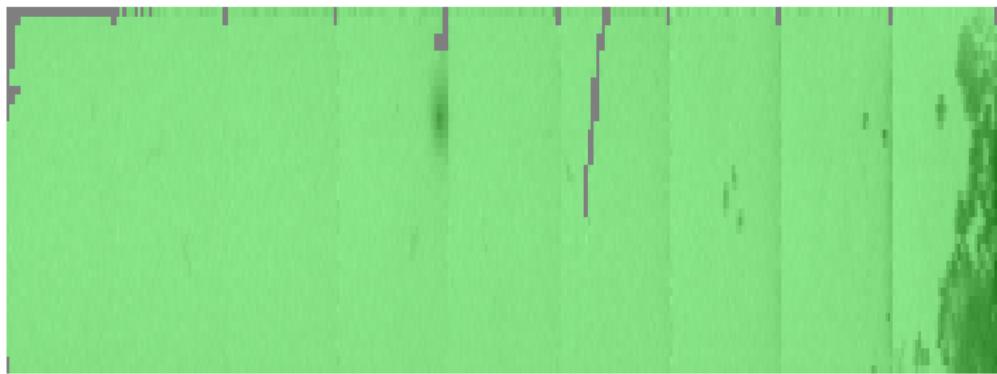
Sometimes obvious artefacts can be apparent on the TIFF image itself



Would you trust any measurements taken on the right of this array?

Tools for assessing spatial artefacts

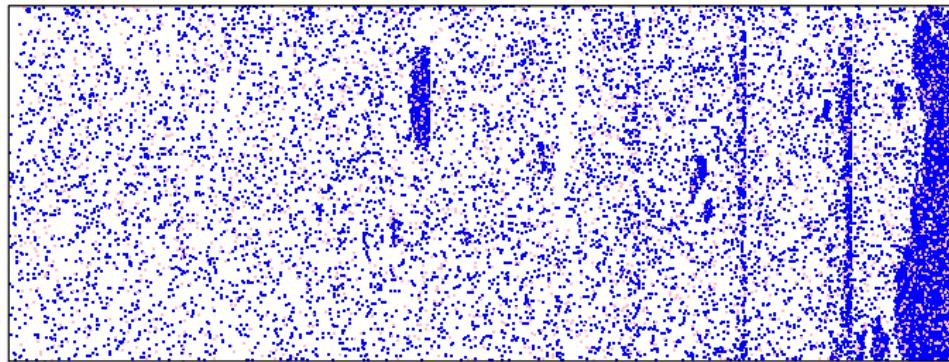
The `imageplot` function can create a false-colour representation of the array surface. Intensities of beads within a window are averaged and displayed.



Tools for assessing spatial artefacts

Outliers for each bead type can be calculated and plotted

```
## 50752 outliers found on the section
```

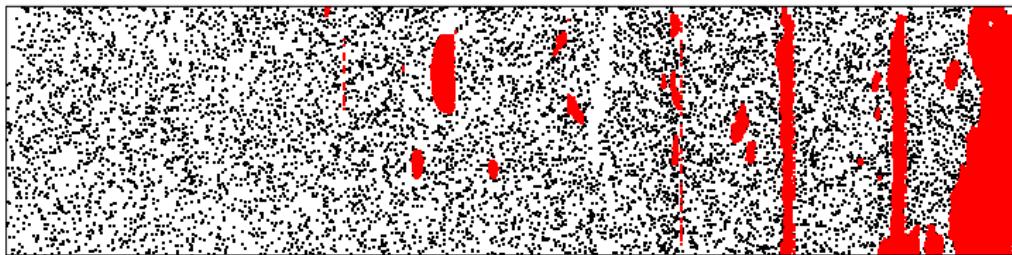


BASH

- The BASH method can be used to mask beads within spatial artefacts
- Adapted from Harshlight method first applied to Affymetrix arrays
- BASH looks for clusters of outliers close together
- Weights are assigned to each bead; 0 = exclude and 1 = include

BASH

4158323231_B_1

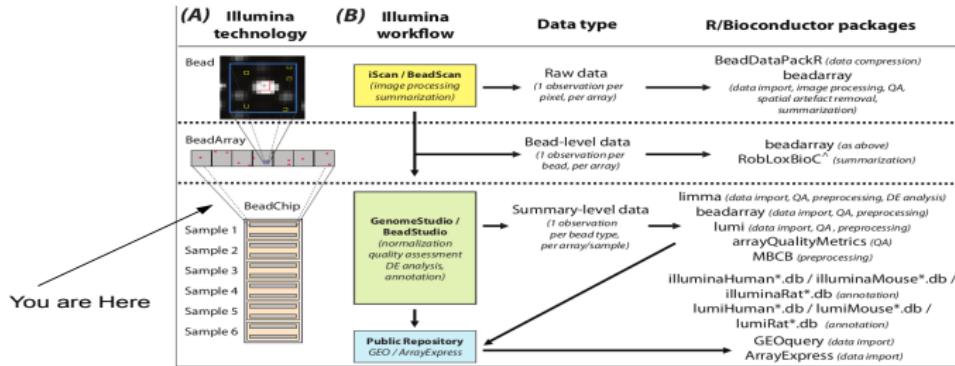


- Spatial investigations only possible with bead-level data
- May be missed as the cause of unreliable data if dealing with summary data
- Persistent manufacture or processing defects may also be overlooked

Producing summary values

For each array...

- Identify spatial artefacts and exclude any beads found within
- Group bead intensities according to Code (corresponding to a bead-type / gene)
- For each Code
 - log₂ transform intensities
 - Calculate mean and standard error
- Repeat for next array

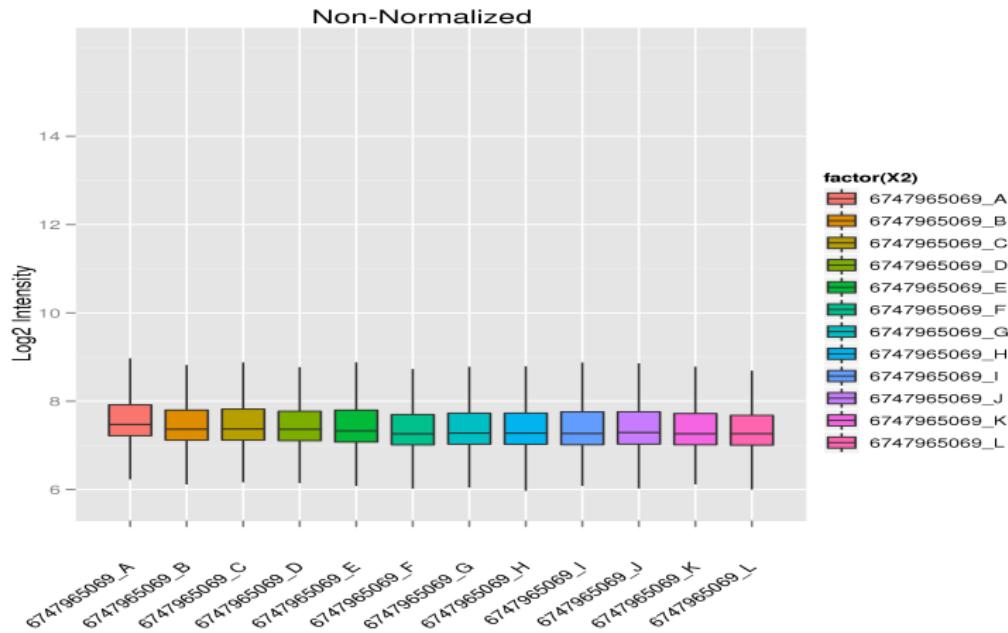


Entry-point for other microarray analysis packages

Starting with summarized data

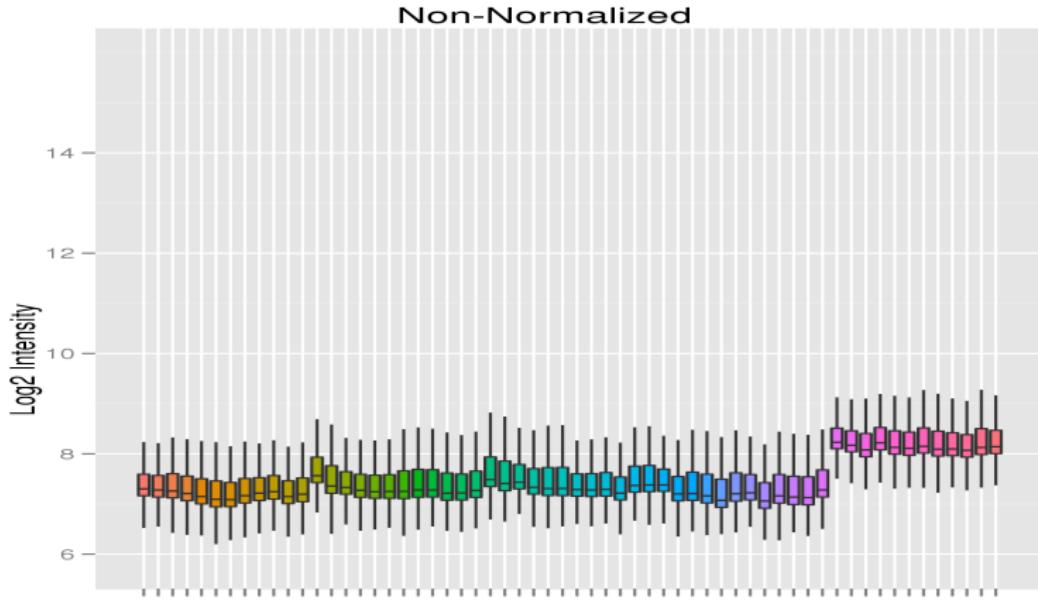
- Data exported from GenomeStudio are already summarized, but un-logged
- Ensure that data are not normalised
- Choose *probe-profile* data
- Summarized data can be read using limma, lumi or beadarray
- **Same analyses can be used on data summarized using beadarray, or exported from GenomeStudio**
- Summarized data can also be obtained from online repositories GEO or ArrayExpress

Quality Control - Summarised beads



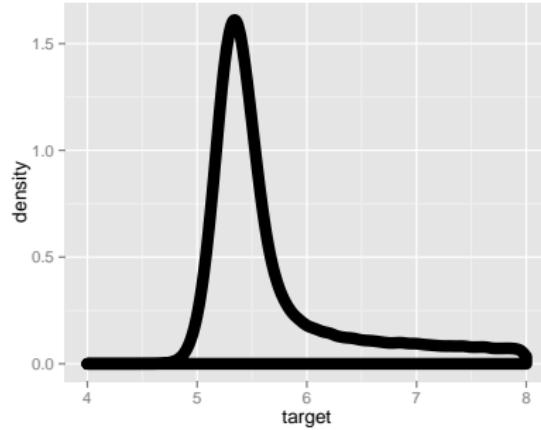
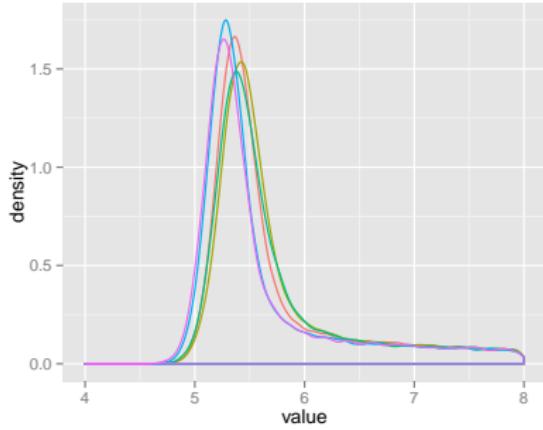
Why normalise?

We want to be measuring *biological* rather than *technical* variation.



Normalisation of Illumina arrays

- Schmid et al review available methods and give guidelines for choosing most-appropriate method.
- The most popular is probably **Quantile normalisation**, or some variation on this.



Quantile Normalization

Consider the following matrix of values to be normalised

```
##      Array1 Array2 Array3
## A      1      3      9
## B      3      4      1
## C      9      2      5
## D      2      1      7
## E      4      9      6
```

Genes A, B, C, D and E measured on three arrays

Quantile Normalization - Step 1

Determine ranks of each column

```
##      Array1 Array2 Array3
## A          1      3      9
## B          3      4      1
## C          9      2      5
## D          2      1      7
## E          4      9      6
```

```
##      Array1 Array2 Array3
## [1,] "Rank1" "Rank3" "Rank5"
## [2,] "Rank3" "Rank4" "Rank1"
## [3,] "Rank5" "Rank2" "Rank2"
## [4,] "Rank2" "Rank1" "Rank4"
## [5,] "Rank4" "Rank5" "Rank3"
```

Quantile Normalization - Step 2

Sort each column Largest...Smallest

Original data

```
##    Array1 Array2 Array3
## A      1      3      9
## B      3      4      1
## C      9      2      5
## D      2      1      7
## E      4      9      6
```

Sorted data

```
##    Array1 Array2 Array3
## [1,]     1      1      1
## [2,]     2      2      5
## [3,]     3      3      6
## [4,]     4      4      7
## [5,]     9      9      9
```

Then calculate target distribution by averaging the sorted rows

```
## Rank1 Rank2 Rank3 Rank4 Rank5
##      1      3      4      5      9
```

Quantile Normalization - Step 3

Go back to the rank matrix

```
##      Array1  Array2  Array3  ##      Array1  Array2  Array3
## [1,] "Rank1" "Rank3" "Rank5" ## [1,] "1"      "Rank3" "Rank5"
## [2,] "Rank3"  "Rank4" "Rank1" ## [2,] "Rank3" "Rank4" "Rank1"
## [3,] "Rank5"  "Rank2" "Rank2" ## [3,] "Rank5" "Rank2" "Rank2"
## [4,] "Rank2"  "Rank1" "Rank4" ## [4,] "Rank2" "Rank1" "Rank4"
## [5,] "Rank4"  "Rank5" "Rank3" ## [5,] "Rank4" "Rank5" "Rank3"
```

Substitute with values from the target distribution

```
## Rank1 Rank2 Rank3 Rank4 Rank5
##     1     3     4     5     9
```

Quantile Normalization - Step 3

Go back to the rank matrix

```
##      Array1  Array2  Array3 ##      Array1  Array2  Array3
## [1,] "1"     "Rank3" "Rank5" ## [1,] "1"     "Rank3" "Rank5"
## [2,] "Rank3"  "Rank4" "Rank1" ## [2,] "Rank3"  "Rank4" "Rank1"
## [3,] "Rank5"  "Rank2" "Rank2" ## [3,] "Rank5"  "Rank2" "Rank2"
## [4,] "Rank2"  "Rank1" "Rank4" ## [4,] "3"     "Rank1" "Rank4"
## [5,] "Rank4"  "Rank5" "Rank3" ## [5,] "Rank4"  "Rank5" "Rank3"
```

Substitute with values from the target distribution

```
## Rank1 Rank2 Rank3 Rank4 Rank5
##     1     3     4     5     9
```

Quantile Normalization - Step 3

Go back to the rank matrix

```
##      Array1  Array2  Array3  ##      Array1  Array2  Array3
## [1,] "1"     "Rank3" "Rank5"  ## [1,] "1"     "Rank3" "Rank5"
## [2,] "Rank3"  "Rank4" "Rank1"  ## [2,] "4"     "Rank4" "Rank1"
## [3,] "Rank5"  "Rank2" "Rank2"  ## [3,] "Rank5" "Rank2" "Rank2"
## [4,] "3"      "Rank1" "Rank4"  ## [4,] "3"     "Rank1" "Rank4"
## [5,] "Rank4"  "Rank5" "Rank3"  ## [5,] "Rank4" "Rank5" "Rank3"
```

Substitute with values from the target distribution

```
## Rank1 Rank2 Rank3 Rank4 Rank5
##     1     3     4     5     9
```

Quantile Normalization - Step 3

Go back to the rank matrix

```
##      Array1  Array2  Array3  ##      Array1  Array2  Array3
## [1,] "1"    "Rank3" "Rank5"  ## [1,] "1"    "Rank3" "Rank5"
## [2,] "4"    "Rank4" "Rank1"  ## [2,] "4"    "Rank4" "Rank1"
## [3,] "Rank5" "Rank2" "Rank2"  ## [3,] "Rank5" "Rank2" "Rank2"
## [4,] "3"    "Rank1" "Rank4"  ## [4,] "3"    "Rank1" "Rank4"
## [5,] "Rank4" "Rank5" "Rank3"  ## [5,] "5"    "Rank5" "Rank3"
```

Substitute with values from the target distribution

```
## Rank1 Rank2 Rank3 Rank4 Rank5
##     1     3     4     5     9
```

Quantile Normalization - Step 3

Go back to the rank matrix

```
##      Array1  Array2  Array3  ##      Array1  Array2  Array3
## [1,] "1"    "Rank3" "Rank5"  ## [1,] "1"    "Rank3" "Rank5"
## [2,] "4"    "Rank4" "Rank1"  ## [2,] "4"    "Rank4" "Rank1"
## [3,] "Rank5" "Rank2" "Rank2"  ## [3,] "9"    "Rank2" "Rank2"
## [4,] "3"    "Rank1" "Rank4"  ## [4,] "3"    "Rank1" "Rank4"
## [5,] "5"    "Rank5" "Rank3"  ## [5,] "5"    "Rank5" "Rank3"
```

Substitute with values from the target distribution

```
## Rank1  Rank2  Rank3  Rank4  Rank5
##     1      3      4      5      9
```

Quantile Normalization - Summary

We then repeat to get the normalized matrix

Original data

```
##   Array1 Array2 Array3
## A     1     3     9
## B     3     4     1
## C     9     2     5
## D     2     1     7
## E     4     9     6
```

Normalized data

```
##   Array1 Array2 Array3
## A     1     4     9
## B     4     5     1
## C     9     3     3
## D     3     1     5
## E     5     9     4
```

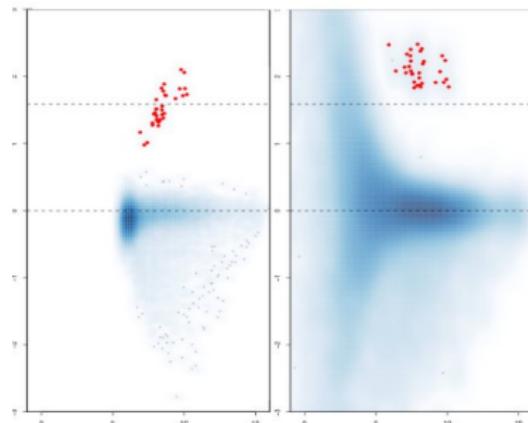
The code

```
df <- data.frame(Array1 = c(1, 3, 9, 2, 4), Array2 = c(3,
 4, 2, 1, 9), Array3 = c(9, 1, 5, 7, 6))
rownames(df) <- LETTERS[1:nrow(df)]
rks <- apply(df, 2, function(x) paste("Rank", rank(x,
ties.method = "min"), sep = ""))
target <- round(rowMeans(apply(df, 2, sort)), 3)
names(target) <- paste("Rank", 1:length(target), sep = "")
for (i in 1:ncol(df)) {
  for (nm in names(target)) {
    rks[, i] <- gsub(nm, target[nm], rks[, i])
  }
}
norm <- as.data.frame(rks)
```

Background correction

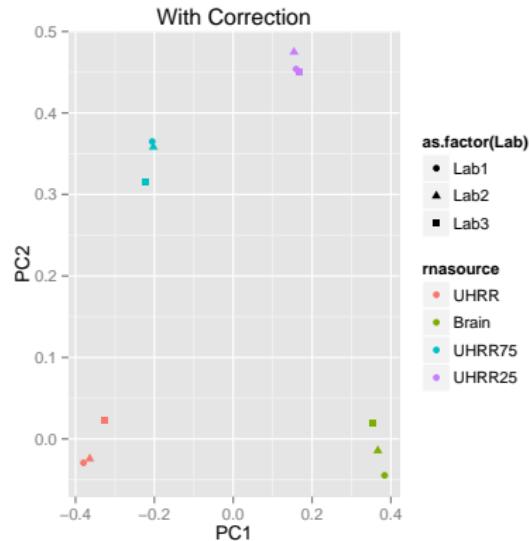
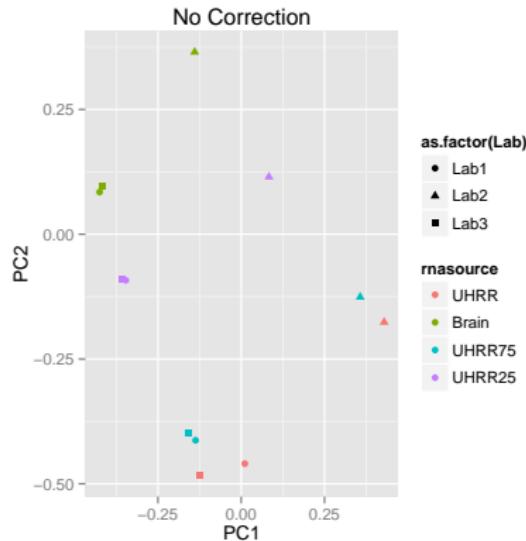
See [Shi et al](#)

- If a gene is not being expression, then its intensity on the array would be zero
- ...but it isn't
- Illumina attempt to measure *non-specific binding* using negative controls
- but they do a bad job (see right)



Batch Effects

Experiment with Brain and Reference RNA hybridised at different labs



See [Leek et al](#) for comprehensive review

Normalisation Options

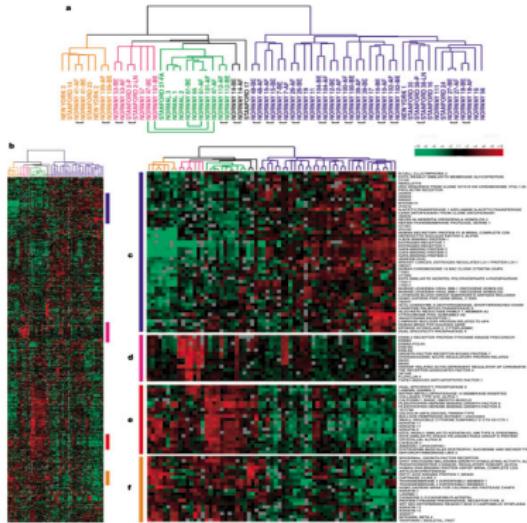
- The [neqc](#) method from limma combines quantile normalisation with a more sophisticated way of dealing with non-specific binding
- Also [vst](#) and [rsn](#) in lumi
- Ideally one would evaluate different methods
- May also need batch effect correction - see [sva](#) package in Bioconductor
- Case-study of batch effects on Illumina data provided by [Kitchen et al](#)

Why annotate probes?

- By default, each probe is assigned a numeric Code (*ArrayAddress*) if analysed at the bead-level, or a manufacturer identifier (*ILMN_*)
- The IDs need to be converted so that we could recognise our favourite genes in the data. e.g. TP53, BRCA1
- Or to relate the findings to biological function
- Is this the end of the story?

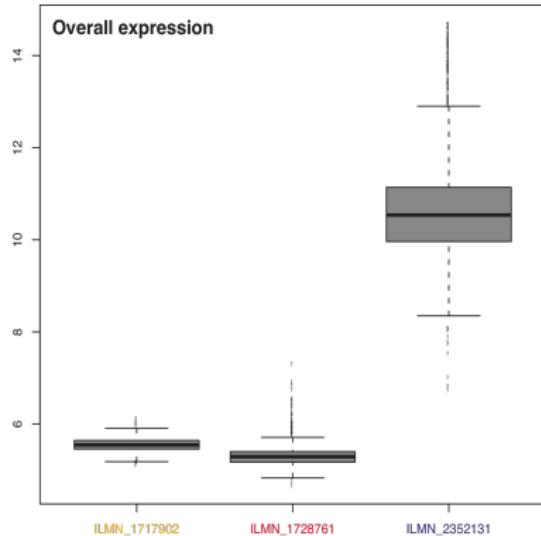
A motivating example

- Starting with Perou et al (right) various papers have classified breast cancer based on microarray profiles. They seem to agree on five subtypes, which have different prognosis and treatment options

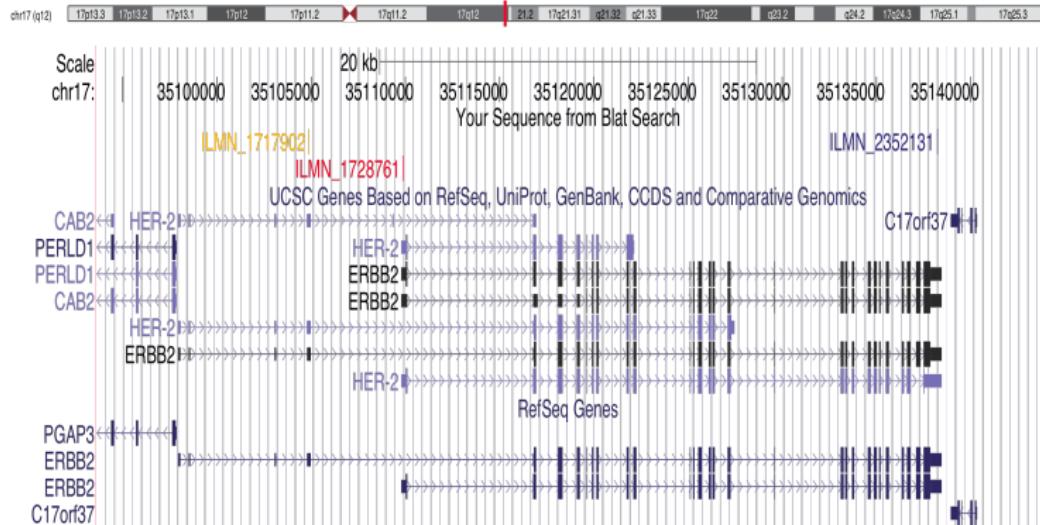


Expression of ERBB2 in a breast cancer study

- The ERBB2 oncogene is important in breast cancer, so we want to be able to measure it accurately
- In the [METABRIC](#) study, three probes for ERBB2 on the microarray
- Simple approach would be to average values for all three probes

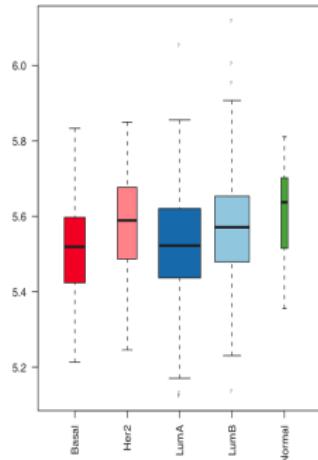


Location of ERBB2 probes

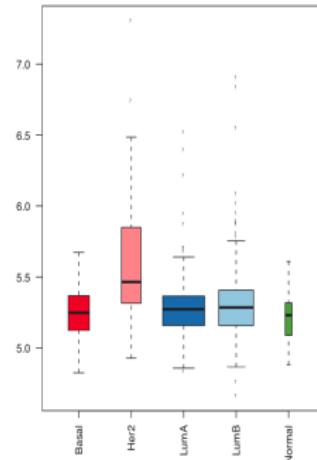


Difference between subtypes

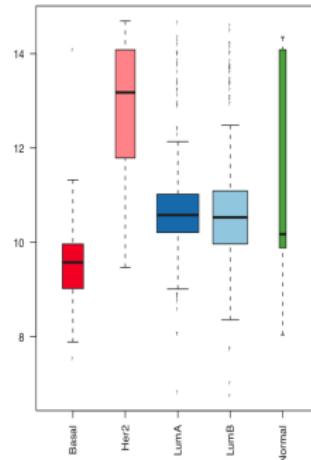
ILMN_1717902 expression in Intrinsic Subtypes



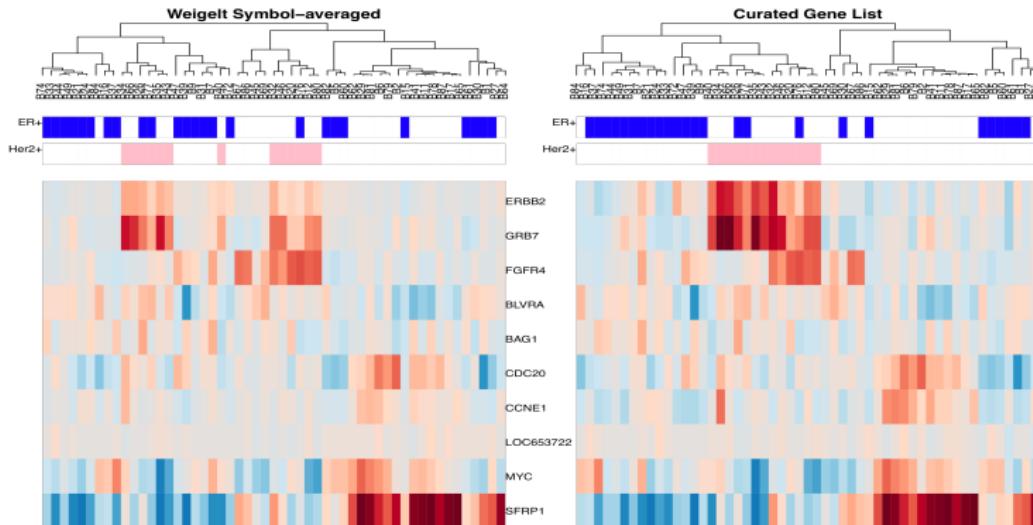
ILMN_1728761 expression in Intrinsic Subtypes



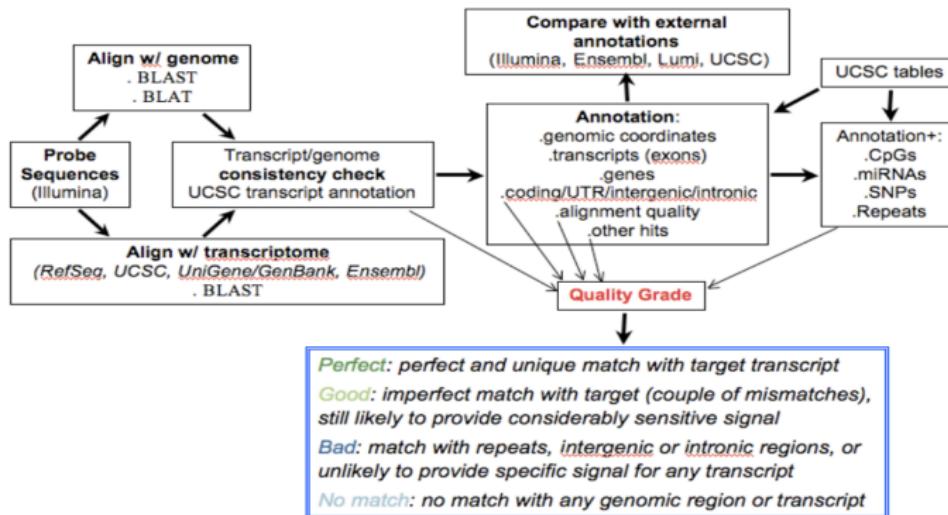
ILMN_2352131 expression in Intrinsic Subtypes



Using external data

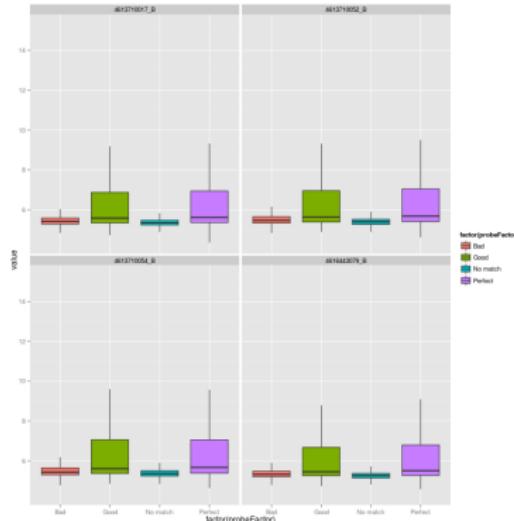


An annotation pipeline



Effect on intensity

- As expected Perfect and Good probes highest intensity
- Less signal for Bad and No match
- Although some exceptions..
- Associations also possible due to SNPs within probes



Annotation conclusions

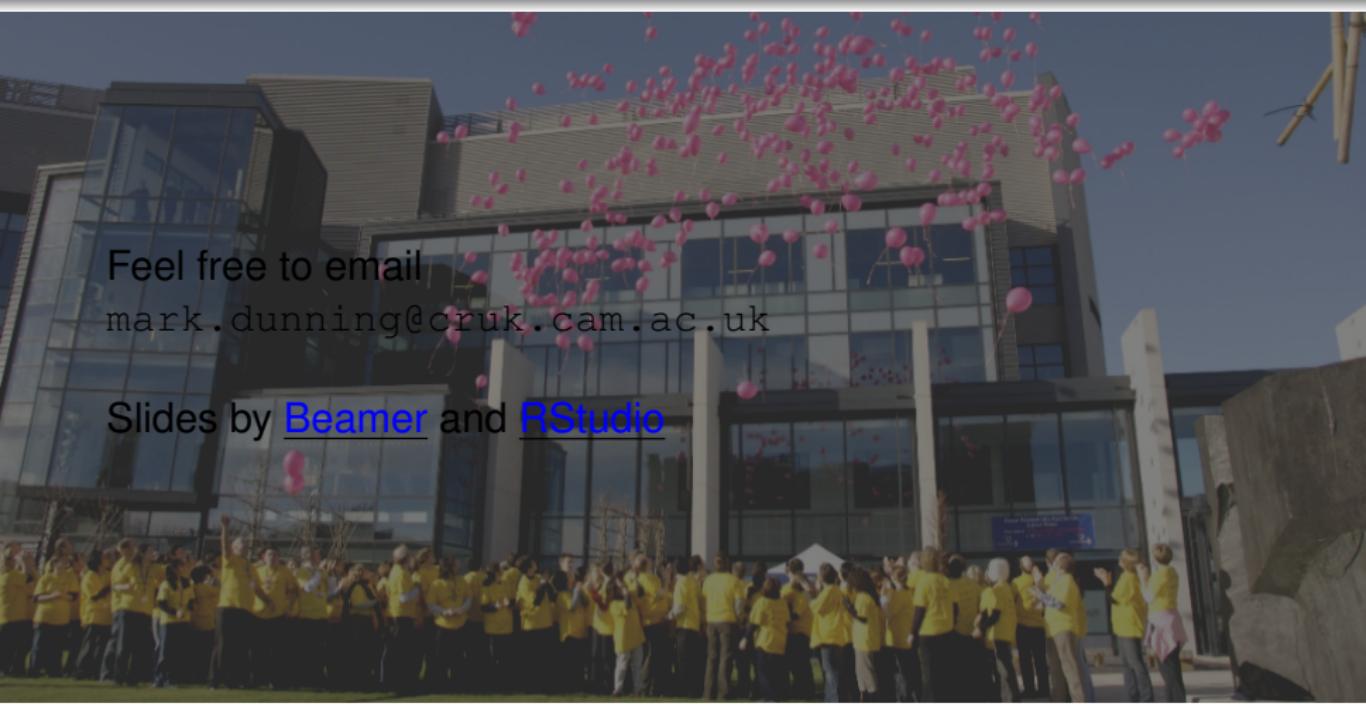
- Annotation quality is known in advance
- Annotation can be used to remove uninformative data for filtering and reducing number of tests performed
- Similar in spirit to removing non-expressed probes
- Surprising results could be explained by annotation problems; missing genes from top hits, or unexpected findings
- If you *must* collapse to the gene-level, consider the annotation whilst doing so. **Please don't just average**

Similar issues for Affymetrix arrays

Where does the annotation come from?

- The annotation slot in the bead-level data is the link to the correct annotation package
- If unsure, the `suggestAnnotation` function will try to advise
- Humanv3 tells beadarray to use `illuminaHumanv3.db` for annotation
- If installed, the `illuminaHumanv3.db` will be loaded
- The control probes can then be identified and IDs converted automatically

The End



Feel free to email
mark.dunning@cruk.cam.ac.uk

Slides by [Beamer](#) and [RStudio](#)