

2.2 - Microarray Data Modeling Lab

Sébastien Lemieux

Elitra Canada Ltd.

Objectives

- to learn the basic principles about microarray experiments.
- to understand the main features seen in microarray data.
- to prepare Java classes to interact with the Golub *et al.* dataset.

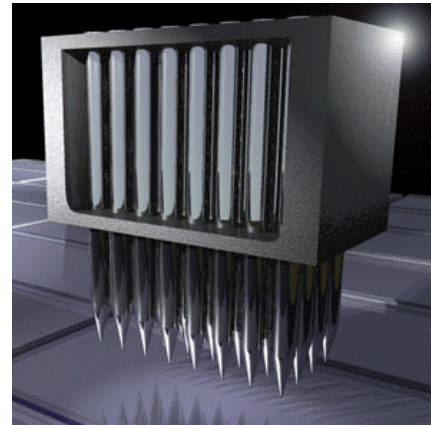
Overview

- Experimental setup.
- Common features of microarray data:
 - Data organization;
 - File formats and databases.
- Golub *et al.* dataset
- Parsing the raw data: Java classes.

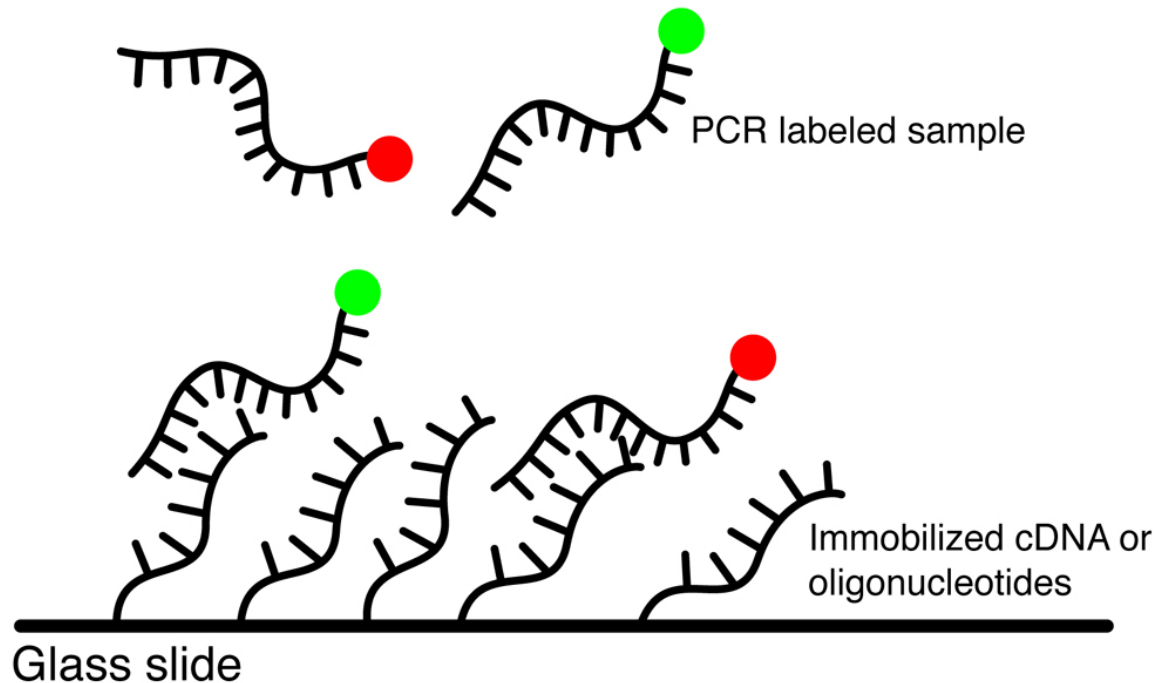
Experimental setup

- Two types of printing:
 - robotically spotted - oligonucleotides or cDNA;
 - photolithography - oligonucleotides (Affymetrix).
- Spotted arrays:
 - two samples are labeled with different fluorescent dyes (Cy3, Cy5) and co-hybridized on the same array.

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.



Hybridization (spotted array)



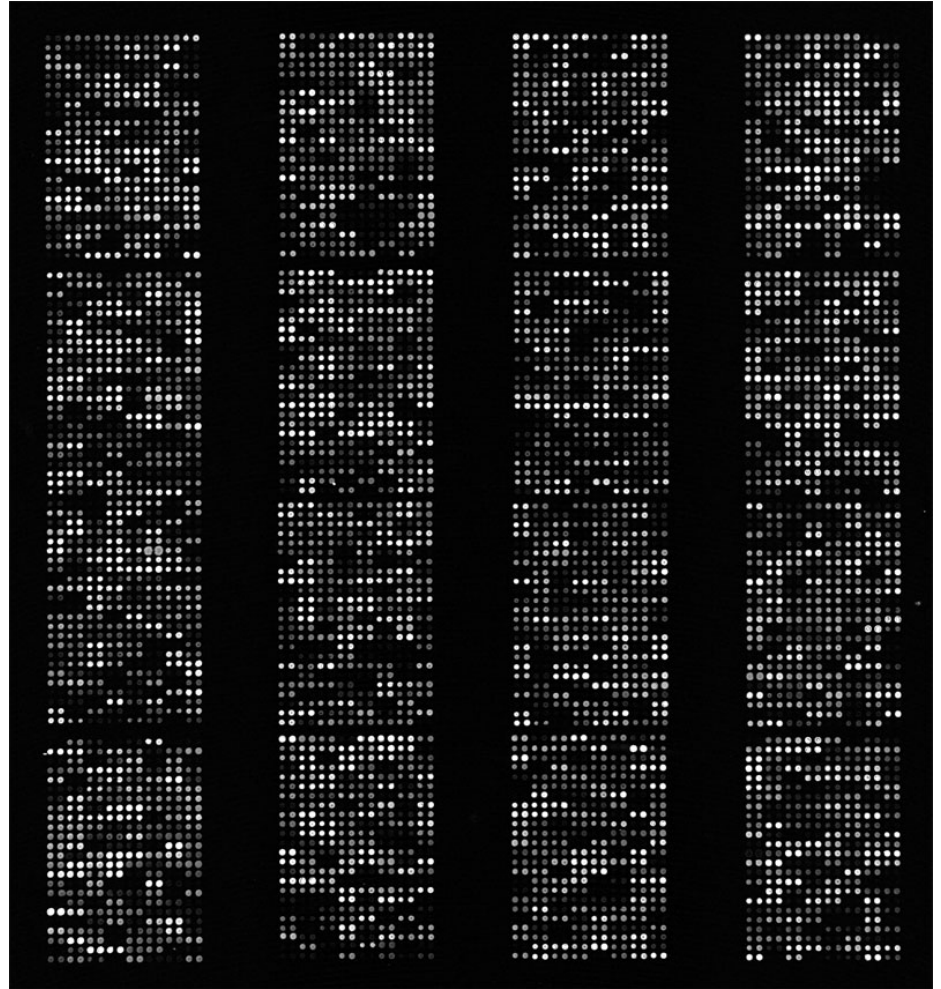
- Competitive hybridization of Cy5 or Cy3-labeled nucleic acids.
- Total amount of Cy5 and Cy3 should be matched.
- Normalization: physical and numerical.

Experimental setup (cont.)

- Scan
 - TIFF 16-bit (0 - 65 536), one image file per dye;
 - Typical size > 10Mb per channel.
- Quantification
 - *Gridding*: identifies the location on the image of each element. Usually done in two step: grid locations followed by spot locations.
 - *Segmentation*: for each spot location, identifies which pixels are part of the spot and which are part of the background.

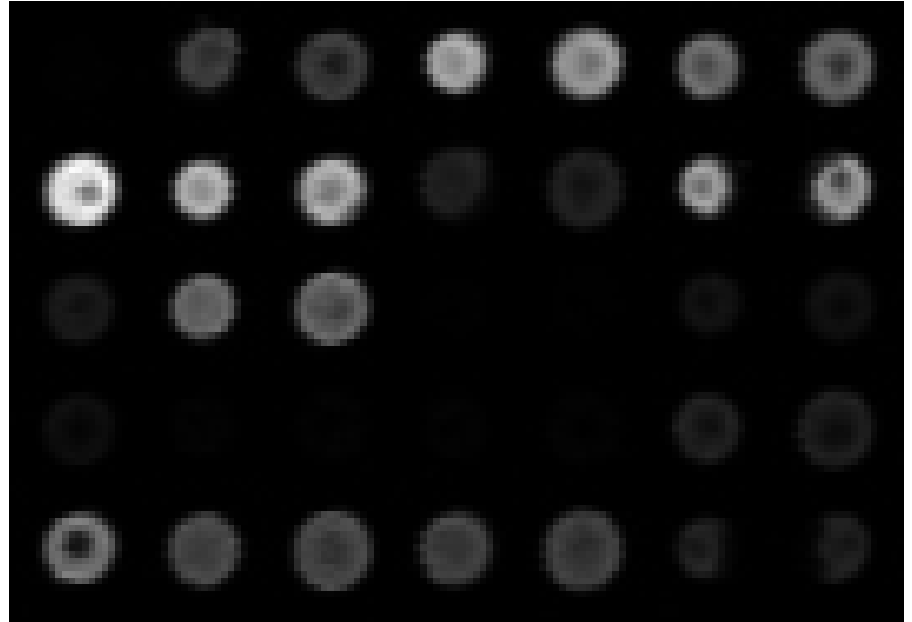
Microarray - the scanned image

- Example of a yeast spotted array.
 - 16 x 16 pins.
 - 15 x 24 subarrays.
 - duplicate spots side-by-side.



Microarray - the scanned image (cont.)

- Target a resolution that will result in spots of ~10 pixels of diameters.
- Too dark spots are unusable, but saturated spots are also unusable.
- The donut shape is a frequent artifact of spotted arrays.
 - Depending on the quantification software it may significantly reduce the quality of the data acquired.



Features of microarray data

- On the experimental side:
 - One dataset contains one or more conditions (cell types, media, time, temperature, etc.);
 - For each condition one or more hybridization can be obtained (replicates, fluor-reverse);
 - Fluor-reverse hybridization is required to account for dye-specific preferential amplification of some probes.
 - The number of replicate will determine the sensitivity of the experiment.
 - Each hybridization is one (affy) or two (spotted) channels;
 - For spotted arrays, the two channels correspond to different dyes (Cy3, Cy5) and are typically obtained from two different conditions.

Features of microarray data *(cont.)*

- On the array (spotted):
 - Multiple grid corresponding to each spotting pin;
 - Each grid contains several probes (cDNA or oligo);
 - Each probe can be replicated (1, 2 or 3 times).
- Keeping track of this structure may help data analysis:
 - pin-specific normalization.

Features of microarray data *(cont.)*

- Published information is typically a matrix presenting for each pair condition / probe the final quantification used:
 - log-ratio for spotted arrays;
 - background corrected intensity for affy.
- More complex to represent:
 - Information about the conditions;
 - Information about the array construction.

Data structures

- A solution:
 - MIAME: Minimum information about a microarray experiment
(http://www.mged.org/Workgroups/MIAME/miame_1.1.html)
 - MAGE-ML: microarray data exchange format adopted as a standard for gene expression by the OMG and supported by MGED
(<http://www.mged.org/Workgroups/MAGE/mage-ml.html>)
 - MGED: Microarray Gene Expression Database
(<http://www.mged.org/>)
- For the actual data:
 - Good old tab-delimited format is still widely used.

Golub *et al.* dataset

- Two datasets:
 - initial (training, 38 samples);
 - independent (test, 34 samples).
- Intensity values have been re-scaled such that overall intensities for each chip are equivalent.
 - Linear regression using intensities of all genes in both the first sample and each other.
 - $1 / \text{slope}$ of the linear regression becomes the re-scaling factor.
 - File: `table_ALL_AML_rfactors.txt`
- Two files:
 - `table_ALL_AML_samples.txt`: contains information about each sample (the condition);
 - `data_set_ALL_AML_train.txt`: contains information about each gene (probe) along with the matrix of quantified intensities.

Golub *et al.* dataset (cont.)

- From the paper's website:
 - <http://tinyurl.com/24wb4>
- Retrieve the following files:
 - Samples table (text);
 - Train dataset (text).
- And, take a look at the Paper (PDF).

Golub *et al.* dataset (cont.)

- Three data structures:
 - Sample: String name;
 - Gene: String description; String accession;
 - Expression: double value;
 - Visualize it as a matrix where the sample information describe each row and the gene information describe each column.
- We will adopt a simple structure:
 - The sample will contain the expression data as a `HashMap` using the `gene_id`.

Sample information

LEUKEMIA	SAMPLES	INFORMATION						
Samples	ALL/AML (if	BM/PB ALL)	T/B-cell (if	FAB AML)	Date/Gender Response	%	Blasts	Treatment
INITIAL	SET							
	1 ALL	BM	B-cell		9/4/96 -	M		1 DFCI
	2 ALL	BM	T-cell		-	M		0.41 DFCI
	3 ALL	BM	T-cell		-	M		0.87 DFCI

- 9 header lines.
- tab / space usage is erratic.
- We will first store the information as a single String...

Expression data file

Gene Description	Gene Accession	Nurr	1	call	2	call	3	call	4	call
AFFX-BioB-5_at (end AFFX-BioB-5_at			-214	A	-139	A	-76	A	-135	A
AFFX-BioB-M_at (enc AFFX-BioB-M_at			-153	A	-73	A	-49	A	-114	A
AFFX-BioB-3_at (end AFFX-BioB-3_at			-58	A	-1	A	-307	A	265	A
AFFX-BioC-5_at (enc AFFX-BioC-5_at			88	A	283	A	309	A	12	A
AFFX-BioC-3_at (enc AFFX-BioC-3_at			-295	A	-264	A	-376	A	-419	A
AFFX-BioDn-5_at (er AFFX-BioDn-5_at			-558	A	-400	A	-650	A	-585	A
AFFX-BioDn-3_at (er AFFX-BioDn-3_at			199	A	-330	A	33	A	158	A
AFFX-CreX-5_at (enc AFFX-CreX-5_at			-176	A	-168	A	-367	A	-253	A
AFFX-CreX-3_at (enc AFFX-CreX-3_at			252	A	101	A	206	A	49	A
AFFX-BioB-5_st (end AFFX-BioB-5_st			206	A	74	A	-215	A	31	A
AFFX-BioB-M_st (enc AFFX-BioB-M_st			-41	A	19	A	19	A	363	A
AFFX-BioB-3_st (end AFFX-BioB-3_st			-831	A	-743	A	-1135	A	-934	A
AFFX-BioC-5_st (end AFFX-BioC-5_st			-653	A	-239	A	-962	A	-577	A
AFFX-BioC-3_st (end AFFX-BioC-3_st			-462	A	-83	A	-232	A	-214	A
AFFX-BioDn-5_st (er AFFX-BioDn-5_st			75	A	182	A	208	A	142	A

- 1 header line.
- One tab per column separation.
- One row per gene:
 - We want to have one object per sample, since we will want to operate mainly on the samples.
- Genes are identified by a description and an accession. We will merge the two as a `String` for now. And we will ignore the absent / present call.

Overall architecture

- LoadData.java: main program to parse and display the data files. Uses a GolubParser to obtain an ArrayList of GolubSample.
- GolubParser.java: takes care of parsing the tab-delimited files published as part of the Golub *et al.* dataset.
- GolubSample.java: handles data for one condition. Will be extended during lab 4.1.

Final structure for data

- Data members for the `GolubSample` java class:

```
public class GolubSample {  
  
    private String sampleName;  
    private HashMap geneExpressionMap;  
  
    public GolubSample(String sampleName,  
                        HashMap geneExpressionMap) {  
        this.sampleName = sampleName;  
        this.geneExpressionMap = new HashMap(geneExpressionMap);  
    }  
  
    ...  
}
```

- `sampleName` will contain information about the sample;
- `expressionMap` will hold a `HashMap` using a `gene_id` as the key.

LoadData

```
public class LoadData {  
  
    public static void main (String[] args)  
        throws Exception {  
  
        GolubParser p = new GolubParser ("table_ALL_AML_samples.txt",  
                                         "data_set_ALL_AML_train.txt");  
        ArrayList allSamples = p.getGolubSamples ();  
  
        ((GolubSample)allSamples.get (0)).print (System.out);  
  
    }  
}
```

- Load the data from the published files and create the `GolubSample` structures.

Sample information (cont.)

```
public class GolubParser {  
  
    private ArrayList sampleInfo;  
    private ArrayList geneInfo;  
    private HashMap sampleData;  
  
    public GolubParser(String sampleFileName,  
                       String dataFileName) {  
        this.sampleInfo = new ArrayList ();  
        this.geneInfo = new ArrayList ();  
        this.sampleData = new HashMap();  
  
        try {  
            this.parseSampleFile (sampleFileName);  
            this.parseDataFile (dataFileName);  
        } catch (FileNotFoundException e) {  
            System.err.println ("Problem opening a file...");  
        }  
    }  
}
```

- The data is parsed in two steps:
 - parseSampleFile: Load information about the samples.
 - parseDataFile: Load information about the genes and their expression values for the different samples.

A simple tokenizer...

```
private ArrayList splitTokens (String line, String sep) {
    ArrayList l= new ArrayList ();

    int last = -1;
    for (int i = 0; i < line.length (); i++) {
        char c = line.charAt (i);
        boolean is_sep = false;
        for (int j = 0; j < sep.length (); j++) {
            if (c == sep.charAt (j)) {
                is_sep = true;
                break;
            }
        }
        if (is_sep) {
            if (last >= 0) {
                l.add (line.substring (last, i));
                last = -1;
            }
        } else if (last < 0) last = i;
    }

    return l;
}
```

- Java 1.4 provides `String.split ()` for that purpose...

Parsing sample info

```
private void parseSampleFile (String sampleFileName)
    throws FileNotFoundException {
    BufferedReader r = new BufferedReader(new FileReader (sampleFileName));

    try {
        for (int i = 0; i < 9; i++) { String skip = r.readLine (); }

        while (r.ready ()) {
            String line = r.readLine ();
            ArrayList tokens = splitTokens (line, "\t");
            if (tokens.size () == 0) break;

            String sampleName = tokens.toString ();
            Integer sample_id = new Integer (sampleInfo.size ());
            sampleData.put (sample_id, new HashMap ());
            sampleInfo.add (sampleName);
        }
    } catch (IOException e) {
        System.out.println ("Done.");
    }

    System.out.println ("Loaded " + sampleInfo.size () + " sample info.");
}
```

- Expression data HashMap are inserted in sampleData as empty HashMap.

Parsing expression data file

```
private void parseDataFile (String dataFileName)
    throws FileNotFoundException {
    int count = 0;

    BufferedReader r = new BufferedReader (new FileReader (dataFileName));
    try {
        for (int i = 0; i < 1; i++) { String skip = r.readLine (); }

        while (r.ready ()) {
            String line = r.readLine ();
            ArrayList tokens = splitTokens (line, "\t");
            if (tokens.size () == 0) continue;

            String geneName = tokens.subList (0, 2).toString ();
            Integer gene_id = new Integer (geneInfo.size ());
            geneInfo.add (geneName);

            for (int i = 2; i < tokens.size (); i += 2) {
                Double exp = new Double ((String)tokens.get (i));
                Integer sample_id = new Integer ((i - 2) / 2);

                ((HashMap)sampleData.get (sample_id)).put (gene_id, exp);

                count++;
            }
        }
    } catch (IOException e) {
        System.out.println ("Done.");
    }
    System.out.println ("Loaded " + count + " expression values.");
}
```

Parsing expression data file *(cont.)*

```
String geneName = tokens.subList (0, 2).toString ();
Integer gene_id = new Integer (geneInfo.size ());
geneInfo.add (geneName);

for (int i = 2; i < tokens.size (); i += 2) {
    Double exp = new Double ((String)tokens.get (i));
    Integer sample_id = new Integer ((i - 2) / 2);

    ((HashMap)sampleData.get (sample_id)).put (gene_id, exp);

    count++;
}
```

Gene Description	Gene Accession Num	1	call	2	call	3	call	4	call
AFFX-BioB-5_at (end A	AFFX-BioB-5_at	-214	A	-139	A	-76	A	-135	A
AFFX-BioB-M_at (enc	AFFX-BioB-M_at	-153	A	-73	A	-49	A	-114	A
AFFX-BioB-3_at (end	AFFX-BioB-3_at	-58	A	-1	A	-307	A	265	A
AFFX-BioC-5_at (end	AFFX-BioC-5_at	88	A	283	A	309	A	12	A
AFFX-BioC-3_at (end	AFFX-BioC-3_at	-295	A	-264	A	-376	A	-419	A
AFFX-BioDn-5_at (er	AFFX-BioDn-5_at	-558	A	-400	A	-650	A	-585	A
AFFX-BioDn-3_at (er	AFFX-BioDn-3_at	199	A	-330	A	33	A	158	A
AFFX-CreX-5_at (enc	AFFX-CreX-5_at	-176	A	-168	A	-367	A	-253	A
AFFX-CreX-3_at (enc	AFFX-CreX-3_at	252	A	101	A	206	A	49	A
AFFX-BioB-5_st (end	AFFX-BioB-5_st	206	A	74	A	-215	A	31	A
AFFX-BioB-M_st (enc	AFFX-BioB-M_st	-41	A	19	A	19	A	363	A
AFFX-BioB-3_st (end	AFFX-BioB-3_st	-831	A	-743	A	-1135	A	-934	A
AFFX-BioC-5_st (end	AFFX-BioC-5_st	-653	A	-239	A	-962	A	-577	A
AFFX-BioC-3_st (end	AFFX-BioC-3_st	-462	A	-83	A	-232	A	-214	A
AFFX-BioDn-5_st (er	AFFX-BioDn-5_st	75	A	182	A	208	A	142	A

Creating the GolubSample

```
public ArrayList getGolubSamples () {
    ArrayList l= new ArrayList ();

    for (int i = 0; i < sampleInfo.size (); i++) {
        GolubSample sample = new GolubSample (
            (String)sampleInfo.get (i),
            (HashMap)sampleData.get (new Integer (i)));
        l.add (sample);
    }
    return l;
}
```

- Extract the expression data from the parse and return an ArrayList of GolubSample.

Going further...

- Create `GoLubGene` to hold information about the genes, keeping the name and accession separate.
- In `GoLubSample`, keep in separate fields the type of cancer ALL or AML and the type of cell used.
- Download the file “Test dataset (text)” and modify the parser to load both the Test and Training dataset. Add a field in `GoLubSample` to distinguish them.
- *What structure would allow direct manipulation of values related to a sample or values related to a gene?*

References

- Y. F. Leung and D. Cavalieri, Fundamentals of cDNA microarray data analysis, *TRENDS in Genetics*, **19**:649-659.
- T. R. Golub, et al., Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring, *Science*, **286**:531-537.