Vrije Universiteit Brussel

Faculty of Science and Bio-Engineering Sciences
Department of Computer Science

Analyzing and Benchmarking Genomic Preprocessing and Batch Effect Removal Methods in Big Data Infrastructure

Graduation thesis submitted in partial fulfillment of the requirements for
the degree of Master of Science in Applied Science and Engineering: Computer Science

Quentin De Clerck

Promotors: Prof. Dr. Ann Nowé
Dr. Alain Coletta
Advisor:

August, 2014

Vrije Universiteit Brussel

Faculteit Wetenschappen en Bio-ingenieurswetenschappen
Vakgroep Computerwetenschappen

Analyseren en Benchmarken van Genomische Preprocessing en Batch Effect Verwijdering methoden in Big Data infrastructuur

Proefschrift ingediend met het oog op het behalen van de graad van
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Quentin De Clerck

Promotors: Prof. Dr. Ann Nowé
Dr. Alain Coletta
Advisor:

Augustus, 2014

Abstract

Genomics lies at the crossroad of the genetics, biomedical sciences and information technologies. It holds great promises for the future of medicine because it has the potential of discovering the genetic origins of thousands of diseases. The principal challenge faced in genomics is that people having biomedical expertise often do not have access to the data, making it impossible for them to interpret results. This bottleneck is mainly due to the IT component of genomics. The solution is to enable those experts to access and interpret the information enclosed in the ensemble of genomes by providing them with tools that can be easily manipulated by persons without a computer science background. In this master thesis we focus on the integration of genomic data to such a tool. We enrich the tool by adding preprocessing methods enabling the analysis of more genomic data. We consequently use the data in an analysis and conclude that the integrated methods offer an interesting perspectives for the combination of different genomic data types.

Samenvatting

Genomica ligt op de kruising van de genetica, de biomedische wetenschappen en de informatietechnologiëen. Het is een veelbelovend vakgebied dat een enorm impact kan hebben op geneeskunde door zijn potentieel om genetische oorzaken te ontdekken voor duizenden ziektes. De grootste uitdaging voor de genomica is dat de mensen die de biomedische expertise hebben om data te interpreteren weinig toegang tot die data hebben. De oorzaak van dit probleem ligt voornamelijk bij de IT component van de genomica. Een oplossing zou zijn om de biomedische experts de toegang te geven tot de informatie omvat in de verzameling van genomen aan de hand van hulpmiddelen die gemakkelijk hanteerbaar zijn voor mensen die geen computer wetenschappelijke kennis hebben. In deze masterproef besteden we onze aandacht aan het integreren van genomische data in dergelijk hulpmiddel. We voegen preprocessing methoden toe waardoor het analyseren van meer genomische data mogelijk wordt. Vervolgens gebruiken we deze data in een analyse en concluderen we dat de toegevoegde methoden interessante perspectieven aanbieden voor het combineren van verschillende genomische datatypes.

Acknowledgement

This master thesis would not have been possible without the input and support of some persons.

Alain Coletta proposed a year ago the project of working with Big Data in Hadoop. This is fora student an unique experience and I am grateful to him for that. I am also grateful for his input and ideas during the whole project. I also would like to thank Ann Nowé for supporting the idea of this master thesis and promoting it. Her guidance in the analysis part of this thesis has been particularly appreciated.

I also appreciated the moments working at InSilico DB with the whole InSilico Team: David S., David W., Robin, Jaro and Nicolas. I would like to thank them all and particularly David S. and Nicolas. David S. because he was one of my advisors for my bachelor proof and offered me the possibility to work for InSilico DB as jobstudent two years ago. This master thesis would never have existed without that. Nicolas for enduring my tons of questions about the biological and bioinformatics background.

During past year I communicated with different researchers for informations about their work. Therefore I want to thank Steve Piccolo for his input on the integration of his SCAN and UPC methods to the InSilico DB. Another researcher I would like to thank is John Farrell. He provided me with the simple trick of using simple scripts for enabling BWA and Bowtie2 to work using Hadoop Streaming.

Furthermore, I would like to thank the persons I have the honor to call my best friends: David (I know a lot of Davids), Mathijs, Kenny and Elke. Thank you for

all those awesome moments. We know each others for five awesome years now and I know that our friendship will last after graduation.

Finally, I want to thank the persons who are the most dear to me. My parents who always stood behind me and providing me with everything I needed. I know I cannot thank them enough for all they did for me, but anyway I thank both you. Of course, I could not end otherwise than by thanking my sister Marine for being who she is, the best sister ever.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Genomics, is an established discipline of genetics that aids from Bioinformatics to infer biological knowledge by sequencing, assembly and analysis of part or whole genome. The amount of publicly available genomic data is incredible after all these years of study. Microarray is a well-established technology for genomic data but since a few years new technologies emerge and become more and more popular like RNA-Sequencing. The problem is that all these years people invested in microarray data and it is now unthinkable to throw this data away. Therefore scientists are now developing methods for combining those different data types.

Another but related problem is that the data is not often accessible to the people having the expertise for interpreting the biological results. Those people do not have the necessary background for being able to organize such amount of data. Some tools aim at making data available in a consistent and comprehensive way. One of these tools has been developed by a spin-off company from the VUB and ULB named InSilico DB. InSilico DB consists of a web-based genomic dataset hub (Coletta et al., 2012) that enables users to manage lots of genomic data in a user-friendly way. They have organised around 200.000 samples from the public domain. The datasets have biological curations and have been preprocessed using a consistent methodology enabling users to perform immediately analysis using the data.

InSilico DB supports various data type including microarray and RNA-Seq. These data types are preprocessed in different ways. Microarray data is currently preprocessed with a robust method using a training dataset as reference for the normalization of data. The downside of this method is its unavailability for a large number of microarray datasets provided by InSilico DB. The preprocessing of RNA-Seq relies mainly on the alignment of observed read sequences to the reference genome. The alignment can be very time consuming because of the Big Data generated from RNA-Seq experiments. In this master thesis we enrich the tools provided by InSilico DB by integrating additional preprocessing techniques for making more data available and by using a Big Data framework for speeding up the preprocessing of RNA-Seq data.

1.1 Objective of the master-thesis

InSilico DB preprocesses its microarray datasets using the frozen Robust Multi-array Analysis (fRMA)(McCall et al., 2010) preprocessing method. FRMA normalizes the microarray data making the combination of different datasets in one analysis possible. But the preprocessing needs a precomputed dataset for being able to normalize the microarray correctly. The drawback of the fRMA method resides in the fact that the precomputed data can only be obtained by making a reference dataset of biological diverse samples. Obtaining a large quantity of biological diverse samples is only possible for popular microarray platforms, making normalization of microarray samples for less popular platforms impossible.

Two recent preprocessing techniques solve the necessity of a reference dataset. The first method, Single Channel Array Normalization (SCAN) (Piccolo et al., 2012) uses intrinsic information of the sample for estimating the background noise and normalizing the microarray data. Since the SCAN method only uses information from the sample itself, it allows to preprocess all the Affymetrix datasets from InSilico DB. The second method, Universal exPression Code (UPC) (Piccolo et al., 2013), uses a similar model than SCAN for microarray data but supports a larger variety of data types making it a good option for combining datasets originating from different technologies like microarray and RNA-Seq.

The major focus of this master thesis is to enrich InSilico DB with the above cited preprocessing methods and to create an experimental genomic pipeline for generating UPC values for the RNA-Seq data type. RNA-Seq data files are usually heavy and can be categorized as being *Big Data*. Also aligning RNA-Seq reads to a reference genome can be time consuming. This Big Data has to be stored in a framework able to manage and process the data as fast as possible. The frame-

work we chose to use in this master thesis is the open-source *Apache Hadoop* project. Hadoop aims at storing huge numbers of large datasets on clusters and allow to process them in parallel. The Hadoop framework will be used to store the RNA-Seq files and speed up the computation of the RNA-Seq reads alignment parallelizing the computation.

The last important aspect of this master thesis is the analysis part. We use all the methods integrated to InSilico DB for performing Machine Learning experiments. These experiments comprise the benchmarking of the integrated preprocessing methods and an analysis for finding differentially expressed genes.

1.2 Outline dissertation

In the chapter 2 we explain the biological background necessary for understanding what genomic data is about. We explain among others the different methods for obtaining gene expression data. Chapter 3 details the preprocessing methods implemented in InSilico DB and the batch effect removal methods used in this master thesis. Chapter 4 lays the foundations for the methods by detailing the Big Data component of this master thesis. We used Hadoop as Big Data environment and give a concise overview of two of its components, namely, the Hadoop Distributed File System and Hadoop MapReduce. In the methods chapter 5, we explain how we enriched the genomic pipeline of InSilico DB with new preprocessing methods. We also use the knowledge acquired in chapter 4 for speeding up the computation of a sequence aligner for enabling a much faster computation of the Big Data. Machine Learning experiments for benchmarking and analyzing the genomic data using combinations of preprocessing and batch effect removal methods are described in chapter 6. We end this dissertation with the conclusions and future work in chapter 7.

CHAPTER 2

Biological Background

In this chapter we explain the biological context of the gene expression data used in this master thesis. The content of the first section is based on the book "Understanding bioinformatics" of Zvelebil and Baum (2007). The DNA microarray and RNA sequencing technologies for obtaining the gene expressions are also introduced.

2.1 Gene expression

Every living organism are made of at least one cells. For eukaryotic cells, the nucleus can be seen as the command center and contains the DNA molecules that encode the genetic information. DNA is a double-stranded helix-shaped structure where each strand is a chain of nucleotides that consists of four different bases: *adenosine (A), cytosine (G), guanine (G) and thymine (T)* . The bases adenosine and thymine are complementary bases and bind together, likewise for cytosine and guanine. These complementary bindings hold the double helix structure together.

Genes are regions of the DNA molecules that encode information about some characteristics of the organism, for example the hair color or important process that keeps the organism alive. For the cell to work, the genetic information held

by the genes must be synthesized into proteins. Proteins are molecules made of amino acids and perform almost every task inside an organism. Synthesizing proteins from DNA happens in two steps:

1. **Transcription**: The genetic information from the DNA molecule is transcribed into a special *ribonucleic acid* (RNA) molecule called messenger RNA (mRNA). The task of mRNA is to carry the genetic information outside the nucleus. RNA is a chain of bases arranged in groups consisting of 3 nucleotides, called codons. *Uracil (U)* replaces thymine in RNA and is also complementary with adenosine.
2. **Translation**: When the mRNA leaves the nucleus it goes into the cytoplasm where a ribosome decodes the carried information and produces an amino-acid chain that is subsequently folded into a protein.

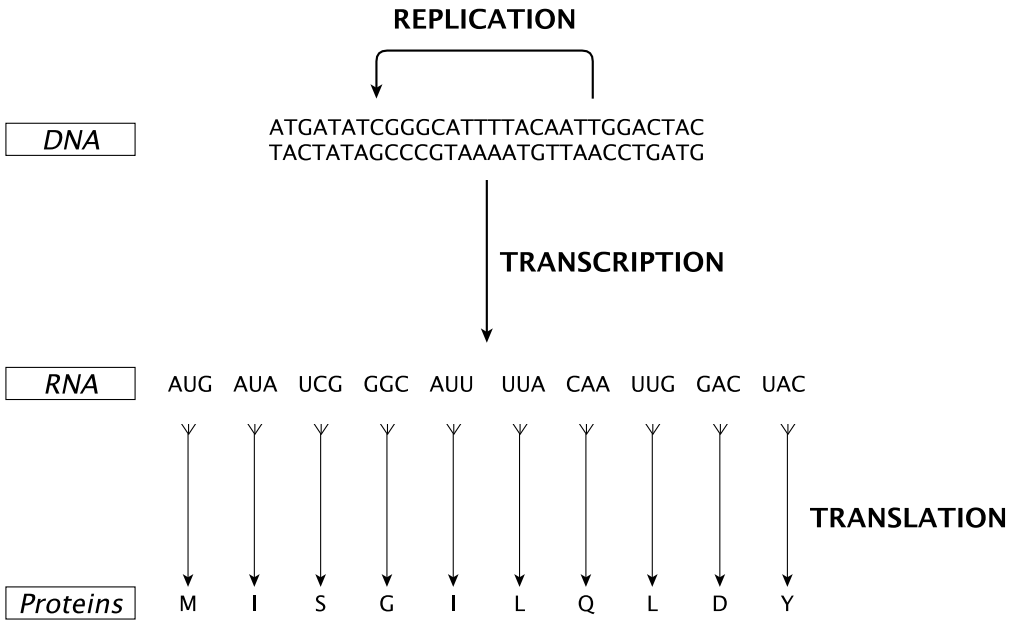


Figure 2.1: Central dogma of molecular biology. Replication (DNA synthesis): The DNA strands are separated during cell division and become blueprints for the synthesis of new complementary strands. Transcription (RNA synthesis): DNA is copied into mRNA. Translation (protein synthesis): mRNA is translated into a chain of amino acids, which are the building-blocks of proteins. Each codon in the mRNA corresponds to a specific amino acid.

Theses steps are part of the central dogma of molecular biology (Crick et al., 1970). The process of synthesizing proteins from DNA is called *gene expres-*

sion. It is important to note that not all the genes are expressed at the same time (only those required for producing the required proteins are expressed) and that the quantity of mRNA in a cell determines the level of gene expression in this cell.

Researcher came with ingenious ideas for measuring the mRNA level. Two popular methods for measuring the expression level of multiple genes simultaneously are the DNA microarray and RNA sequencing technologies. In following section we will describe how microarrays are designed and how they are able to measure the mRNA level of each gene. We also detail some preprocessing methods for normalization of microarray data. RNA sequencing will be briefly explained in section 2.3.

2.2 DNA Microarray

InSilico DB possess mainly Affymetrix microarrays (Lockhart et al., 1996) data. A DNA microarray (or simply microarray) is a solid surface (i.e. glass) with DNA fragments fixed onto it. These DNA fragments are short sequences from a gene of interest and are called *probes*. Every probe is positioned in a collection of probes having similar nucleotide sequences. This collection is referred to as *probe set* and determines the gene expression of a specific gene. Affymetrix arrays typically have probes consisting of 25 nucleotides (Lipshutz et al., 1999). There exist two different kind of probes on an Affymetrix microarray, namely, the perfect match (PM) and mismatch (MM) probes. Each PM probe is paired with a MM probe. The PM probe still represents a part of a gene sequence. The MM probe has the same sequence as the PM probe except for its central base (13th base). The MM probe aims to measure background noise and non-complementary bindings. Important to note is that there exist a lot of different microarray designs. One microarray can be designed with i.e. a smaller amount of probe sets than another microarray. Microarrays with the same design are said to be of the same *microarray platform*.

Obtaining gene expressions from microarrays can be summarized by the following steps (Lockhart et al., 1996):

1. **Target sample preparation:** mRNA is extracted from the target sample and fluorescent-labelled.
2. **Hybridization:** The mRNA binds with complementary probes on the DNA microarray. This binding process is called hybridization. The microarray

is washed and only the DNA fragments that are hybridized remain on the array.

3. **Scanning:** Laser-based scanners detect where the hybridization occurred by checking fluorescent spots on the array. The signal strength emitted by the spots indicates the intensity of the hybridization. The intensity values of the probe sets are determined by observing the intensity of hybridization at their respective location on the microarray .

The data resulting from the extraction are stored in CEL files and stores the intensity values of the probes.

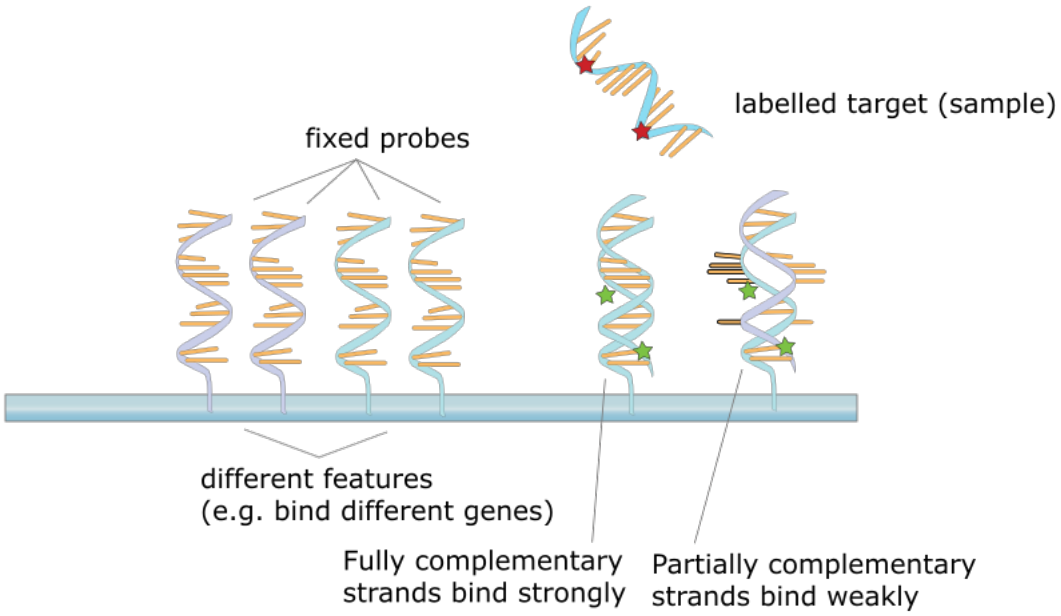


Figure 2.2: Hybridization of probes in DNA microarrays. Show fixed probes on the surface of an array before hybridization (left side of the figure) and after hybridization (right side). There are two cases after hybridization: the mRNA from the sample has bound with a complementary probe or it does not bind with a complementary probe. Non-complementary bindings occurs between DNA fragments that are similar but not identical and is referred to as cross-hybridization.

2.3 RNA Sequencing

RNA sequencing or *RNA-Seq* (Nagalakshmi et al., 2008) is a next-generation sequencing approach that is used for quantifying the gene expressions levels. The RNA-Seq method works as pictured in Figure 2.3 (Wang et al., 2009). The mRNA transcript (extracted from the sample) is converted into cDNA fragments. Sequencing adaptors are added to both ends of the cDNA fragments. A high-throughput sequencing method is used for obtaining the short sequence reads. The short-reads are obtained from one-end (single-end reads) or both ends (paired-end reads) of the fragments. These reads are aligned with the reference genome or reference transcripts and are classified in three different types: junction reads, exonic reads and poly(A) end-reads. These tree types determine the gene expression profile of the sequenced sample.

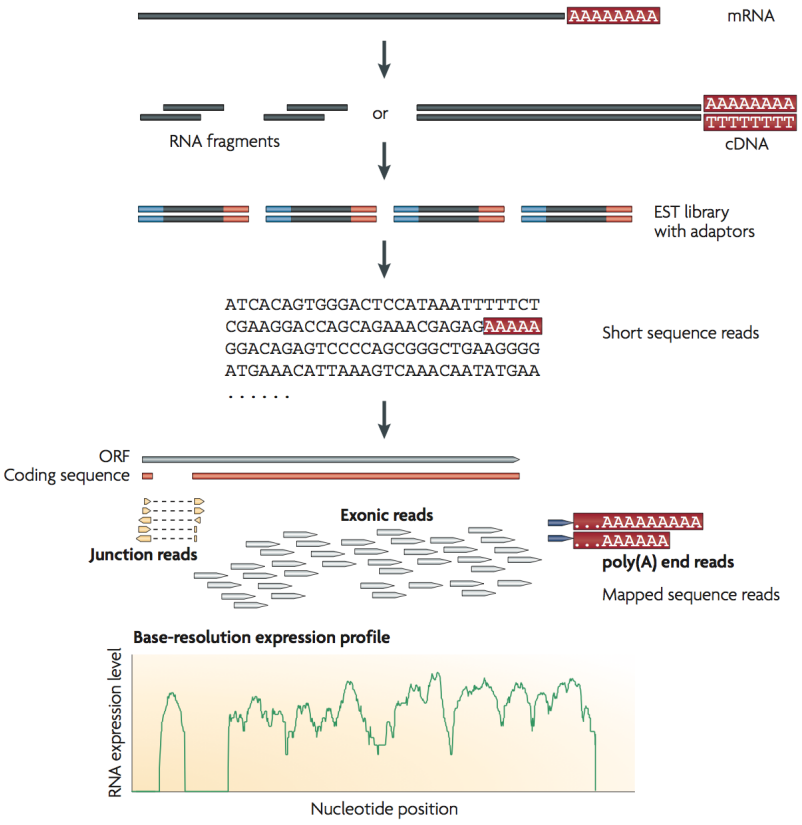


Figure 2.3: Working of the RNA-Seq method. Image taken from (Wang et al., 2009)

There are two different approaches for reconstructing the transcripts from RNA-Seq reads(Haas and Zody, 2010). The first approach is called the *assemble-then-aligns* or the *de novo* approach. It first assembles the reads and then align them to the reference genome. The second approach does the opposite. It first aligns the reads to the genome and then assembles the aligned reads. Hence the name of the *align-then-assemble* approach. InSilico DB reconstructs the transcripts using the align-then-assemble approach. The software used in InSilico DB for aligning the reads to the reference genome is TopHat (Trapnell et al., 2009) and Cufflinks (Trapnell et al., 2010) is used for assembling the transcripts.

2.3.1 FASTQ file format

The FASTQ files format contain records representing the short read sequences and a corresponding quality score. Each record consists of four lines in the file. An example of a record is:

```
@SEQ_ID
GCCGAGATCTTTCTCCCACCATGGCCTGAAGAGGCTTGGTGGACCCCGAT
+
HHHHEHBHAEDFHHEHHEHEEEHH@HHHHHHGEEHEGHEHCGHAHEHEH
```

The definition of a record in a FASTQ file is as follows (Cock et al., 2010):

First line: @ character followed by the identifier of the sequence. It is optionally followed by a description.

Second line: Represents the read sequence. The sequence is composed of the bases (for the DNA the bases are A, C, G, T).

Third line: The third line is a + character, optionally followed by the same sequence identifier and description as in the first line. This line serves to denote the end of the sequence and the beginning of the qualities.

Fourth line: Represents the quality scores of the corresponding bases in the sequence. The qualities are encoded as ASCII characters. The first character is the quality of the first base, the second character of the second base, and so on. These quality scores indicates the probability that the corresponding base is incorrect. The lower the quality score, the higher the probability the base is incorrect. There exist different quality score formats, each mapping to another range of ASCII characters. For example, InSilico DB uses the Sanger format which has for lowest quality the ASCII character ! (score 0) and as highest quality the characters I (score 40).

Another important thing about FASTQ files is that there are two sorts of FASTQ files:

Single-end FASTQ file: A single-end FASTQ file contains the sequence data obtained during a RNA Sequencing experiment that sequenced only one of the ends of the molecules.

Paired-end FASTQ files: Sequence data is stored in two different files when the RNA-Sequencing experiment sequences both ends of the molecules. The two resulting files are called paired-end files and each file stores the information of one of the ends. This means both files contain exactly the same sequences identifiers, but the content of the records is the data of their respective end.

2.4 Summary

The DNA is a chain of nucleotides describing the genetic information of the organism. Regions defining a specific characteristic of the organism are called genes. Genes are expressed during the synthesis of proteins, which are the molecules performing almost all the functions of the cell. Messenger RNA (mRNA) is produced during the synthesis of proteins. The quantity of mRNA denotes the level of gene expression in the cell.

The DNA microarray technology (Lockhart et al., 1996; Lipshutz et al., 1999) is able to measure the quantity of mRNA in a sample by using the principle of hybridization. The microarray is a chip that has DNA fragments on its surface called probes. Similar probes are grouped in sets representing a specific gene. A microarray experiment is performed starting with the preparation of the to-analyse tissue. Messenger RNA is extracted from the tissue and fluorescent labelled. This mRNA is hybridized with the microarray probes. Finally the microarray is scanned and the intensity of the hybridization is determined by the fluorescent signals. This intensity is measured for each probe set and is its raw gene expression.

RNA-Seq (Nagalakshmi et al., 2008) is a next-generation sequencing technology for determining the level of gene expression in samples. It converts mRNA into cDNA and reads in parallel the short read sequences from the adaptors added to the ends of the cDNA fragments. The read are then aligned to the reference genome for the reconstruction of the gene expression profile of the sample.

CHAPTER 3

Bioinformatics Background

In this chapter we explain two important microarray-related concepts. The first concept is the preprocessing of data. Preprocessing of microarrays is necessary if one wants to compare data from multiple microarrays. There exist a certain amount of different techniques like MAS5 (Affymetrix, 2002), RMA and fRMA, etc. We decided to explain a representative subset of them, namely, RMA, fRMA, SCAN and UPC. The reason for choosing to explain RMA is because it is a popular preprocessing technique having a lot in common with fRMA. FRMA was chosen because the InSilico DB database uses this prepocessing technique for its samples. SCAN and UPC were chosen because those are the preprocessing techniques that have been integrated to the InSilico DB database for the purpose of this master thesis.

The second important concept is the batch effect. It is a variation caused by combining microarrays data that do not originate from the same batch. In section 3.2.1 we explain what a batch and batch effect really are and how batch effect occurs. In section 3.2.2 we describe some methods for removing the batch effect.

The InSilico DB stores and is optimized for mainly Affymetrix microarray data. Thus, every time the term ”microarray” is used in the reminder of this dissertation we are meaning Affymetrix microarray.

3.1 Data preprocessing

An essential step in obtaining gene expression values is to preprocess the data. The preprocessing aims at making different samples usable in analysis by reducing the noise and normalizing the gene expression values. The focus of this section is mainly on microarray data, but RNA-Seq preprocessing will also be explained in section 3.1.5.

The values extracted from DNA microarray experiments are array dependant, therefore preprocessing the gene expressions is required before any analysis. Most preprocessing methods consist of three distinct steps (Bolstad, 2004) :

1. **Background correction:** Adjusts the background noise from the microarray data. Such background noise can be caused by i.e. cross-hybridization. Cross-hybridization is the non-complementary binding of a DNA strand from the DNA sample with a probe sequence.
2. **Normalization:** Removes the non-biological variation from the data. There a many factors introducing such variations. One example could be that different microarray equipment results in different distributions due to different experimental settings.
3. **Summarization:** Combines the intensity values from every probe in each probeset into a gene expression value for the probe set.

Typically, the normalization and summarization steps are multiarray procedures. Also, the gene expression values are log transformed in every preprocessing method described below. A lot of different preprocessing techniques exist, therefore only a small representative subset of them are detailed below.

3.1.1 Robust Multiarray Average

Robust multiarray Average (RMA) (Irizarry et al., 2003a) is a popular preprocessing technique for Affymetrix microarrays. Even if the InSilico DB database does not preprocess data using RMA, it is important to detail this method before explaining frozen Robust Multiarray Analysis which is very similar.

Noise is always introduced during a microarray experiment. Therefore RMA begins the preprocessing of the observed probe intensities by correcting the background noise. RMA assumes the probe intensity PM to be a combination of background noise bg and biological signal s. This is modelled in following equa-

tion:

$$PM_{ijn} = bg_{ijn} + s_{ijn} \tag{3.1}$$

with $i \in 1, ..., I$ being the microarray wherein probe $j \in 1, ..., J$ in probe set $n \in 1, ..., N$ is observed. The goal of the background correction is to obtain the value of the signal. Since the noisy probe intensity is the only known value, the background correction function can be defined as a conditional expectation of the signal given the noisy probe intensity:

$$B(PM_{ijn}) = E(s_{ijn}|PM_{ijn}) > 0 \tag{3.2}$$

where the signal is assumed to be positive. Assuming the distribution of the background noise is normally distributed $bg \sim N(\mu, \sigma^2)$ and the distribution of the signal is exponentially distributed $s \sim exp(\lambda)$. Those assumptions make it possible to derive a formula for $E(s_{ijn}|PM_{ijn})$ (Bolstad, 2004, p. 17–21).

$$E(s|PM) = a + b\frac{\phi(\frac{a}{b}) - \phi(\frac{PM-a}{b})}{\Phi(\frac{a}{b}) + \Phi(\frac{PM-a}{b}) - 1} \tag{3.3}$$

with $a = PM - \mu - \sigma^2\lambda$, $b = \sigma$, $\Phi(.)$ being the standard normal distribution and $\phi(.)$ the standard normal density function. This equation reduces the problem to the estimation of the values of μ, σ and λ in order to obtain the background corrected value for the probe intensity.

After every microarray has been background corrected separately, the data needs to be normalized to make different microarrays comparable in analysis. For normalizing the data, RMA applies quantile normalization(Irizarry et al., 2003b) using all the microarrays simultaneously. It results in equal distributions for the probe intensities of every microarray and thus make their values comparable.

The last step is to summarize all the probe intensities for every probe set into one gene expression value for this probe set. The background corrected and normalized probe intensity Y can be described as the true gene expression θ plus an effect specific to the probe ϕ and a measurement error ϵ. (Irizarry et al., 2003a,b). The following probe-level linear model describes the summarization problem:

$$Y_{ijn} = \theta_{in} + \phi_{jn} + \epsilon_{ijn} \tag{3.4}$$

The indices notation is the same as for 3.1. The probe-specific effect is assumed to have the characteristic $\sum j\phi_{jn} = 0 \forall n$. This characteristic means that all the probes in the probe set are chosen to be representative of the gene modelled by the probe set. The goal is to estimate the values of θ_{in} which are in fact the

gene expression values. Estimating the gene expressions is performed by median polish(Irizarry et al., 2003a), which is a robust method against outliers.

There are a few drawbacks by using multiarray preprocessing methods like RMA (McCall et al., 2010). First, they need all the data samples to be preprocessed at once. This means the dataset need to be preprocessed again every time a new sample is added. Secondly, two datasets preprocessed using the same multiarray preprocessing method cannot be compared unless they were preprocessed in the same batch. A batch is a set of samples computed together at the same time. A more detailed explanation of a batch is given 3.2.1. The normalization step normalizes the probe sets using the data from the batch. It implies that two separately normalized batches have different distributions for the same probe set and cannot be compared. Methods for removing such variations are introduced in next section.

3.1.2 Frozen Robust Multi-array Analysis

McCall et al. (2010) proposed the *frozen Robust Multiarray Analysis* (fRMA) preprocessing method to address those drawbacks. They created a training data set by taking datasets from GEO (Edgar et al., 2002) and ArrayExpress (Parkinson et al., 2009), two repositories of publicly available datasets. McCall et al. selected a large variety of studies and tissue types and generated all the possible combinations. They selected randomly 5 samples from the 170 study/tissue type combinations resulting in a training dataset of 850 samples. This training dataset is used as reference distribution and for estimation of probe-specific effects and variances. Those informations are saved (frozen) and used during the normalization and summarization step, hence the name frozen Robust Multiarray Analysis.

In the fRMA method, a random-effect model is assumed McCall et al. (2010) for the summarization of probes:

$$Y_{ijkn} = \theta_{in} + \phi_{jn} + \gamma_{jkn} + \epsilon_{ijkn} \tag{3.5}$$

the same parameters and notation is used as in model 3.4 except for the added notation of different batches $k \in 1, ..., K$ and the random-effect parameter γ representing the effect of probes across batches.

This model should be used for the estimation of the frozen components, but fitting this model is not straightforward. Therefore the RMA model 3.4 is used for fitting the training dataset and obtaining the estimators $\hat{\phi}_{jn}$ and $\hat{\theta}_{in}$. These estimators are then used to derive residuals from model 3.5: $r_{ijkn} = Y_{ijkn} - \theta_{in} + \phi_{jn}$.

The residuals are used for the estimation of the variances of the random-effect $Var(\gamma_{jkn}) = \tau_{jn}^2$ and the variance of the measurement error $Var(\epsilon_{ijkn}) = \sigma_{jn}^2$. The estimators for the probe-specific effect $\hat{\phi}_{jn}$ and the variances $\hat{\tau}_{jn}^2$ and $\hat{\sigma}_{jn}^2$ are frozen as components of the fRMA and will be used during the summarization step.

The fRMA preprocessing method (McCall et al., 2010) follows the same three steps as RMA. Each microarray's probe intensities are background corrected by the same background correction function as in RMA. The next step is to normalize the distributions of the probes across the samples. This is a single-array procedure in fRMA, since the frozen reference distribution is now used during the quantile normalization. Every microarray is thus quantile normalize separately with the reference distribution. There are two different approaches for summarizing the gene expression for a probe set: a single-array or a batch summarization approach. The single-array approach will be the one described in this dissertation since it is the one used by InSilicoDB.

The summarization of the probes starts by correcting the probe-effect. The correction is performed by subtracting the frozen estimated probe-effect $\hat{\phi}_{jn}$ from the background corrected and normalized probe intensity.

$$Y_{ijkn}^* \equiv Y_{ijkn} - \hat{\phi}_{jn} = \theta_{in} + \gamma_{jkn} + \epsilon_{ijkn} \tag{3.6}$$

The gene expressions θ_{in} of each probe set has to be summarized after the probe-effect correction. A robust weighted-average method is used for the estimation of θ_{in}. Each intensity is weighted by the inverse of its variance.

$$\theta_{in} = \sum_{j=1}^{J_n} \frac{w_{jn}}{v_{jn}} Y_{jn}^* / \sum_{j=1}^{J_n} \frac{w_{jn}}{v_{jn}} \tag{3.7}$$

Where $v_{jn} = Var(Y_{jn}^*) = \hat{\tau}_{jn}^2 + \hat{\sigma}_{jn}^2$ and w_{jn} is determined by an M-estimator function. Note the batch and array notations have been omitted for the intensity since its single-array.

Even if fRMA has not the drawbacks of multiarray preprocessing techniques it still has one problem: fRMA needs for every Affymetrix microarray platform a lot of datasets for the creation of the frozen training set and its components. This limits the usage of fRMA to popular microarray platforms for which a large variety of publicly available datasets exist.

3.1.3 Single Channel Array Normalization

Single Channel Array Normalization (SCAN) (Piccolo et al., 2012) is a preprocessing method that removes the dependence of multiple arrays or reference training set for preprocessing microarray data. It background corrects the data using only intrinsic information about the genomic composition of the probes and estimates the effect of the composition on the probe intensities using a linear statistical model.

The idea of SCAN is that there are two distinct populations in the microarray data. There are probes that hybridized and emit a biological signal influenced by background noise and probes that did not hybridize and only show background noise. A mixture-model of two normal distributions is used to measure this background noise and the signal plus background noise:

$$Y_i = (1 - \Delta_i)Y_{1i} + \Delta_i Y_{2i} \tag{3.8}$$

where Y_i is the probe intensity of probe $i \in 1, ..., I$. The random variables $Y_{1i} \sim N(X_i\theta_1, \sigma_1^2)$ and $Y_{2i} \sim N(X_i\theta_2, \sigma_2^2)$ represent the distributions for respectively the background noise and the background noise + signal. The means of the distributions of probe i, $X_i\theta_m$, are determined by a slightly modified Model-based Analysis of Tiling arrays method (MAT) (Johnson et al., 2006). The variable Δ_i is 1 or 0 and indicates which of the background or background plus signal distribution is of importance for probe i. The value of $\Delta_i \sim Bernoulli(\pi)$ is unknown and is estimated by reformulating the model as the likelihood:

$$L(y_i, \Delta_i) = [(1 - \pi)f_1(y_i|\theta_1)]^{1-\Delta_i}[\pi f_2(y_i|\theta_2)]^{\Delta_i} \tag{3.9}$$

where $f_m(y_i|\theta_m)$ is the density function of the m-th mixture component and π is the percentage of non-background probes. The Expectation-Maximization algorithm (Dempster et al., 1977) is used for estimating the Δ_i. The Expectation-step assigns to the missing data from the above model its new expected values using the maximized parameters from previous Maximization-step. In the Maximization-step, the model parameters are maximized given the expected values for the missing data in previous Expectation step. This process is iterated until convergence.

The background and background plus signal distributions are represented as modified MAT models:

$$Y_{mi} = \alpha_m n_{iT} + \sum_{j=1}^{25} \sum_{k=A,C,G} \beta_{mjk} I_{ijk} + \sum_{l=A,C,G,T} \gamma_{ml} n_{il}^2 + \epsilon_{mi} \tag{3.10}$$

where the first term represents the effect of the number of thymines in the probe sequence. α_m is a constant based on the nucleotide count n_{iT} for nucleotide T. The second term describes the effect of the others nucleotides (adenosine, cytosine, guanine) in the sequence. It looks for every position $j \in 1, ..., 25$ of probe i which nucleotide $k \in A, C, G$ is present on position j. The effect of the nucleotide on position j is determined by β_{mjk}. The third term is the effect γ_{ml} for every nucleotide on its squared count in the probe sequence i. The last term is an error term for probe i.

First, the parameters α, β and γ are estimated by ordinary least square for a subset of 50000 probes. The estimated parameters enable to predict for each probe a baseline intensity, which is used for fitting model 3.8. Once the model is fitted, the background component can be used as estimate for the background intensity m_i for every probe i. The probes are assigned to bins of 5000 probes with similar ms. The equation for obtaining the standardized probe intensity t_i is very similar to a z-score. Every probe i is normalized by subtracting its estimated background intensity m_i from its probe intensity Y_i (background correction) and divided by the standard deviation of its bin sd_{ibin} :

$$t_i = \frac{Y_i - m_i}{sd_{ibin}} \tag{3.11}$$

3.1.4 Universal exPression Code for DNA microarray

The last preprocessing method presented in this section is the *Universal exPression Code* (UPC) (Piccolo et al., 2013). UPC's goal is to provide a platform independent preprocessing method. This should enable to perform data analysis regardless the data technology used i.e. microarray or next-generation sequencing data. The approach of this method assumes, likewise SCAN, the presence of two distinct populations in the data, namely, active and inactive genes. Inactive genes measure the background noise while active genes measure background noise plus the signal. The UPC values are not strictly speaking gene expressions but rather the probability the gene is expressed.

The same two-component mixture model 3.8 as the SCAN model is used to estimate whether a gene is active or inactive. Expectation-Maximization is also used for estimating the model parameters. After convergence of the EM-algorithm, the UPC values are given by the conditional expectation of Δ_i given the EM estimates $\hat{\pi}$, $\hat{\theta_1}$ and $\hat{\theta_2}$:

$$P_i = E(\Delta_i | y_i, \hat{\pi}, \hat{\theta_1}, \hat{\theta_2}) = \frac{\hat{\pi} f_2(y_i|\hat{\theta_2})}{(1-\hat{\pi}) f_1(y_1|\hat{\theta_1}) + \hat{\pi} f_2(y_i|\hat{\theta_2})} \tag{3.12}$$

For Affymetrix microarrays, the components of the mixture model are two normal distributions likewise SCAN and are represented by the same modified MAT model 3.10.

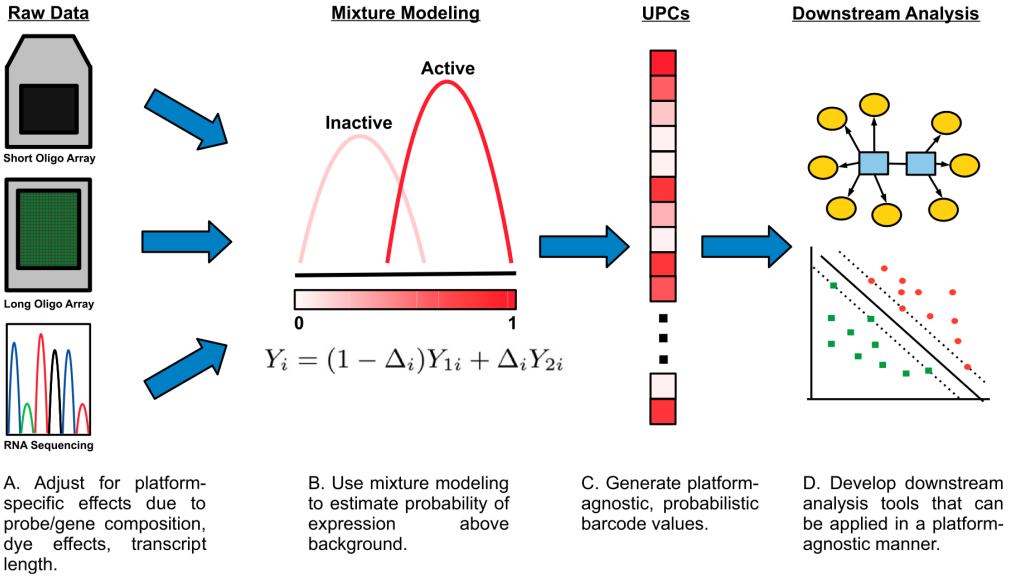


Figure 3.1: Universal exPression Code. Figure taken from (Piccolo et al., 2013)

3.1.5 Universal exPression Code for RNA-Seq

The UPC preprocessing method can also be used the preprocessing of RNA-Seq data using the RNA-Seq read-counts for each gene. The mixture model 3.8 approach is still used. The difference between UPC preprocessing of microarray and RNA-Seq data resides in how the mixture components are modelled. The following model represents the m-th component for gene i:

$$Y_{mi} = \alpha_m + GC_i\beta_m + I_i\delta_m + \epsilon_{mi} \tag{3.13}$$

with α_m as intercept, β_m modelling the effect of the Guanine-Cytosine composition in gene i, δ_m being the effect of the length I_i of gene i. The error term ϵ_{mi} is assumed to be normally distributed for all the genes.

Unlike UPC for microarray, the mixture components are not always assumed to be normal distributions for RNA-Seq data. Since the input are discrete read counts. Two other possible distributions are used for approximation, namely, log-normal and negative-binomial distributions. The parameters of the models are estimated using the Expectation-Maximization algorithm.

3.2 Batch effect removal

Combining multiple datasets into one large dataset for large-scale analysis is possible thanks to the huge amount of available data in public repositories such as InSilico DB. A large-scale analysis of thousands of samples could potentially lead scientists to new knowledge important for topics such as cancer prognosis (Ein-Dor et al., 2006). But integrating tens or hundreds of datasets in one experiment adds new variations. Batch effect is an important source of variation when multiple datasets coming from different microarray/RNA-Seq experiments are combined into one analysis (Lazar et al., 2012). Several techniques have been developed by researchers for addressing the problem of undesired variations in cross-platform analysis.

In this section we explain what batch effect is and its possible sources in subsection 3.2.1. In subsection 3.2.2 we review some of the existing batch effect removal techniques for microarray data. Those techniques are implemented in the InSilicoMerging package (Taminau et al., 2012).

3.2.1 Batch-effect

For understanding what batch effect is about it is important to know what a *batch* truly is. A batch is defined as being a set of samples processed over the same period, in the same environment and using the same technology (i.e. RNA-Seq sequencer, microarray platform,...)(Chen et al., 2011).

Lazar et al. reviewed multiple definitions of batch effect and unified them into one definition:
Definition 1. *The batch effect represents the systematic technical differences when samples are processed and measured in different batches and which are unrelated to any biological variation recorded during the microarray gene expression experiment (Lazar et al., 2012).*

This definition is interesting since it describes the most important characteristics of batch effect. *Systematic* indicates that batch effect is always an issue when combining multiple batches in one experiment. The term *technical* points out that batch effect arises from all the manipulations from the start till the end of the experiment. Finally, the definition insists that the variations are not biological. Interpreting results without removing the batch effect can mislead the scientists to false conclusions for their experiment. Batch effect removal is thus a crucial step when integrating multiple batches in one experiment.

Many different sources can cause batch effect. Each stage of the microarray experiment from the manufacturing of the array to the extraction of the data after hybridization can possibly add batch effect to the data. Typical examples (Luo et al., 2010) of sources introducing batch effect are:

- using different technologies, i.e. different microarray platforms
- differences of the microarray qualities
- different environment conditions during the complete process (storage, preparation, ...)
- different protocols for the hybridization process
- different material or settings

Figure 3.2 gives an overview of a large number of possible origins of batch effects for every stage in a microarray experiment. This figure shows clearly that batch effect can be introduced at any time during the experiment. Another important remark is that microarray experiments can be subject to more than one source of batch effect at the time or during the experiment lifespan, increasing the batch effect on the final data. This stresses even more the necessity of performing batch effect removal.

3.2.2 Batch effect removal techniques

Batch effect removal techniques assume the batches being from the same organism and having a similar distribution of samples for the biological variables of interest Lazar et al. (2012). This means that the batch effect of studies with only diseased tissue cannot be removed, making the study unsuitable for integration in an analysis aiming for the discovery of differentially expressed genes for healthy and diseased tissues.

Batch effect is assumed to be an additive or multiplicative effect on the gene expressionLazar et al. (2012). When the gene expression values are in log 2 scale, both additive and multiplicative batch effects are added to the gene expression. Following equation is a general mathematical model for the representation of batch effect:

$$Y_{ij} = g_{ij} + b_{ij} + \epsilon_{ij} \tag{3.14}$$

where Y_{ij} is the gene expression affected by the batch effect b_{ij} and the noise ϵ_{ij}. The unaffected gene expression is represented by g_{ij}. The goal of the batch effects removal techniques is to make the studies comparable by either adjusting the gene expression Y_{ij} or by removing the batch effect term b_{ij} from the equation.

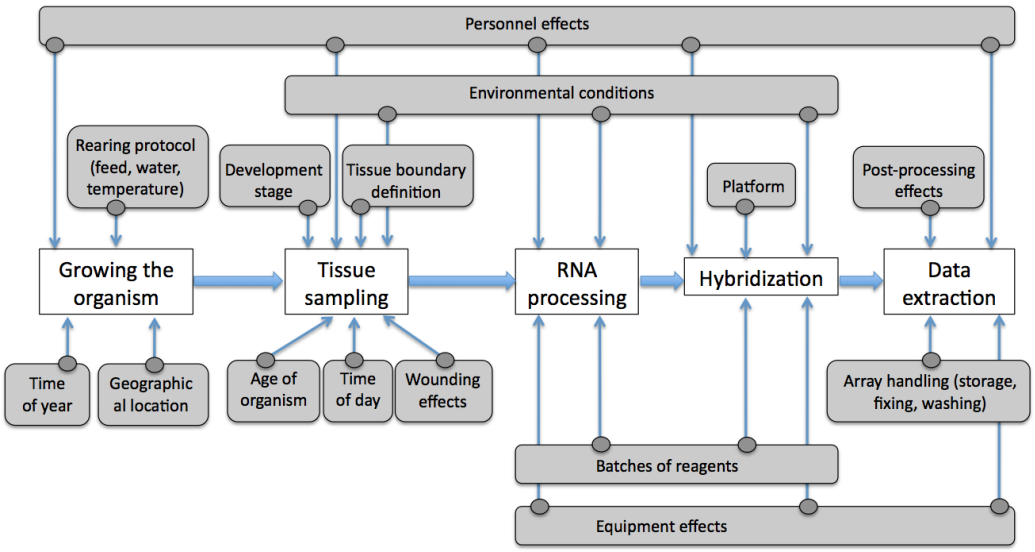


Figure 3.2: Depiction of possible batch effect sources during the complete microarray experiment. The white boxes represent the different stages of a microarray experiment. The grey boxes are the possible origins of batch effect. Figure taken from(Lazar et al., 2012).

Empirical Bayes

A popular batch effect removal technique is the *Empirical Bayes* method, also known as *ComBat* (Johnson et al., 2007). ComBat estimates the parameters of the model by regrouping information from genes in each batch. This information is used to shrink the batch effect parameter estimates for each gene toward the mean of the batch effects estimates across the batches.

The measured gene expression Y_{ij} for gene $i \in 1, ..., I$ in sample $j \in 1, ..., J$ can be described as:

$$Y_{ij} = g_i + C\beta_i + \gamma_i + \delta_i \epsilon_{ij} \tag{3.15}$$

the real gene expression value g_i for gene i, combined with the covariates $\mathbf{C}$ describing the sample conditions and the regression coefficients β_i corresponding to the covariates. γ is an additive and δ is a multiplicative form of batch effect for gene i. The measurement error is represented by ϵ_{ij} and is assumed to be normally distributed with mean zero and variance σ_i^2.

ComBat is a three-steps algorithm (Johnson et al., 2007):

1. Standardization of data

2. Estimation of parameters
3. Adjustment of batch effect

1. Standardization of Data The gene expression values are first standardized so that the overall mean and variance is equal. To do so, g, β and γ are estimated using ordinary least square method. The standardized gene expression follows the normal distribution $z_{ij} \sim N(\gamma_i, \delta_i^2)$:

$$z_{ij} = \frac{Y_{ij} - \hat{g}_i - \mathbf{C}\hat{\beta}_i}{\hat{\sigma}_i} \tag{3.16}$$

2. Estimation of parameters The parametric approach for estimating the batch effect parameters assumes the prior distributions $\gamma_i \sim N(\gamma, \tau^2)$ and $\delta_i^2 \sim InverseGamma(\lambda, \theta)$, where the parameters γ, τ^2, λ and θ are estimated empirically using the standardized data.

3. Adjustment of batch effect Once batch effect estimators γ_i^* and δ_i^* are calculated and enable to adjust the batch effect of the gene expression:

$$Y_{ij}^* = \frac{\hat{\sigma}_i}{\hat{\delta}_i^*}(z_{ij} - \hat{\gamma}_i^*) + \hat{g}_i + \mathbf{C}\hat{\beta}_i \tag{3.17}$$

Batch mean-centering

Batch mean-centering (BMC) (Sims et al., 2008) transforms the data such that the mean of each gene becomes zero. The transformation is performed per batch by determining the mean of each genes and subtracting the mean from the gene value.

$$\hat{Y_{ij}} = Y_{ij} - \bar{Y_i} \tag{3.18}$$

Gene standardization

The *Gene standardization* (GENENORM) (Luo et al., 2010) goes a step beyond batch mean-centering. Gene standardization transforms per batch each gene by applying a z-score standardization. This results in a zero mean and a standard deviation of one for each gene.

$$Y_{ij} = \frac{Y_{ij} - \bar{Y_i}}{\sigma_i} \tag{3.19}$$

Distance-weighted discrimination

Distance-weighted discrimination (DWD) has been introduced for improving the Support Vector Machines (SVN) (Cristianini and Shawe-Taylor, 2000) performance in "high-density low sample size context" (Marron et al., 2007). Instead of maximizing the minimum distance to the separating planes, DWD maximizes the sum of inverse distances. Benito et al. (2004) described how DWD could be used for removing the batch effect of two batches :

1. The direction vector Δ_i from DWD is computed. It is the normal vector of the hyperplane between the two batches.
2. The samples of the two batches are projected in the DWD direction.
3. The mean distance $d(\bar{b}_Y)$ to the hyperplane is determined for all the samples of each batch.
4. The samples are moved in the direction of the hyperplane by subtracting for each gene the result of the multiplication of the DWD direction and the mean distance.

The last step can be summarized in formula (Lazar et al., 2012):

$$\hat{Y}_{ij} = Y_{ij} - d(\bar{b}_Y)\Delta_i \tag{3.20}$$

3.3 Summary

In this chapter we described two important concepts for the remain of the master thesis. In the first section we detailed various Affymetrix microarray normalizations and in the second section we explained the need of using batch effect removal methods when combining multiple datasets.

The preprocessing of microarray data is a necessary step before any analysis. The data is influenced by different factors during the hybridization process making the results array specific. The gene expression values need to be preprocessed for removing the noise introduced during the microarray experiment. This preprocessing also normalizes and summarizes the data of each probe set enabling analysis of multiple microarrays. Four representative preprocessing techniques have been detailed in this chapter, namely, RMA, fRMA, SCAN and UPC.

There are two kinds of preprocessing methods: multi-array and single-array methods. Multi-array methods need a set of microarray samples for normalizing the data. The preprocessing must be done again for every new sample added to the

	RMA	fRMA	SCAN	UPC
multi-array method	X	X		
single-array method		X	X	X
independent of external information			X	X
technology independent				X

Table 3.1: Summary of the preprocessing techniques

dataset. Also, the normalization uses only information from the given microarray samples making two datasets preprocessed separately incomparable (if the variation is not removed). The single-array method has the advantage of making microarrays from different datasets comparable without having to preprocess them again. Another important aspect of preprocessing techniques is the dependency on external information. For example, the fRMA method uses precomputed information from a training dataset. This dependency limits the usage of fRMA for microarray platforms where such information is available. Finally, universal exPression Code is a preprocessing methods independent of the underlying technology. This enables to preprocess microarrays and RNA-Seq using UPC, making it possible to use both data types in the same analysis.

A combination of datasets from different experiments is often done when performing large-scale data analysis. Those experiments are most of the time coming from different studies and/or batches. Using multiple batches simultaneously adds an extra variation in the data due to technical differences during the complete process of the each batch's experiment. This variation is called batch effect. Since batch effect is unavoidable, scientists combining datasets for analysis should always keep batch effect in mind and try to remove this unwanted variation before making conclusions based on their unchanged data.

Even if preprocessing techniques (i.e. fRMA, SCAN and UPC) try to remove the batch effect, it is still present when datasets from multiple microarray platforms are integrated to a microarray analysis (Lazar et al., 2012). The existing batch effect removal methods aim at removing the batch effect not only across batches of the same platform but also across batches from different technologies. It is important to note that batch effect can only be removed in datasets having similar distributions for the variable of interest.

Batch effect removals methods have been introduced in subsection 3.2.2. Those methods were implemented in a R package by Taminau et al. (2012) and will be used in the Machine Learning experiments in chapter 6.

CHAPTER 4

Big Data background: Apache Hadoop

Next-generation sequencing technologies, like RNA-Seq, output large quantities of data making the analysis of all the data is not complex without the tools able to cope with such Big-Data. A promising software for managing Big Data is Apache Hadoop (O'Driscoll et al., 2013).

The Apache Hadoop software is an open-source framework consisting of multiple modules for enabling distributed processing of large datasets across a network of machines. The modules we detail in this chapter are the Hadoop Distributed File System (HDFS) and the Hadoop MapReduce modules. HDFS is a distributed file system that is designed to be fault-tolerant and performing well with large datasets. The distributed data-processing is achieved by the Hadoop MapReduce module.

We begin this chapter by giving a brief history of the Hadoop project, its main inspiration and its contributors. Then the focus is on two of Hadoop's modules: Hadoop Distributed File System and Hadoop MapReduce. First, the goals, architecture and design choices of the HDFS module are described. In the third section, the working of the MapReduce framework is explained in details. The book "Hadoop: The Definitive Guide" White (2009) has been used as main reference for the whole chapter.

4.1 A short history of Hadoop

Apache Nutch was an open-source web search engine created in 2002 by Doug Cutting and Mike Cafarella as part of the text search library project, *Apache Lucene* (White, 2009). After implementing the base of the search engine, they realized its architecture could not scale to the billions of webpages. At this time, Google published their papers describing the *Google File System* (GFS) (Ghemawat et al., 2003) and the *MapReduce* paradigm (Dean and Ghemawat, 2004). Those publications had a great impact on the Nutch project. In 2004, Nutch had its own working open source version of a distributed file system called *Nutch Distributed File System* (NDFS) and a MapReduce implementation working on it.

The NDFS and MapReduce implementations became another subproject of Lucene, namely, the open-source *Apache Hadoop* project (White, 2009). *Yahoo!*, interested in the Hadoop project, hired Doug Cutting at the same period and became a really active contributor to the development of the project (Shvachko et al., 2010). They mainly contributed to the Hadoop Distributed File System and the MapReduce frameworks accounting for 80% of the developments. Hadoop became in 2008 an official top-level project of Apache and has been used by numerous companies since. The Hadoop clusters of Yahoo! counted in 2010 more than 25 000 machines accounting for 25 petabytes of data. Other prominent companies use and contribute to Hadoop, i.e. Facebook with the Fair Scheduler (White, 2009).

4.2 Hadoop Distributed File System

Hadoop Distributed File System (HDFS) is like its name suggests a distributed file system. A distributed file system is a file system that manages data storage across a network consisting of multiple machines (also called nodes). Managing a large amount of nodes is a very complex task and necessitates all sorts of security measures. A typical example could be data corruption or a machine that shuts down. How does a distributed file system prevent data loss in the cluster? This is the kind of problems the Hadoop Distributed File System tries to address.

4.2.1 Goals

When coping with huge amounts of data and/or large clusters of servers, some critical points need to be taken into consideration. The Hadoop Distributed File

System aims at providing solutions to following critical points (White, 2009; Borthakur, 2007):

Measures against hardware failures: When there are hundred or thousand servers in a cluster (possibly commodity hardware), there exists a high probability that at some point in time a machine stops to work correctly. HDFS has mechanisms to identify and recover of such hardware failures.

Performance with large datasets: The file system should be able to support very large files (from gigabytes to terabytes) and facilitate their computation. The clusters are scalable in HDFS and provide a large aggregate bandwidth.

Data consistency: It is very important that data stays consistent in a file system, corrupted data should be detected and removed.

Hadoop's architecture is thus designed with these critical points in mind.

4.2.2 Architecture

In HDFS (White, 2009; Shvachko et al., 2010) there are two distinct types of cluster nodes, namely, the NameNode and the DataNodes. The NameNode is an unique master node that manages the file system namespace and the access to the files. The DataNodes are worker nodes that manage the data storage of the system. The to-store data is split by the NameNode in blocks of predefined size and replicated on different DataNodes. The blocks are typically 64/128 megabytes large and replicates (the standard is three replicates) are made for each block. Those values can be configured by the user. All the nodes in the clusters are connected and are able to communicate with each other using TCP/IP-protocols.

Blocks

Each file stored in the HDFS is chunked into *blocks* (White, 2009) of a predefined size and replicated on different nodes. The block size is typically 64/128 megabytes (depending on the installation). Since Hadoop's goal is to store large files, having small blocks (i.e. 1 kilobyte) would create a lot of requests and create a lot of data transfer. Having a large block size minimizes the requests enabling the data transfer to be at the disk transfer rate. The replication of blocks is important for preventing data loss if a node crashes or if data becomes corrupt. Another use of the replication is to increase the read bandwidth of the block.

Splitting files in blocks have various benefits for distributed file systems. The first benefit is that a file may be larger than the size of any node in the network. The blocks are smaller and distributed over the nodes in the file system. The second benefit is the predefined size. It permits to know exactly how many blocks each node can store and facilitate the block placement. It also removes the need for storing the metadata (such as file permissions) with the block. Blocks are seen as a chunk of data and their metadata can be managed separately.

NameNode

The roles of the *NameNode* (White, 2009; Shvachko et al., 2010) is to control the access to the HDFS (i.e. opening/closing files) and to determine the mapping of blocks to DataNodes. The NameNode is also responsible for answering client queries. Data never goes through the NameNode, instead it returns to the client the list of DataNodes to which each block has to be written.

Another of the NameNode roles is to maintain the HDFS by keeping its namespace up-to-date. The namespace is a representation of the current files and directories in the file system and their metadata. This namespace is stored on the local disk of the NameNode as two files: the *namespace image* and the *edit log*. (White, 2009; Shvachko et al., 2010) The namespace image captures the state of the file system and the edit log stores all the changes since the last capture.

Secondary NameNode

A secondary NameNode (White, 2009; Shvachko et al., 2010) can be designated for making a checkpoint of the namespace and edit log. This checkpoint consist in applying the changes of the edit log to the namespace image. This prevents the edit log of becoming too large. The new namespace image will be stored on both the primary and secondary NameNodes. In case of failure of the primary NameNode, the last checkpoint can be restored by the secondary NameNode and it becomes the new primary NameNode. Important to note is that in the case of primary node failure, data loss is almost inevitable since the checkpoint occurs only periodically and the changes in the edit log would be lost.

DataNodes

A DataNode (White, 2009; Shvachko et al., 2010) is mainly responsible for the storage of data. It stores two files per block replica, namely, the block itself and

a file containing the block's metadata. This metadata comprise a checksums of the data and a generation stamp. The checksum computed the when the block is stored on the distributed file system and is used to verify the block's integrity. DataNodes are responsible for verifying this checksum.

Communication between DataNodes and NameNode

The NameNode and the DataNodes are always communicating using a heartbeat system(Shvachko et al., 2010; Turkington, 2013). When the DataNode's process starts, it begins by sending a report to the NameNode containing information about its status and blocks. The NameNode responds by sending instructions to the DataNode. Heartbeats sent by the DataNodes consist of statistics about the node and informations about each block stored on it. The statistics describe the state and activity of the DataNode (total storage capacity, percentage storage used and data transfers in progress) important for i.e. the optimization of space or bandwidth load. The information about the blocks sent to the NameNode contains the block ids, their size and generation stamps. This gives the NameNode an accurate report of the blocks replicas in the cluster. The NameNode sends replies to the heartbeats of the DataNode. These replies take the form of instruction maintaining the integrity of the file system. Examples of such instructions are: replication of blocks to other DataNodes, removing block replicates, creation of new block, etc. A heartbeat is sent periodically and signals that the DataNode is operating like expected and that its blocks are available. If the NameNode does not receive the heartbeat of a DataNode after a predefined period of time, the DataNode is then considered as unavailable and the NameNode instructs the replication of its blocks to other DataNodes.

Replica placement

The placement of block replicas (White, 2009; Shvachko et al., 2010) is crucial for data reliability, accessibility and read/write performance. The default HDFS placement policy is a trade-off between minimizing writes cost and maximizing the reliability, availability and aggregate bandwidth. The policy is defined as follows :

1. The first replica is placed on the DataNode of the client.
2. The second replica is placed on another rack.
3. The third replica is placed on the same rack as the second replica but on a different DataNode.

4. Other replicas are placed on random DataNodes.

The advantage of this policy is that the write traffic is reduced and the write performance increased, since the data only needs to go through one switch. Because rack-failure is less probable than node-failure, the policy does not impact the reliability of the file system. Having blocks on two racks guarantee the availability of the blocks. The disadvantages of having two replicas on the same rack is that the bandwidth is reduced.

Replica Management

The NameNode can manages the number of replicas thanks to the heartbeats sent by the DataNodes (Shvachko et al., 2010). It detects when blocks are under/over replicated. When a block is over replicated, the NameNode decides to remove one replicate following policy:

1. If possible, remove replicas from DataNodes on the same racks. Removing a block replica from a rack that has only one replicate would reduce the bandwidth.
2. Remove the block replicas from the DataNode with the least amount of space left.

The NameNode places the replication of under-replicated blocks in a priority queue. If there is only one replica left, the replication gets the highest priority in the queue. Otherwise, the replication gets the lowest priority. The replication policy of under-replicated blocks is similar as the block placement policy:

1. If there is only one replica left, the block is replicated on a different rack.
2. There are two cases when two replicas are left. If the replicas are on the same rack, then the block is replicated on a rack. Otherwise, the block is replicated on a different node in one of the replicas' rack.

The NameNode always tries to spread replicas on different racks.

4.3 Hadoop MapReduce

The MapReduce (Dean and Ghemawat, 2004) is a programming paradigm enabling parallel data processing. It consists of two phases called the *map* and *reduce* phase. The MapReduce program needs to solve a problem. The map phase divides the input data into sub-problems. The reduce phase combines the

output of the map phase and solve the original problem by answering all the sub-problems.

An well-known example is the word count problem. The MapReduce program needs to count the occurrences of every word in a document. The map phase reads the document line by line and breaks them into words. Each word's occurrences per line are counted by the mapper and sent to the reducer. The reduce phase receives all the occurrences of a word per line and accumulates the occurrences for obtaining the total occurrences of the word in the document. The reduce phase solves the word count problem is solved by performing the accumulation on each word in the document.

In this section we detail every step of the Hadoop MapReduce framework (White, 2009) from the initialization of the MapReduce program to its output. Also we describe a Hadoop utility enabling developers of MapReduce programs to adapt code from other programs to work using MapReduce.

4.3.1 Initialization

When a client asks Hadoop to perform a program using MapReduce on data stored in HDFS, a *MapReduce job* is created. This job consists of the three components, namely, the stored data, the MapReduce program and its configurations. The job is divided in two types of tasks: *map* and *reduce tasks*. The map tasks are responsible for dividing the data into sub-problems and the reduce tasks solve the sub-problems.

The execution process of jobs is taken care of by one node called the *jobtracker* and multiple nodes running the tasks called the *tasktrackers*. The jobtracker manages all the jobs and schedules the tasks to the available tasktrackers. The tasktrackers run tasks and send reports describing the task progress to the jobtracker. The jobtracker uses the reports of the tasktrackers as information on the progress of the running jobs. Note the similarity between jobtracker/tasktrackers and NameNode/DataNodes. It is the same principle of one master node managing the other slave nodes. But the similarity does not imply that the jobtracker and the NameNode are on the same node.

4.3.2 Splits as input of the map tasks

An input file must be on the Hadoop Distributed File System before a MapReduce job is created. It is important the file is on the file system since the map task

takes blocks as input. The input of a map task is called splits in the context of MapReduce, even if its data corresponds to the data of a block.

There are multiple reasons why blocks are the ideal splits (White, 2009). The first reason is that the data must be split for parallel processing. Since the file system already splits the data in blocks, blocks seem to be the natural candidates for being the map input. The second reason is that by assigning one block to each map task, it becomes possible to place the computation near the data (on the same node if possible) and decrease considerably the data transfer in the network. This optimization is called the *Data Locality Optimization*. An important note is that the reduce tasks do not benefit of the Data Locality Optimization. Their data is transferred over the network from the multiple maps making Data Locality Optimization impossible to achieve.

4.3.3 The InputFormat

It is important to know that the map tasks need an *InputFormat* for being able to read the data from the split. The split is read by the reader corresponding to the defined InputFormat. For example, the TextInputFormat reader reads the input assuming the data is text. It determines the values of the input data by splitting the input at each newline character.

The reader is also responsible for verifying that the input of the map task is complete. It happens sometimes that the data at the end of a split is not complete, since blocks are split with predefined size. An example of such incomplete data is a sentence that has been split in its middle. The reader detects such abnormalities and completes the split by transferring the data from the next split. This detection prevents errors in the results or computation failures during the MapReduce process. This is also a reason why the term split is used in MapReduce context instead of block. The splits are not always strictly speaking blocks.

4.3.4 Map task, shuffle and reduce task

A map task (White, 2009) consists of applying the map function to the input data. It transforms the values in a sub-problem for the reducer to solve or directly solve the problem if possible. The map function outputs data in a <key, value> pairs format. The output from the map function is processed by the shuffle step before using it as input for the reduce task.

The *shuffle* (White, 2009) is a very important step that makes the MapReduce paradigm simpler by taking care that all the same keys go to the same reducer. The shuffle is built in the MapReduce framework enabling the developer of a MapReduce program to focus only on the map and reduce functions and may assume all the keys will end in the correct reducer. But the developer should always take the shuffle step into consideration. The shuffle process begins in the mappers. Each mapper partitions its <key, value> pairs. All the keys corresponding to a same reduce task are grouped together in one partition. The partitions are sorted when all the keys are in their respective partition. The sort is performed based on the keys. Once the partitions are sorted, each partition is copied to the node running the reduce task that corresponds to the keys in the partition. The reduce task node receives partitions from all the map task node. Those partitions are merged and the output of the merge is used as input for the reduce task.

Likewise to the map task, the reduce task (White, 2009) applies its user-defined function to the data. But now the output needs to be stored on the file system. The block placement policy introduced in 4.2.2 is used as guideline for storing the blocks split from the output.

4.3.5 Different reduce tasks setups

The number of map tasks is defined by the amount of blocks, while the number of reduce tasks is user-defined. The MapReduce job behaves a little bit differently depending on the number of reduce tasks (White, 2009):

No reduce tasks: The mappers store their output directly in the job's output directory.

One reduce task: Each mapper prepares one partition as output. The partition is sorted and copied to the node running the reduce task. The partitions from all the nodes are merged. The merging consists of grouping all the values with the same key together. The result of the merge is used as input for the reduce task. The output of the reduce task is stored on the distributed file system. Figure 4.1 gives a schematic overview of the MapReduce case with one reducer.

Many reduce tasks: Each mapper prepares a partition per reducer. The map tasks store the same keys in the partitions destined for the same reduce task. The partitions of every map task are sorted and transferred to the node running their corresponding reduce task. All the same keys originally from different map nodes are now on the same reduce node. Those partitions are merged and the result of the merge is used as input for the reduce task. The

output of the reduce task is stored on the distributed file system. Figure 4.2 gives a schematic overview of the MapReduce case with one reducer.

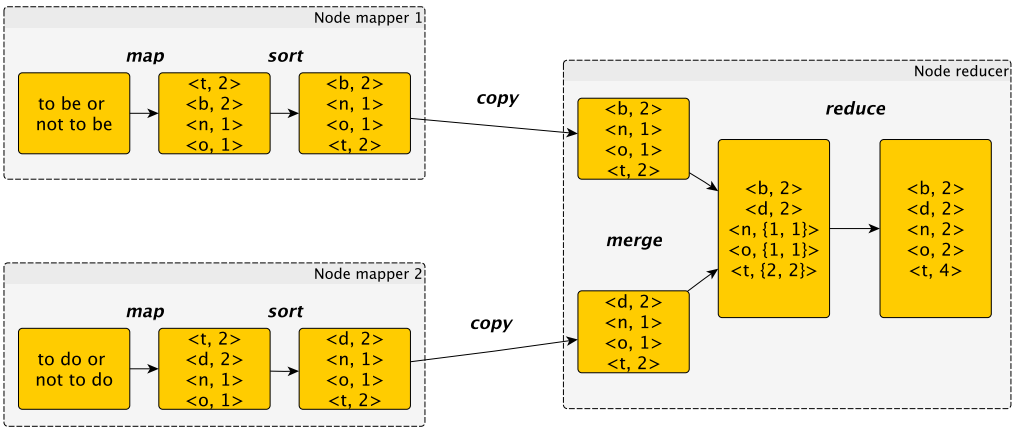


Figure 4.1: The figure describes the case where there is only one reduce task.

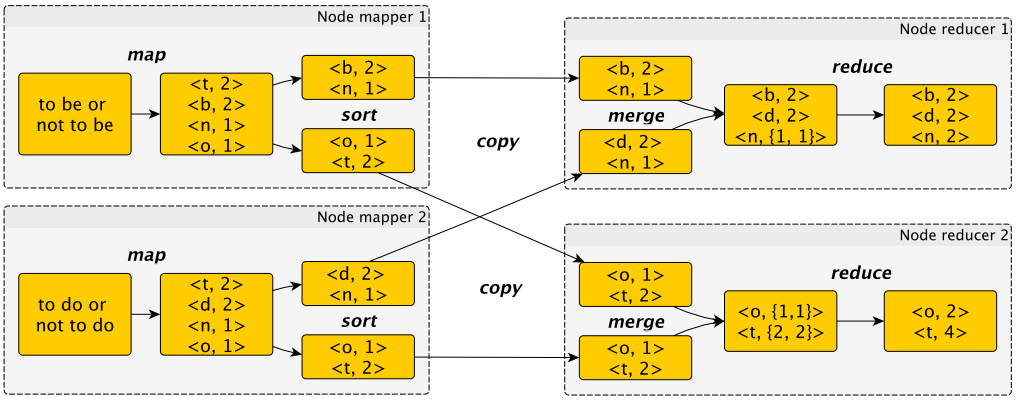


Figure 4.2: The figure describes the case where there are multiple reduce tasks.

4.3.6 Hadoop Streaming

Hadoop MapReduce allows users to write Java programs for parallel processing on the cluster. This implicates that only people with knowledge of Java are able to write MapReduce programs. Hadoop Streaming[1] (White, 2009) is a program included in the Hadoop distribution and enables to run self-made scripts or executables from languages such as C, C++, Python, Ruby, etc. The only constraints

[1] `http://hadoop.apache.org/docs/r0.18.3/streaming.pdf`

are that the code must be able to read the input from the standard input and write its output on the standard output. The user must provide a script or executable for both mapper and reducer. This also means that any program that is parallelizable can be used in combination with Hadoop Streaming for borrowing the computation capacity of Hadoop.

Working of Hadoop Streaming

Like mentioned above, the only condition for Hadoop Streaming to work is that the mapper and reducer must be able to read and write on the standard streams. Creating a Hadoop Streaming job is similar to creating a MapReduce job. The data to process must be stored on the cluster and the job's output will be written in an output directory on the nodes.

The content of a block is placed on the standard input and is streamed as input to the map task. The map tasks has been configured with the given script/executable. It reads the input line per line and outputs the processed line in a <key, value> format, likewise to Hadoop MapReduce. The key is separated from the value by a tab-character. Everything before the tab-character is the key, everything after is the value. If no tab-character is present, then the whole output is considered as the key and the value is set to null.

Likewise to the Hadoop MapReduce, the output of each map task goes through the shuffle process before being reduced. The reduce task has also been configured with the script/executable. It reads the <key, value> pairs streamed by the stdin, performs the reduction script/executable and the reduce task's output is converted in a new <key, value> pair. This pair is then streamed on the standard out and written in its output file in the output directory.

4.4 Summary

In this chapter we described two of Hadoop's major components, namely, the Hadoop Distributed File System and Hadoop MapReduce.

The Hadoop Distributed File System (White, 2009) has been designed for preventing data loss due to hardware failures, data corruption and support large data file efficiently. The HDFS is a file system that manages data across a network of servers. It distributes the data by splitting files in blocks size of 64/128 megabytes. Those blocks are replicated and spread in the cluster. There are two kind of nodes in a HDFS cluster, namely, the NameNode and DataNodes. The NameNode is

the unique master node controlling operations and maintaining the image of the file system called the namespace. A secondary NameNode can be designed for backing up the namespace in case the NameNode crashes. The DataNodes are the worker nodes of the file system. They store the block replicas and check the blocks' integrity when accessed by users. The NameNode and DataNodes are constantly communicating by periodically signals. These communications comprise instructions for the DataNodes, informations about the status of the DataNode and its blocks. An important characteristic of the HDFS is the replication of blocks. The replication is a robust measure for preventing data corruption and data loss. The HDFS has policies for block placement and block replication. Those policies assure the reliability of the Hadoop Distributed File System.

The MapReduce is a programming paradigm enabling parallel data processing. The initialization of a MapReduce (White, 2009) program starts by a MapReduce job being created by a client. The to-process data must be stored on the Hadoop Distributed File system if it is not already on it. The jobtracker looks which tasktrackers are available and assign the map and reduce tasks to them. The map tasks are assigned (if possible) to the same node as the data blocks. This is an optimization reducing the data transfer across the network called Data Localization Optimization. During the execution, the tasktrackers send reports of their progress to the jobtracker for continuing monitoring. A split of the file is assigned to each map task selected by the jobtracker. The map task receives a split with<key, value> pairs as input, transforms the pair for describing a less complicated sub-problem or a direct solution to the problem. The output is also in <key, value> format and is processed by the shuffle step which takes care that all the same keys are send to the same reduce task. The reduce task applies its user-defined function to the pairs and stores the output on the distributed file system.

Hadoop Streaming is an utility provided in the Hadoop installation. It permits to make user-defined functions in the other languages and run them on Hadoop. The only constraint is that the map and reduce functions communicate with Hadoop using the standard input/output.

CHAPTER 5

Methods - Enrichment of an existing genomic platform

The future of medicine is closely related to genomics. Therefore it is important to enable for every biomedical scientist the access to the genomic data. The problem is that genomic data is so huge and has so many different data types that it currently is impossible for a researcher to access all this data. InSilico DB is a tool that aims at giving the means to biomedical experts for analyzing the genomic data at there disposal. We enriched the InSilico DB platform with preprocessing techniques for microarray and RNA-Seq data types.

5.1 Motivation

There are a lot of different data types in genomics. For gene expressions microarray and RNA-Sequencing are the most popular ones. Microarray data can be seen as legacy data since the emergence of RNA-Sequencing. People tend to use more and more RNA-Sequencing but microarray data cannot be put aside because of the millions of dollars that have been invested in research using this technology. Therefore it is important to use methods enabling the combination of these data types.

Another important consideration is that RNA-Sequencing generates large quantities of data that can be classified under the name of Big Data. This data can be so huge that storing and preprocessing it becomes a problem. Notable companies such as Google[1] understood this problem and came up with solutions for the researchers. Their solution is to provide a Big Data platform that manages and computes the clients data using their distributed framework for MapReduce.

The goal of this chapter is to enrich the InSilico DB with methods enabling to use the full potential of the data provided by the platform.

5.2 InSilico DB architecture

InSilico DB is a web-based genomic platform for preprocessed genomic data, counting more than 200.000 gene expression profiles (Coletta et al., 2012). It provides a user-friendly user-interface for managing, curating and integrating datasets for analysis. InSilico DB is a company that originated from a collaborative project between the VUB and ULB.

InSilico preprocesses microarray and next-generation sequencing data in a consistent way by making use of *genomic pipelines*. Taminau (2012, section 4.3.1) gives a good overview of the InSilico DB's genomic pipeline architecture.

Summarized, a genomic pipeline consists of a sequence of *jobs* where each job can be seen as a specialized entity performing a specific action. Since the jobs are performed in sequence, the output of a job is the input of the next job. This implies that jobs are *dependent* on previous jobs. A job can only start if the output of previous job matches its input and that the previous job did not fail. The modularity of the pipeline makes it possible to use different programming languages or executables for each job, thus there are no implementation restrains. Figure 5.1 gives an example of a simple pipeline for generating fRMA datasets with probesets as features.

[1] `https://developers.google.com/genomics/what-is-google-genomics`

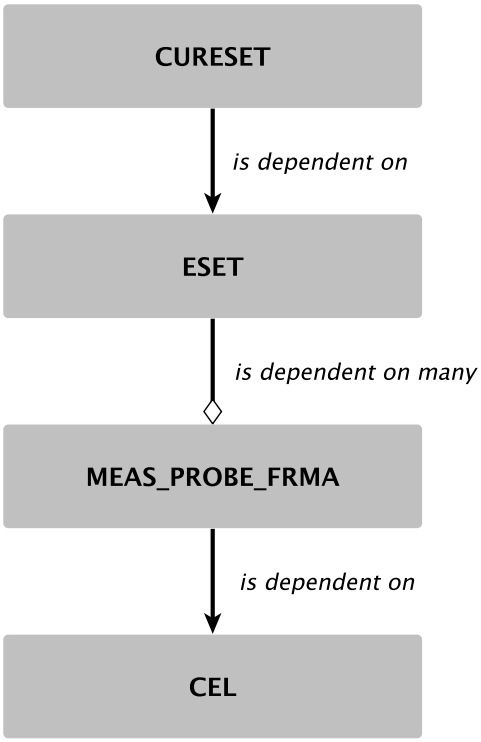


Figure 5.1: Pipeline for generating fRMA datasets with probesets as features. The simple arrows denote a single dependence and the arrows with the white diamond denotes a dependence on many of the same job

5.3 Enriching InSilico DB with the SCAN and UPC preprocessing methods

As mentioned in section 3.1.2, fRMA (McCall et al., 2010) needs a frozen training dataset for being able to preprocess the single sample microarrays. It still is difficult to create such training dataset with the current publicly available samples for all the platforms supported by InSilico DB . Therefore, the SCAN preprocessing method has been added to the already existing preprocessing techniques. SCAN(Piccolo et al., 2012) makes it possible to normalize all the Affymetrix platforms from InSilico DB. This gives the possibility to the user to integrate more platforms in his data-analysis. Another problem of fRMA (and also SCAN) is that it cannot preprocess data from other technologies like RNA-Seq or microarrays that are not Affymetrix. This makes it impossible to combine the data available for the technologies supported by InSilico DB. UPC (Piccolo et al., 2013) makes

it possible to combine microarray data from different manufactures and also next-generation sequencing data. At the moment, UPC has only been integrated for Affymetrix microarray data. In the next sections we explain how InSilico DB has been enriched with a pipeline for the preprocessing of RNA-Seq data with UPC.

The added pipeline jobs for SCAN and UPC are really similar to the existing jobs for microarray preprocessing, namely, `MEAS_PROBE_FRMA` and `MEAS_GENE_FRMA` (Taminau, 2012, section 4.3.1). Their code has been slightly modified for enabling the seamless integration of the new jobs in the microarray pipeline. This also enabled to reuse the code for the parts of the jobs that have the exact same behaviour. The pipeline supporting the added jobs is used by the new version of the InSilico DB package Taminau et al. (2011), making the SCAN and UPC preprocessing available for the users.

The inputs of the job are the microarray measurement file called CEL file and the microarray platform identifier. The first step is to retrieve the correct annotation given the microarray platform identifier. Finding the annotation is more for documenting the resulting ExpressionSet (Gentleman et al., 2005), while the annotation is important in the fRMA job for determining if a fRMA frozen training dataset exists for given platform identifier. Once the annotation is determined, the CEL file is preprocessed using the `SCANfast` or `UPCfast` function (from the `SCAN.UPC` R package(Piccolo et al., 2012, 2013)) depending on the required preprocessing. The next steps are identical to the fRMA job for probes. The information needed for mapping probesets to genes is retrieved and the meta-information is stored as documentation for the users. The result is an ExpressionSet and is stored on the file system. The mapping of probes to genes happens exactly in the same way as described for the gene jobs (Taminau, 2012, section 4.3.1).

It is also possible to give a custom chip definition file (CDF) (Dai et al., 2005) to the `SCANfast` or `UPCfast` fuction. Custom CDFs enable to map probeset identifiers to other identifiers like Ensembl Genes or Entrez Genes. They also take the versions of the genome and transcriptome into account. Ensembl Genes identifiers are useful for the combination of RNA-Seq with microarray data because those are the kind of identifiers used for RNA-Seq. BrainArray[2] provides such custom CDFs for all the platforms supported by InSilico DB. We added such CDF for each platform enabling the mapping of probesets to Ensembl Genes identifiers. The mapping of Ensembl Genes to the genes symbols is also included in the resulting ExpressionSet that is stored on the file system.

[2] `http://brainarray.mbni.med.umich.edu/Brainarray/Database/CustomCDF/CDF_download.asp`

5.4 Enriching InSilico DB with an aligner adapted to work in a distributed environment

In this section we combine the knowledge acquired from Hadoop and the data supported by InSilico DB. The combination of the read sequence aligner with Hadoop Streaming was successful thanks to an example of the adaptation of the Burrows-Wheeler Aligner (Li and Durbin, 2009) using Hadoop Streaming. This example was found in a presentation made by John Farrell (Farrell, 2013). We personally communicated with Farrell on the subject of the file format and inspired us from the example for designing the MapReduce adaptation of Bowtie2.

Parallelization of Bowtie2

Bowtie2 (Langmead and Salzberg, 2012) is a tool for aligning sequence reads to the reference genome or reference transcriptome. It takes as input one single-end or two paired-end FASTQ files for determining the alignments. It is possible to speed the computation up by parallelization since each read is aligned separately. Bowtie2 streams its output on the standard output stream making it possible to adapt Bowtie2 for parallel processing using Hadoop Streaming. Hadoop Streaming is a tool provided by Hadoop that enables to perform MapReduce jobs by streaming data on the standard streams. It supports any programming language or executable that uses these streams, making it the perfect tool for parallelizing Bowtie2. But there is a problem when running Bowtie2 using Hadoop Streaming: the file format.

Problems with the FASTQ format

We explained in the Hadoop chapter that the InputFormat plays an important role in the attribution of the data to the map task splits. The default InputFormat is the TextInputFormat. The TextInputFormat is paired with a reader that checks if the last line in the split is complete. If not, the InputFormat takes care of transferring the missing data to the incomplete record. The problem for the FASTQ records is that they consist of four lines. The probability a record will be wrongly split in one of the first three lines for at least one map task is very high. When one of these lines is split, the InputFile only transfers the missing data for this line. The other lines will still be in the other map split, resulting in a failure of the program.

There are multiple ways to solve such problem. The most intuitive solution is to write a custom InputFormat for FASTQ files. It is certainly the most clean way in

Hadoop to solve the problem. But writing an InputFormat is not straightforward. Also, the custom InputFormat needs to be integrated to Hadoop for enabling the Hadoop Streaming tool to use it. This can be troublesome. Another solution is to write a script that transforms the FASTQ records into records consisting of one line. This approach is very simple to implement and integrates much better with Hadoop Streaming.

Another problem related to the format is that, in the paired-end case, Bowtie2 expects two files. Hadoop does not permit to pass multiple files at a time to a map task. Therefore the paired-end files are stored on the Hadoop Distributed File System as interleaved paired-ends. This format alternates the paired-records from both files. Now the problem is that Bowtie2 does not support the interleaved format. Therefore we convert the interleaved file back to paired-end files during the map task.

Steps before the MapReduce

Now we describe how the script adapting Bowtie2 by using Hadoop Streaming works. The pseudocode for the script running the job is shown in Algorithm 1. The first step is to detect if the input is single-end or paired-end. If it is paired-end, then the files are first interleaved before the records are transformed so that both records are represented as one line. In case it is a single-end file, the file is only transformed so that each record is represented as one line. The trick for representing a record as oneline is to use a separator for each newline character in the record. The separator we use for replacing newlines is eight times the ASCII character x. The character x is not used in the sequence or qualities (see 2.3.1) making it a safe character for separating the lines. But the character x can still be present in the identifier. The repetition of the character should reduce the probability that it is encountered in the file to a really small number. Also, the oneline-transformation is a little bit different for the paired end. It transforms paired-records to one line instead of transforming single-record. This is needed for paired-end processing of Bowtie2. The result of the oneline-transformation is stored on the Hadoop Distributed File System. Once the file is stored on the HDFS, the MapReduce job is created. The MapReduce job looks like depicted in Figure 5.2.

The map task

The input of a map task is a split of onelined-records. The first action is to undo the oneline-format for retrieving the original records. This is easily done by replacing

the separators by newline characters. For the single-end case, the records are streamed to Bowtie2. For the paired-end case, the file is split in two paired-end files and saved in a temporary directory of the machine. Those files are passed as parameters to Bowtie2 for paired-end processing. Bowtie2 aligns the sequences to the reference genome and prints the resulting alignments on the standard out stream.

There are some important observations that need to be mentioned before explaining the reduce phase. For each map task a Bowtie2 program has been executed and prints the results on the standard output. This has two consequences. The first consequence is that for each execution of Bowtie2 a header containing meta-information has been created. This means that for each mapper, the first lines printed on the standard output stream are meta-information. This meta-information is necessary and must be the first lines of data in the output file. The second consequence is that Bowtie2 prints alignments and not <key, value> pairs. In Hadoop Streaming, a value is treated as a key when no key is specified. Thus the outputted alignment becomes a key and is associated with the value nil. These two observations are crucial for choosing the number of reduce tasks.

The reduce task

The reduce task is very simple because it is the UNIX command `cat`. This may seem odd because `cat` only prints the data, but this is also what is needed since the map task has done all the computation for the alignments. In chapter 4 we explained the different configurations depending on the number of reduce tasks. There are three possibilities for configuring the number of reduce tasks for the MapReduce job. These different configurations are not trivial and are explained using the observations made in previous paragraph:

What if the MapReduce job is configured with many reduce tasks? Then the alignments of each map task, which are keys, would be distributed over the partitions for the shuffle. The partitioning is determined by the *Partitioner* of Hadoop (White, 2009). The Partitioner is a more advance feature of the MapReduce framework. Rewriting a custom Partitioner could be a possible solution. Also, using many reduce tasks would generate many outputs that need to be combined into one output. There exist an Hadoop command for this purpose called `getmerge`. This seems to be the fastest option since the reduce tasks are also in parallel, but a custom Partitioner needs to be written.

What If the MapReduce job is configured without reduce task? Then the out-

puts of the map tasks are directly stored on the Hadoop Distributed File System. No shuffle will take place, thus the partitioning problem is bypassed. The resulting outputs of the map tasks can be concatenated using the same `getmerge` command. The concatenation has for result that the headers are spread across the file. This is undesirable since they must be above in the file.

What If the MapReduce job is configured with only one reduce task? The outputs of the map tasks would be stored in one partition, removing the partitioning problem. The keys are sorted and merged during the shuffle phase and the result is the input of the reduce task. Since all the keys from all the mappers are merged together, all the headers will be grouped together in the output. Because headers start with the ASCII @ character and the alignments with their identifier (which are letters and number, not special characters), the headers will be the first lines in the output file.

It is clear from the description of the three configurations that the simplest solution is to use one reduce task. It is not the fastest job configuration but the command `cat` is not a computationally expensive task (unlike for example Bowtie2) making the need for parallelization less important. This configuration has also two big advantages. The first advantage is that the Hadoop optimized shuffle mechanism sorts and merges the alignments in the desired order. The other and perhaps the most important benefit is that it frees all the used cores from the map tasks except one. If fifty map tasks were needed for the job, then only one remains for the reduce task and forty-nine cores are now available for new map tasks.

Correcting the header of the output file

The result form the MapReduce job is a sam file(Li et al., 2009). A sam file contains data about alignments to the reference genome. The sam file is copied from the HDFS to the InSilico DB file system. The sam file is converted using samtools to a bam file (Li et al., 2009), which is a binary version of the sam file taking less space in the file system. The bam file headers generated during the map tasks are corrected. This is necessary because a lot of those headers are duplicates and may cause problems with certain programs. The correction simply removes the duplicated headers. The result of the script adapting Bowtie2 for using Hadoop Streaming is a header corrected bam file.

```
input : fastq_files // list with one or two FASTQ files
        map_single_end // path to single-end script
        map_paired_end // path to paired-end script
        path_HDFS // path on HDFS for input file
        separator // separator for oneline function
        output_path // path on HDFS for output file
output: bam_file // Header corrected bam file

fastq_end1 = fastq_files [1];
fastq_end2 = fastq_files [2];

/* Check if paired or single end case                    */
if isNull (fastq_end2) then
    /* Case single-end                                   */
    onelined = onelineFastq (fastq_end1, separator);
    saveHDFS (onelined, path_HDFS);
    MapReduce (path_HDFS, map_single_end, separator, output_path);
else
    /* Case paired-end                                   */
    interleaved = interleaveFastq (fastq_end1, fastq_end2);
    onelined = onelineFastq (interleaved, separator);
    saveHDFS (onelined, path_HDFS);
    MapReduce (path_HDFS, map_paired_end, separator, output_path);
end
/* Retrieve the result from Hadoop and correct the
   headers                                               */
bam_file = correctHeader (outputMapReduce (output_path));
```

Algorithm 1: Pseudocode representing the outline of the script running the MapReduce task.

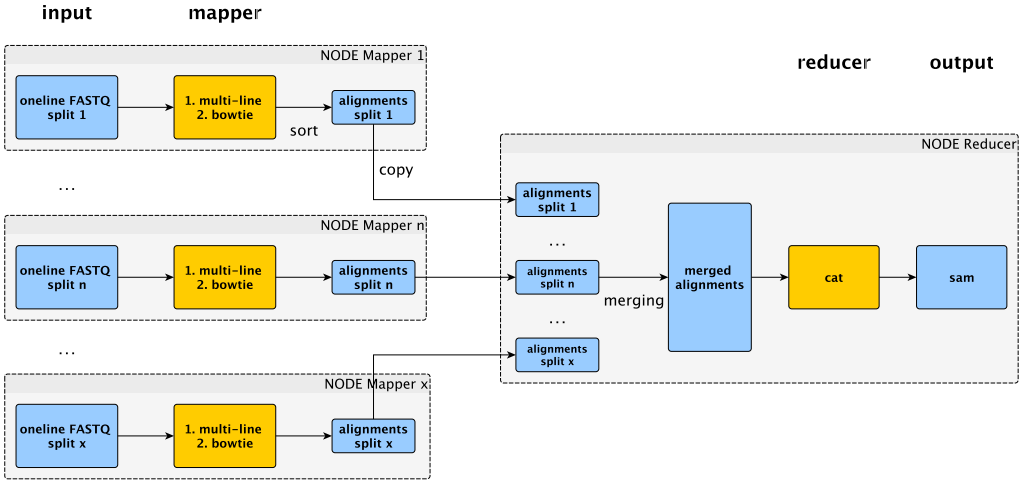


Figure 5.2: MapReduce job created for Bowtie2. Is configured with only one reduce task. The input of each map task is a onelined-FASTQ file and the output of the reduce task is a sam file.

5.5 Benchmark Bowtie2 using Hadoop

We benchmarked Bowtie2 working on Hadoop by comparing its performance with the performance of Bowtie2 using one and eight cores. Eight cores is the maximum amount of cores the tool can be configured with. Each experiment is configured with the same parameters except for the Bowtie2 experiment with eight cores. The experiments have been ran ten times each and their average time is shown in table 5.1. The measurement used for this benchmark is GSM661224 from study GSE20898 (Wei et al., 2011). The FASTQ file is 4.0 gigabyte large and the onelined-FASTQ file is 4.4 gigabyte large. The number of cores (map tasks) is 35 since 4.4 gigabytes divided by 128 megabytes (block size of the distributed file system) is 35.

algorithm	#cores	$\hat{time}$
Bowtie2	1	826 min 34 sec
Bowtie2	8	146 min 32 sec
Bowtie2 + Hadoop	35	27 min 13 sec

Table 5.1: Performance of the Bowtie2 with one and eight cores against the Bowtie2 adapted version that uses Hadoop Streaming. The experiments have been ran 10 times each configured with the same parameters (except for the number of cores).

Note the difference of performance between Bowtie2 with one and eight cores is not linear, it is 5-6 times faster. The same conclusion can be made for Bowtie2 working on Hadoop but there are other computations (i.e. the shuffle and reduce task) taking place. Bowtie2 working on Hadoop is 30 times faster than the single core Bowtie2 experiment and 5 times faster than Bowtie2 at its maximum capacity of cores. The Hadoop version of Bowtie2 is thus a little bit less than the number of mappers faster than a single core Bowtie2 program. This means that if enough machines are provided, very large files can be processed very quickly. Of course, this is when the MapReduce framework provides its full capacity to one job. If someone runs for example 100 MapReduce jobs on a Hadoop cluster having ten 12-cores machines, the MapReduce jobs will not have a good performance because each job would approximatively have one map task working. This could even result in worse performance with the shuffle and reduce steps included.

5.6 Enriching InSilico DB with a genomic pipeline for UPC preprocessing of RNA-Seq data

The goal of the pipeline is to preprocess RNA-Seq data with UPC so that combining RNA-Seq and microarray data becomes possible. The constructed pipeline consists of following jobs:

- `DOWN_SRA`
- `FASTQ_DUMP`
- `FASTQ_PREPROCESSING`
- `BOWTIE2_HADOOP`
- `EXPRESS`
- `MEAS_ENSEMBL_UPC`
- `ESET`
- `CURESET`

The `BOWTIE2_HADOOP`, `EXPRESS` and `MEAS_ENSEMBL_UPC` jobs have been added to the InSilico DB backbone with the purpose of preprocessing RNA-Seq using UPC. Those are the jobs we implemented. The other jobs were already implemented by the InSilico DB team and are used in their genomic pipelines. We reused these jobs in our experimental genomic pipeline.

The Download SRA step retrieves the necessary Sequence Read Archive (SRA) files from a public repository. Those files are the archives obtained from the next-generation sequencing experiment for the dataset and contain the FASTQ files.

The FASTQ Dump step extracts FASTQ files from the downloaded SRA archives. The result of the FASTQ Dump are one or multiple FASTQ files. Those are stored on the file system.

The inputs of the FASTQ Preprocessing job are the FASTQ files extracted by the FASTQ Dump job. Those are merged and preprocessed into one single-end or two paired-end files in three steps:

1. Merge corresponding FASTQ files into one single-end or two paired-end files.
2. Performs a quality control check using the fastqc[3] software. The quality control check is necessary for being sure of the correctness of the qualities.
3. Detects which quality format is used in the FASTQ file and converts it to the Sanger format if needed.

The result are corrected FASTQ files ready to align using Tophat Trapnell et al. (2009), Bowtie2 (Langmead and Salzberg, 2012) or other sequence aligners.

The `Bowtie2_HADOOP` job gets a single or paired-end input. The input is placed on the Hadoop Distributed File System and computed by creating a MapReduce job. The map task will be the computation of the alignments with Bowtie2 (Langmead and Salzberg, 2012) and the reduce task is simply the UNIX cat command. The result of the MapReduce job is a sam file (Li et al., 2009). A sam file contains a header with meta-information and alignments. This sam file is converted to its binary version using the samtools software (Li et al., 2009). This bam file is the output of the job.

The transcript-level in the header corrected bam file is quantified using the eXpress software (Roberts and Pachter, 2013). EXpress is a similar tool as Cufflinks (Trapnell et al., 2010), but its computation time is proportional to the number of read sequences and the memory requirement is proportional to the size of the reference genome/transcriptome used. One of the values determined by eXpress are the raw read counts for each transcript region. Those are the values needed by UPC for RNA-Seq preprocessing.

Each sample is preprocessed in a `MEAS_ENSEMBL_UPC` job by applying the function `UPC_RNASeq` from R package `SCAN.UPC` (Piccolo et al., 2012, 2013) to

[3] `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`

the raw read counts obtained in previous step. The features of the resulting ExpressionSet are the Ensembl Transcript identifiers and not gene symbols. Those identifiers need to be converted to Ensembl Gene identifiers before combining RNA-Seq datasets with microarray datasets. An RData file is stored on the file system for each sample. A RData file stores objects from the R language as binary data.

The `ESET` job combines the resulting RData files from the `MEAS_ENSEMBL_UPC` jobs into one non-curated ExpressionSet. `CURESET` add the biological curation to the ExpressionSet. The result of the `CURESET` job can be downloaded and used in analysis in combination with other datasets, for example microarray datasets with Ensembl Gene identifiers as features.

5.7 Summary

In this chapter we explained the integrations made to the InSilico DB platform for the microarray and RNA-Seq genomic pipelines. These additional jobs have for goal to enable new preprocessing methods for microarray and RNA-Seq data.

Jobs have been added to the microarray pipeline for supporting the SCAN and UPC preprocessing methods. Each preprocessing method has been added for returning gene expression values for three possible kind of identifiers, namely, probesets, gene names and Ensembl Genes identifiers.

The RNA-Seq pipeline for UPC preprocessed gene expressions is summarized by Figure 5.3. Three jobs have been added to the existing ones for being able to make such pipeline, namely, the `BOWTIE2_HADOOP`, `EXPRESS` and `MEAS_ENSEMBL_UPC` jobs. The `BOWTIE2_HADOOP` runs the Bowtie2 aligner using Hadoop to speed up the computation. The result is used by the `EXPRESS` for determining the read counts for each transcriptional region. These counts are used in `MEAS_ENSEMBL_UPC` for obtaining UPC normalized gene expressions.

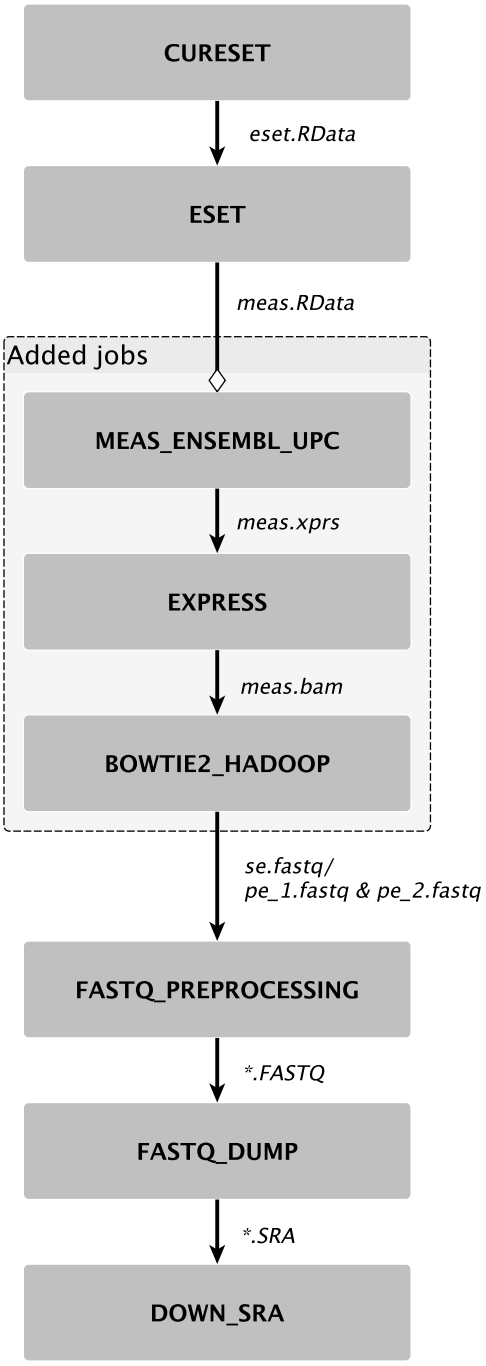


Figure 5.3: Pipeline for preprocessing RNA-Seq data with UPC. `BOWTIE2_HADOOP` uses Hadoop Streaming for faster Bowtie2 computation. `EXPRESS` calculates the raw read counts from the output Bowtie2. The read counts are used as input for the `UPC_RNASeq` function for obtaining UPC values for each measurement. The arrow notation is the same as in Figure 5.1

CHAPTER 6

Results

In this chapter we apply the methods developed in previous chapter for data analysis. First we validate the preprocessed data. The combination of microarray and RNA Sequencing data is validated in using the same approach. Then we benchmark the preprocessing methods for determining if the combinations of preprocessing and batch effect removal methods have a similar performance or if some combinations are much better than others. Once we analyzed the benchmarking experiment, we apply our knowledge for developing a method aiming at discovering differentially expressed genes. The method is applied on the different preprocessing methods and on the combination of microarray and RNA-Seq data.

6.1 Data used in the experiments

In this section we present the datasets used in this chapter. The datasets we use are referenced in table 6.2. Taminau (2012) put a group of datasets together suitable for merging and used this group in his analysis. He obtained a large list of differential expressed genes (DEGs) from merging these datasets (Taminau, 2012, p. 128 and pp. 132–133). Since we also need a suitable group of datasets for merging and we also do an experiment for discovering DEGs we decided to use

Dataset	Platform	#Genes	#Samples (control/cancer/unknown)	Reference
GSE10072	GPL96	12496	107 (49/58/0)	(Landi et al., 2008)
GSE7670	GPL96	12496	66 (27/27/12)	(Su et al., 2007)
GSE31547	GPL96	12496	50 (20/30/0)	No publication
GSE19804	GPL570	19851	120 (60/60/0)	(Lu et al., 2010)
GSE19188	GPL570	19851	156 (65/91/0)	(Hou et al., 2010)
GSE18842	GPL570	19851	91(45/46/0)	(Sanchez-Palencia et al., 2011)
total:			590 (312/266/12)	

Table 6.1: Overview of the lung cancer microarray datasets used during the experiment. This table is based on the table from (Taminau, 2012, p. 123)

Dataset	Platform	# Ensembl Gene Ids	#Samples (control/cancer)	Reference
GSE6956	GPL571	11939	89 (20/69)	(Wallace et al., 2008)
GSE22260	GPL9115	39293	30 (10/20)	(Kannan et al., 2011)
total:			119 (30/89)	

Table 6.2: Overview of the microarray dataset GSE6956 and RNA-Seq dataset GSE22260 for prostate cancer.

the same datasets. The list of DEGs will be used as reference for comparing our results and will be called *DEGs Taminau*. The number of genes is different in our datasets since the version of the annotation package used for mapping probeset ids to gene ids has changed since Taminau's work.

For datasets aiming at combining microarray and RNA-Seq we decided to use two datasets. The RNA-Seq dataset was generated using the custom genomic pipeline and the microarray dataset is preprocessed with UPC for making enabling the combination. An overview of the datasets is given in table 6.1.

6.2 Visual validation of batch effect removal on different preprocessing techniques

In this section we visualize the results from applying batch effect removal to the different preprocessing techniques. The visualization should help to validate the result of batch effect removal on the preprocessing techniques.

6.2.1 Visualization methods

We use one visualization methods for validation of the merging on the data, namely, Principal Component Analysis (PCA) (Jolliffe, 2002). PCA applies a orthogonal transformation to samples transforming the possible correlated variables into uncorrelated variables called principal components. The first principal component is the variable having the largest variance and must be orthogonal to the second principal component. In its turn the second principal component has the second largest variance and is orthogonal to the following component and so on. The number of principal components is always smaller or at most equal to the number of correlated variables. This means that PCA reduces the dimensionality of the data.

Only the first two principal components are plotted for validation of the results, since those are the components that capture the most variance and that the greatest source of variance is the due to the difference in batch and not due to the biological difference of the samples (Leek et al., 2010). A batch effect removal method is considered effective if the combined batches are overlapping when plotting the first two principal components of the samples (Lazar et al., 2012). The plot function used is the `plotMDS` implemented in the InSilicoMerging R package (Taminau et al., 2012).

6.2.2 Visual Validation of fRMA, SCAN and UPC

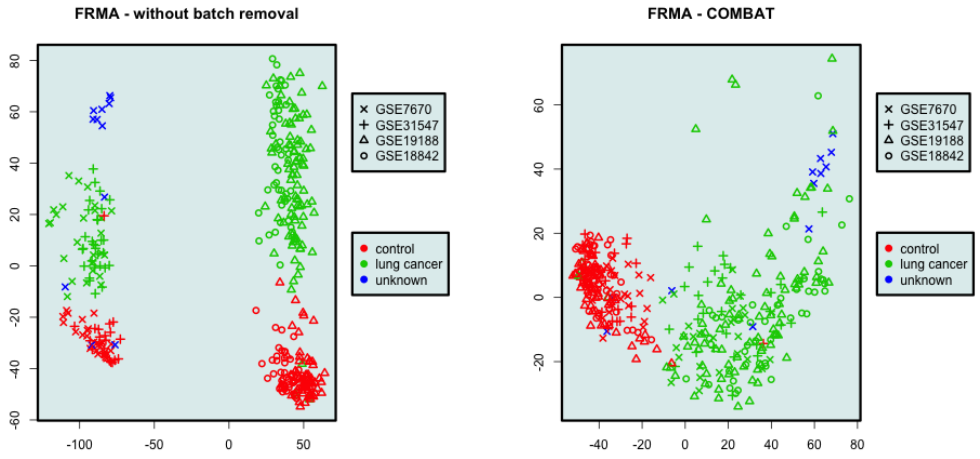
Preprocessing techniques try to remove the possible variations in microarray experiments by:

- background correction for removing variations due to the hybridization
- normalization for removing variations due to different experiment settings
- summarization by using models that take the probe/batch effect and measurement errors into account.

These three steps are effective for removing the within-batch variation and are sometimes able to remove the batch effect of datasets from the same microarray platforms. We notice in figures 6.1a, 6.2a and 6.3a that the samples are well merged for the three preprocessing techniques when it comes to samples of the same platform. But the studies are still subjected to batch effect. The different platforms GPL96 and GPL570 are still dissociable.

We use the ComBat (Johnson et al., 2007) batch effect removal method on the different preprocessing techniques. The difference is directly visible on the four

datasets. The resulting figures 6.1b, 6.2b and 6.3b show clearly that batch effect is removed for the three preprocessing techniques. The three classes are well separated from each other while the samples from the four studies are undeniably merged.

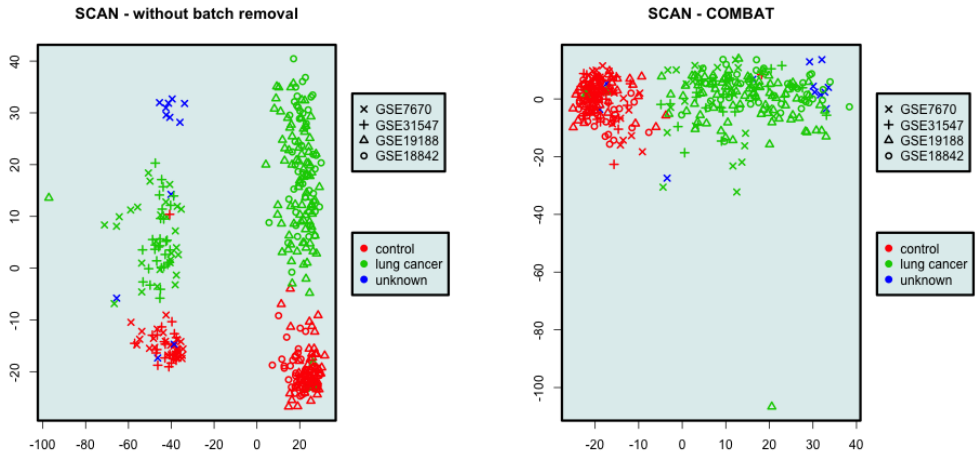


(a) fRMA without batch effect removal (b) fRMA with ComBat

Figure 6.1: Validation of batch effect removal on fRMA preprocessed samples

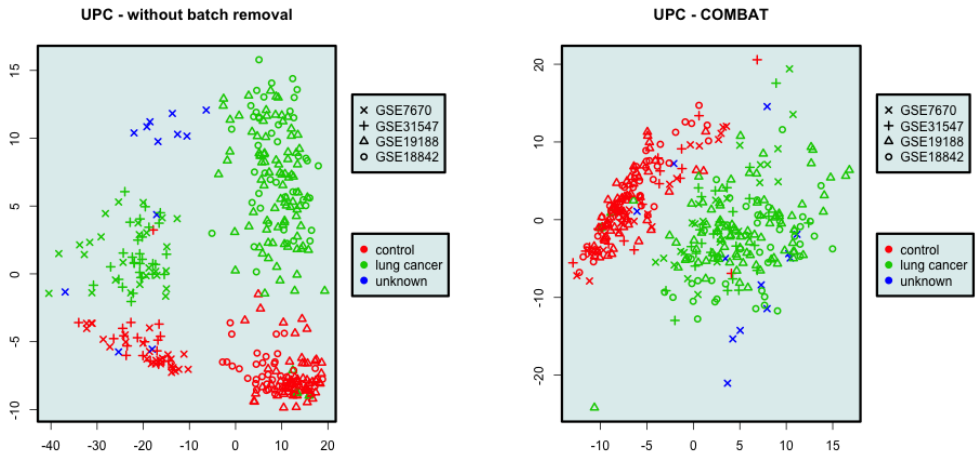
6.2.3 Visual validation of the merging microarray and RNA-Seq data.

The merging of microarray and RNA-Seq data is validated using the same method. The data is preprocessed using the UPC technique. Three batch effect removal methods have been applied on the two datasets. Figure 6.4 shows the different results obtained for each batch effect removal technique. Figure 6.4a shows clearly that there is batch effect in the samples. A lot of the batch effect is most likely coming from the difference in data types. In Figures 6.4b, 6.4c and 6.4d we notice a clear amelioration in comparison with Figure 6.4a. Since the samples overlap in the three figures, we may conclude that a large amount of the batch effect seems to be removed. Unlike the lung cancer datasets, the control and cancer samples are mixed on the PCA plots after batch effect removal. But on these figures it is at least possible to imagine a hyperplane separating the controls from the cancers. This is much more complicated for the figure without batch effect removal, implying that performing an analysis on these values would fail.



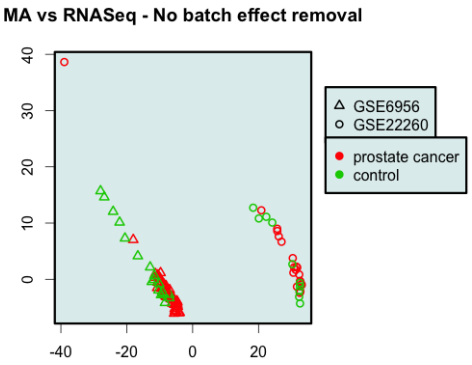
(a) SCAN without batch effect removal (b) SCAN with ComBat

Figure 6.2: Validation of batch effect removal on SCAN preprocessed samples

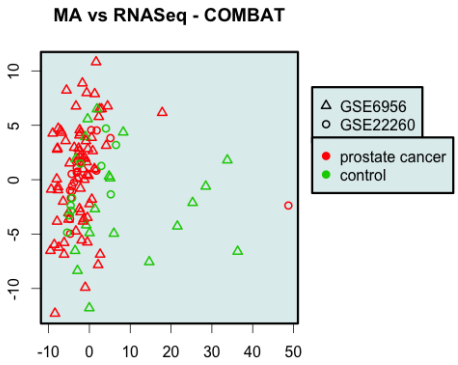


(a) UPC without batch effect removal (b) UPC with ComBat

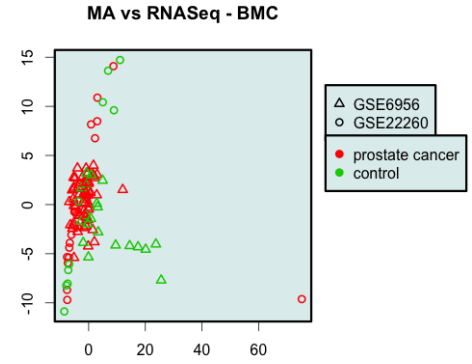
Figure 6.3: Validation of batch effect removal on UPC preprocessed samples



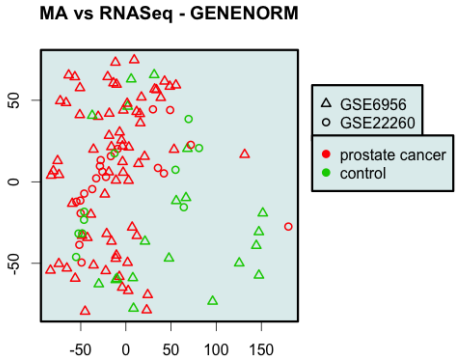
(a) No batch effect removal



(b) Batch effect removal using ComBat



(c) Batch effect removal using BMC



(d) Batch effect removal using GENENORM

Figure 6.4: PCA plots representing the merging of a microarray and a RNA-Seq dataset using batch effect removal methods.

6.3 Benchmarking the combination of batch effect removal and preprocessing methods

An important question for benchmarking batch effect removal methods is: ”What is a good batch effect removal method?”. A good batch effect removal method has two characteristics. First, it removes the batch effect from the data. Second, the classes for the target variable must still be differentiable after batch effect removal. For example in Figures 6.3 we see that ComBat seems to be a good batch effect removal method since it merges the data such that it became impossible to distinguish a group of samples that comprises only elements of the same platform. Whereas grouping the samples from platform GPL96 is very easy without batch effect removal. The ComBat method also preserves the biological difference between the disease classes. We see that the cancer samples from both platforms now form one distinct group of cancer. The same observation can be made for controls. Summarized, a good batch effect removal method should transform the data such that it is easy to separate the data when looking at the variables of interest and hard to separate when looking at the variables related to batch effect.

Given the two characteristics of a good batch effect removal method, we can now determine which method is better than the others by measuring how complex it is to separated the data after batch effect removal. Therefore we use the transformed gene expressions of a batch effect removal method as training data for decision trees. The complexity of the generated tree is used as measure. One tree is generated for each characteristic. We assume that a decision tree for classifying i.e. disease will have a low complexity for good batch effect removal methods. The other assumption is that a good batch effect removal method will have a very complex decision tree when build for classifying a variable related to the batch effect i.e. studies or platforms.

6.3.1 The rpart Decision Tree

For the classification of the datasets we use the decision tree implemented in the *rpart* R package (Therneau et al., 2014). The rpart classification tree is constructed by searching the attribute that splits the data in the best two sub-trees. Once this attribute has been found, the data is split in two sub-trees according to the attribute. This process is repeated for each sub-tree until a stop criterion is met (i.e. predefined number of element in the sub-tree) or that the sub-tree only consists of one class.

In rpart, the best attribute is measured by means of an impurity function (Therneau

et al., 1997). In other words, a function measuring how diverse the elements of the node are. The function f used by rpart for measuring the amount of diversity in a node N is:

$$I(A) = \sum_{i=1}^{C} f(p_i N) \tag{6.1}$$

$p_i N$ is the proportion of elements of class i in node N, where f is the gini index $f(p) = p(p-1)$. The attribute that splits the best the data is the attribute that maximizes the impurity reduction (Therneau et al., 1997):

$$\Delta I = p(N)I(N) - p(N_L)I(N_L) - p(N_R)I(N_R) \tag{6.2}$$

where N_L and N_R are the two sub-trees of node N.

6.3.2 Benchmarking

We determine the complexity of the decision tree by multiplying two factors: the total number of nodes in the tree and the depth of the tree. The choice of the first criterion is motivated by the fact that complex tree structures have more nodes than simple decision trees structures. The criterion of depth is for taking the flatness of the tree into account. In general, a deeper tree indicates a higher difficulty for the classification of the data.

The benchmarking consists of a comparison between the combinations of batch effect removal method and the preprocessing methods. We generated for each combination 20 trees: 10 for the studies and 10 for the diseases. Ten percent of the samples were left out the classification for generating the decision trees. The left out samples were randomly selected. The results of the benchmarking experiment are depicted in Figure 6.5 and are discussed in following remarks.

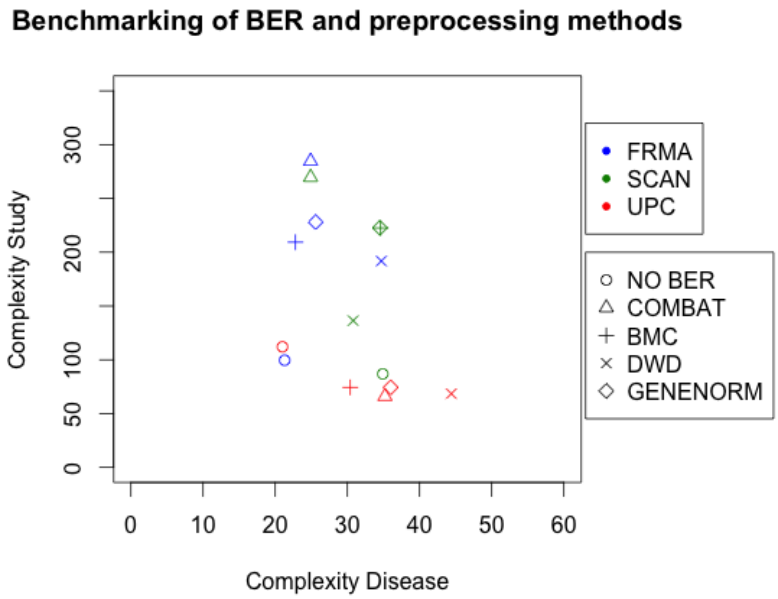


Figure 6.5: The figure depicts the complexity of the generated decision trees for all batch effect removal and preprocessing techniques combinations for Studies and Diseases. The complexity measure is defined as the number of nodes in the tree multiplied by the depth of the tree.

Remark 1: Baseline combinations

The baseline combinations are the trees generated using data that has not been transformed by batch effect removal. They have been added as reference point for the complexity of decision trees for studies. They also have a low complexity for diseases, meaning they classify the diseases very well. This might seem odd, but in Figures 6.1a, 6.2a and 6.3a it is clear that the diseases are separated from the controls. It is possible to draw an hyperplane that separates them even if there is a gap between the platforms, this implying there might exist genes able to split the two groups easily even without batch effect removal.

Remark 2: Performance fRMA

FRMA is the preprocessing technique that combines the best with batch effect removal methods for the datasets. All its combinations with batch effect removal methods have an high complexity for the decision trees classifying the studies and a low complexity for the decision trees for diseases. A possible explanation is that since fRMA accounts for batch-effect in its method by using the frozen components. The batch effect is reduced for data on the same platform. This may have a positive influence on the gene expressions across platforms since they are supposed to be the same.

Remark 3: Performance UPC

UPC clearly does perform worse after batch effect removal. This is strange since the visual validation of UPC showed that the batch effect was effectively removed using ComBat. Our hypothesis is that the values of the UPC preprocessing method are not suited for this benchmark. A UPC value represents the probabilities that a gene is expressed above the background for the sample(Piccolo et al., 2013) . The values are in an interval between 0 and 1, since the values represent probabilities. The UPC preprocessing method returns for a lot of genes one of the endpoints of the interval. If we consider the case that all the batches share the exact same value for a gene (i.e. the probability 1) and that only one batch is affected by batch effect for this gene, then applying batch effect removal will change only the UPC value of the batch affected by batch effect. An example is given in Figure 6.6, were the ACTB gene has always probability 1 for each dataset before batch effect removal and is slightly changed for dataset GSE19188. The difference is almost not visible, but it exists and is exploited by the decision tree. The result is that decision trees easily trains on such genes and generates a simple tree for studies.

It is clear in Figure 6.7 that the decision tree is constructed using genes having endpoint value (in this case 1) that were affected by batch effect.

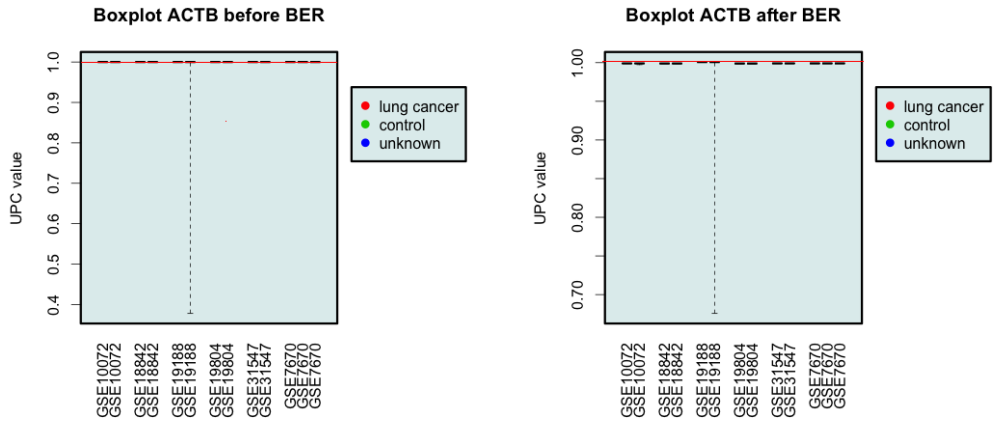


Figure 6.6: Gene ACTB before and after batch effect removal. The datasets were preprocessed using UPC. The batch effect was removed by using ComBat. The plots were generated using the `plotGeneWiseBoxPlot` function from the InSilicoMerging package.

Remark 4: Performance ComBat

ComBat scores the best of all batch effect removal methods when combined with SCAN and fRMA. An explanation of the effectiveness of ComBat compared to the other batch effect removal methods can be that ComBat removes batch effect in three steps (Johnson et al., 2007). It first standardizes the gene expression values of the gene, such that they have an equal mean and variance (which is similar to BMC or GENENORM). It has two additional steps that are the estimations of batch effect factors and the adjustment of the gene expressions using these estimations. These additional steps removes extra batch effect, which is not done in other methods.

Remark 5: Performance DWD

DWD seems to be the batch effect removal method performing the worst. It has a higher complexity for studies for FRMA and UPC and a lower complexity for disease for SCAN. DWD can only combine two datasets at once, thus the combination of the 6 datasets was performed by recursively calling the DWD function (Lazar et al., 2012). We suspect that the iterative combination of datasets makes it

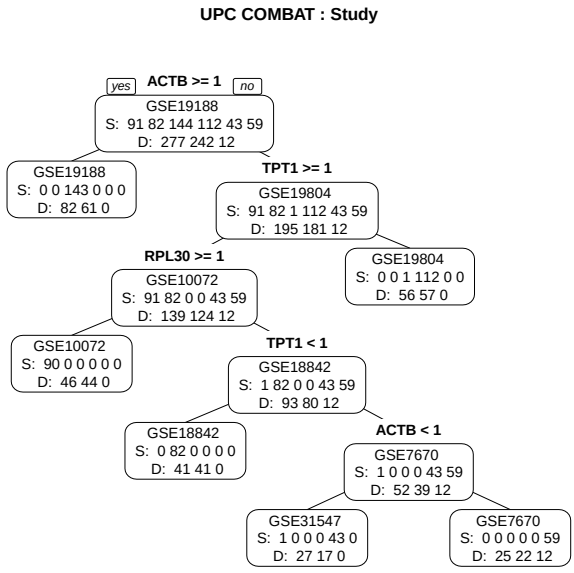


Figure 6.7: The decision tree classifies the samples by their study for the UPC preprocessing technique combined with the ComBat batch effect removal method. The attributes splitting the samples in subgroups are on top of each node. The label in the node states the class with the most samples in the node. The second line states the number of samples for each study. The order of the classes are: GSE10072, GSE18842, GSE19188, GSE19804, GSE31547 and GSE7670. The last line in the node denotes the number of samples for each disease. The order of the classes are: lung cancer, control and unknown.

more difficult to remove all the batch effect. DWD removes batch effect by combining two datasets and projecting the samples in the direction of the hyperplane separating the batches. If there are four datasets, then the datasets are merged two by two. The two resulting batches are combined in their turn and form the batch effect free dataset. We suspect that if there is still a little bit of batch effect after the first merge, the batch effect may scale with the values in the later iterations. Also when DWD is applied, the order of the datasets is important, a different order results in different values after the preprocessing. We ran DWD for the same samples in different orderings and got two different results. The results are visible in Figures 6.8 and 6.9.

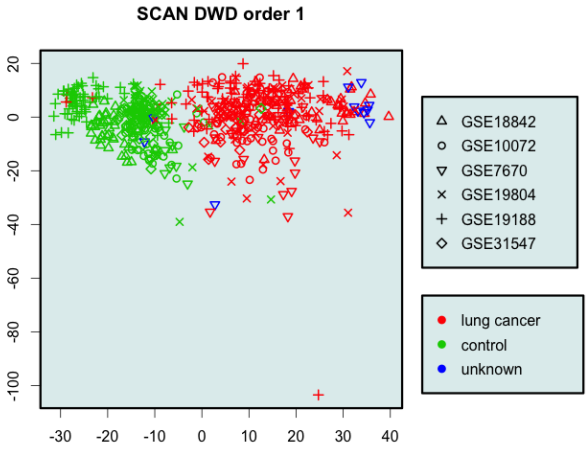


Figure 6.8: DWD applied on SCAN preprocessed datasets with the following order: GSE10072, GSE7670, GSE31547, GSE19804, GSE19188, GSE18842

Remark 6: Performance GENENORM and BMC

GENENORM and BMC have an equivalent performance for both FRMA and SCAN. This is not surprising since those methods are very similar. GENENORM standardizes the mean and standard deviation of the genes (mean = 0 and standard deviation = 1), when BMC only standardizes the mean (mean = 0).

Remark 7: Best combination for the datasets

The best combination form our benchmarking experiment on the selected datasets is the combination of FRMA and COMBAT. The combination of SCAN and ComBat also has a good performance.

6.4 Stability of the generated decision tree

In this section we look more in details the stability of the generated decision trees. It is important to be aware if some genes are more often chosen as attributes when the datasets are merged. Also the larger the sample size the better the population of gene expressions for lung cancer are represented. This may imply that the gene

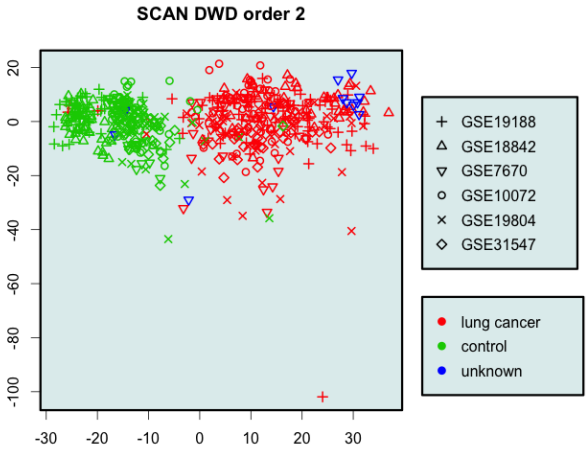


Figure 6.9: DWD applied on SCAN preprocessed datasets with the following order: GSE10072, GSE19804, GSE7670, GSE19188, GSE31547, GSE18842

is important for the classification of diseases and thus could be of significance for lung cancer.

Here we use two resources to verify if a gene can be important, namely, *genecards*[1](Rebhan et al., 1997) and the DEGs determined by Taminau. Genecards is a database that references genes and proteins with their related diseases. The experiment consisted of generating 10 decision trees for classifying diseases and count the occurrences of every genes. We referenced in the tables the genes occurring at least 5 times in the 10 trees for the datasets or combinations. If there were no genes occurring at least 5 times, then we referenced the gene occurring the most in the experiments. Plots have been generated for every experiments and can be found in the Appendix.

Stability decision trees

We notice in Table 6.3 that the decision trees generated for individual datasets only have one recurring gene. When we clearly note multiple recurring genes per combination in most of the generated decision trees. The FRMA method seems to be less stable than the other preprocessing methods. This information is important and will be taken into consideration when performing experiments. The other two preprocessing techniques are more stable, but the most stable genes are not always

[1] `http://www.genecards.org`

Dataset	Gene	# in tree	lung cancer related	in DEGs Taminau
FRMA				
GSE10072	RNF38	3	no	no
GSE7670	AHNAK	9	no	no
	AKAP12	6	yes	no
GSE31547	RNF38	5	no	no
GSE19804	SPOCK2	8	no	yes
GSE19188	STX11	7	no	no
GSE18842	ASPM	4	no	yes
SCAN				
GSE10072	VWF	7	yes	no
GSE7670	CFP	7	no	no
	AGR2	5	yes	no
GSE31547	ICA1	4	no	no
GSE19804	SPOCK2	7	yes	no
GSE19188	RILPL2	6	no	no
GSE18842	ASPM	7	no	yes
UPC				
GSE10072	CAV1	8	yes	yes
GSE7670	AGER	8	yes	yes
GSE31547	PTRF	8	yes	no
GSE19804	SPOCK2	8	no	yes
	ADIRF-AS1	6	no	no
GSE19188	ADH1B	4	yes	yes
GSE18842	ASPM	6	no	yes

Table 6.3: References the genes that occurred the most when generating the decision trees for individual datasets. These genes are assumed to be the most stable ones.

Batch effect removal	Gene	# in tree	lung cancer related	in DEGs Taminau
FRMA				
No BER	CAV1	7	yes	yes
	SOX4	5	yes	no
ComBat	CAV1	5	yes	yes
BMC	CAV1	5	yes	yes
GENENORM	CAV1	6	yes	yes
	A2M	6	no	no
DWD	AGER	6	yes	yes
	A2M	6	no	no
SCAN				
No BER	BST2	9	no	no
	AGER	6	yes	yes
	CLDN18	5	yes	yes
ComBat	AGER	6	yes	yes
BMC	A2M	8	no	no
	CLIC5	6	yes	yes
GENENORM	A2M	10	no	no
	AGER	5	yes	yes
	CLIC5	5	yes	yes
DWD	AGER	9	yes	yes
	ASPA	6	no	no
	A2M	6	no	no
UPC				
No BER	AGER	10	yes	yes
	SERINC2	5	yes	no
ComBat	AGER	10	yes	yes
	MMP19	6	no	no
	EFNA4	5	no	no
BMC	AGER	10	yes	yes
	ANP32A	8	yes	no
	MMP19	7	no	no
GENENORM	AGER	8	yes	yes
	EFNA4	7	no	no
	A2M	6	no	no
	PPAP2C	6	no	no
DWD	AGER	10	yes	yes
	ADH7	10	no	no
	KPNB1	6	no	no

Table 6.4: References the genes that occurred the most when generating decision trees after merging the datasets and applying batch effect removal on them. These genes are assumed to be the most stable ones.

the gene that are biologically relevant (i.e. the A2M gene for the combination of SCAN with GENENORM).

For individual datasets it seems like GSE7670 is the most stable dataset except for UPC. This is surprising since it is one of the datasets with the least number of samples. The other stable dataset is GSE19804. We observed that the genes occurring the most in the decision trees generated for these datasets were not often related to lung cancer nor categorized as DEGs by Taminau. Only SPOCK2 occurred multiple times for the same dataset when using different preprocessing method. Since SPOCK2 is in the differentially expressed list of Taminau, we conclude that it is an important gene in this dataset. This also shows that the three preprocessing methods are equivalent when it comes to important genes for individual datasets. This can also be noticed in table 6.4, were AGER is used at least one time as attribute in the decision trees for every preprocessing technique.

Genes related to lung cancer or in DEGs Taminau

Tables 6.3 and 6.4 show clearly that merging different datasets forces the decision tree to find attributes that are more biologically relevant. The genes AGER and CAV1 are almost always selected in every preprocessing/batch effect removal method combination as being the most stable genes. Both are known to be related to lung cancer and are in the list of differentially expressed genes discovered by Taminau (2012). If we look at some of the generated decision trees we can conclude that AGER and CAV1 are the root of the trees that is generated. This implies that these genes split the diseases in cancer and control very well. We see in Figures 6.10 and 6.11 two of the many examples that those genes are used for splitting the diseases in two groups and thus are genes with a possible biological influence on cancer. We chose to show the combination of SCAN and GENENORM since A2M is a very stable gene, but is neither related to lung cancer nor in the differentially expressed genes list of Taminau. A2M occurs 10 times but never as splitting attribute of the root node.

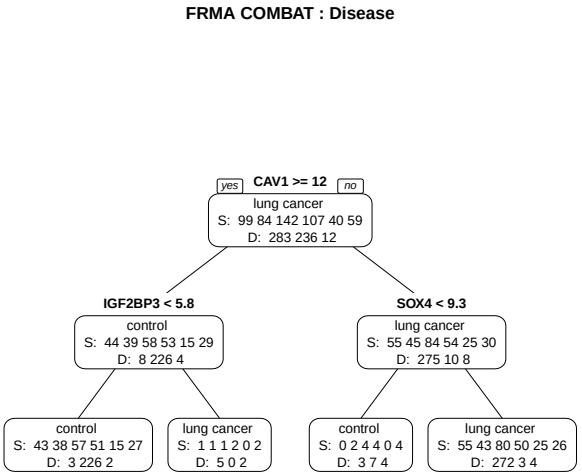


Figure 6.10: The decision tree classifies the samples by their diseases for the FRMA preprocessing technique combined with the ComBat batch effect removal method. The attributes splitting the samples in subgroups are on top of each node. The label in the node states the class with the most samples in the node. The second line states the number of samples for each study. The order of the classes are: GSE10072, GSE18842, GSE19188, GSE19804, GSE31547 and GSE7670. The last line in the node denotes the number of samples for each disease. The order of the classes are: lung cancer, control and unknown.

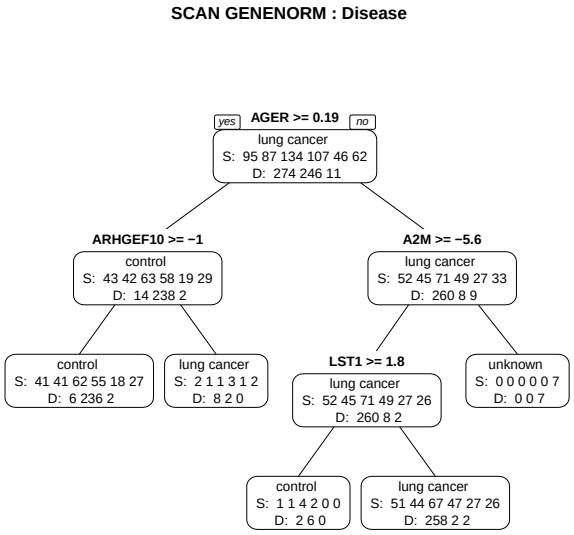


Figure 6.11: The decision tree classifies the samples by their diseases for the SCAN preprocessing technique combined with the GENENORM batch effect removal method. The attributes splitting the samples in subgroups are on top of each node. The label in the node states the class with the most samples in the node. The second line states the number of samples for each study. The order of the classes are: GSE10072, GSE18842, GSE19188, GSE19804, GSE31547 and GSE7670. The last line in the node denotes the number of samples for each disease. The order of the classes are: lung cancer, control and unknown.

6.5 Discovering differentially expressed genes

Motivated by the results of previous section, we try to discover the best genes for splitting the merged datasets in cancer and control samples. Such genes are called *differentially expressed genes* (DEGs). Differentially expressed genes are important because they give insight to researchers which genes are potentially responsible for the tested disease. Better drugs may be produced if the gene is confirmed to be a cause of the disease.

First we explain the metrics used for finding the best attributes. These metrics are used in a method that evaluates if a gene is differentially expressed or not. We end the section with two experiments that applies the method on the lung cancer and prostate cancer datasets.

6.5.1 Measures for finding attributes that split the data efficiently

The entropy (Shannon, 1948) is a measure of homogeneity for a group of samples N:

$$Entropy(N) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{6.3}$$

where c is the number of different classes and p_i is the proportion of elements of the group of samples ^{N} that are of class i (Mitchell, 1997). It is now possible to define the information gain of an attribute using the formula of Entropy. The information gain of an attribute denotes its effectiveness for splitting samples in homogeneous groups. The information gain (Mitchell, 1997) of attribute A for the group of samples N is determined by:

$$InformationGain(N, A) = Entropy(N) - \sum_{v \in Values(A)} \frac{|N_v|}{|N|} Entropy(N_v) \tag{6.4}$$

where $Values(A)$ is the set of unique values the attribute can take and N_v are the elements in node N that have value v.

6.5.2 Determining differentially expressed genes

Our approach consists of analyzing each gene at a time and determining if it is differentially expressed or not. The original values were gene expressions preprocessed by either FRMA, SCAN or UPC. A DEG should be differentially expressed

in the transformed gene expressions of every batch effect removal method. Therefore we apply a supervised method on each of the transformed gene expressions and combine the results.

The idea is to iterate over the list of genes and apply our supervised method to each gene g one by one. The input of the method is the gene specific N x M transformed gene expressions matrix E, where N is the number of samples and M are the number of batch effect removal methods. The element on position (n, m) represents the transformed gene expression for sample n using batch effect removal method m for gene g. The result is a list of genes with an corresponding score. The higher the score, the more likely the gene is a DEG.

The score of gene g is calculated by applying a function that determines the maximum information gain of each column in matrix E. This means that the maximum information gain is determined for the gene expressions of each batch effect removal method one by one. The score is equal to the mean of the maximum information gains of each batch effect removal method.

The maximum information gain function we imagined consists of four steps:

1. Arrange the samples and their corresponding classes in increasing gene expression
2. Evaluate if the gene is a differentially expressed gene
3. Determine candidate attributes.
4. Calculate the maximum information gain

Below we explain each step more in details. At each step we added a short example for making the idea behind the method the clearest possible. The example consists of 10 samples. Each sample has a corresponding gene expression and class. The gene expressions have been transform by one of the batch effect removal methods before the application of the method. There are two possible classes : L_1 and L_2. The example dataset S is:

Sample:	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Gene Expression:	0.7	0.8	0.3	0.6	0.9	0.5	1.1	0.4	1.2	1.0
Classe:	L_1	L_1	L_1	L_1	L_2	L_2	L_2	L_2	L_2	L_2

Sample rearrangement

The first step is to order the samples and their corresponding gene expression by increasing gene expression.

Sample:	S_3	S_8	S_6	S_4	S_1	S_2	S_5	S_{10}	S_7	S_9
Gene Expression:	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Class:	L_1	L_2	L_2	L_1	L_1	L_1	L_2	L_2	L_2	L_2

Evaluation if the gene is differentially expressed.

The evaluation step is not necessary, but it results in a huge gains of time. Only a small subsets of genes are DEGs and a lot of them can be eliminated just by looking at the homogeneity of the classes after the sample rearrangement. Therefore the ordered samples are split in their middle, resulting in two equal groups of samples. If a gene is differentially expressed, then a large amount of cancer or control samples should be in one of the two splits (since the samples are ordered by gene expression). The entropies of the two splits are measure the homogeneity of the splits. A homogeneous split has a low entropy and a heterogeneous split has a high entropy.

Once the entropies have been determined, their values are compared to a threshold. The threshold is defined by $maxEntropy(\#classes) * h$, where h represents a homogeneity factor between 0 and 1. The lower the homogeneity factor, the faster the method determines the DEGs. Of course, the gain in computation time comes at the expense of accuracy. A low homogeneity factor permits only very homogeneous splits to pass the threshold. If the threshold is passed, then the gene is evaluated as being a potential DEGs. Otherwise, the gene is evaluated as being not enough differentially expressed for further computation and its resulting information gain for the batch effect removal method is 0.

Sample:	S_3	S_8	S_6	S_4	S_1	S_2	S_5	S_{10}	S_7	S_9
Gene Expression:	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Class:	L_1	L_2	L_2	L_1	L_1	L_1	L_2	L_2	L_2	L_2

$$Entropy(Group_1) = -\tfrac{2}{5}log_2\tfrac{2}{5} - \tfrac{3}{5}log_2\tfrac{3}{5} = 0.9709506$$
$$Entropy(Group_2) = -\tfrac{1}{5}log_2\tfrac{1}{5} - \tfrac{4}{5}log_2\tfrac{4}{5} = 0.7219281$$

There are two classes in our example thus the maximum entropy is: $maxEntropy(2) = log_2(2) = log_2 2 = 1$. The homogeneity threshold is equal to the homogeneity factor h in this example. If the homogeneity factor is lower than 0.7219281, then the gene would not be evaluated as being a DEG. Otherwise, the gene would pass the threshold and continue the next steps of the computation.

Determining candidate attributes

When values are ordered it is possible to make good guesses of candidate attributes. The candidates are the samples that have at least one neighbour with a different class for the target variable. It is shown that the maximum information gain lies between two samples with different classes in a dataset (Fayyad, 1992; Mitchell, 1997). In our method we consider the gene expressions of the samples that have a neighbour with a different class as one of the candidate attributes.

Most of the time cancer and control samples are still mixed in the ordered dataset, even if they passed the homogeneity step. It is not uncommon to have more than 100 candidate attributes. After some experiments we noticed that the candidates in the middle of the datasets are most of the time the ones that return the maximum information gain. This motivated us for adding an additional reduction step for the candidate attributes. The parameter r controls the reduction of the list and is a value between 0 and 1 that represents the percentage of candidates to keep. For example: if the threshold is 0.5, then 25 percent of the candidates at both ends of the list are removed. This parameter speeds the computation a little bit up and does not reduce the accuracy if the parameter is well chosen.

Sample:	S_3	S_8	S_6	S_4	S_1	S_2	S_5	S_{10}	S_7	S_9
Gene Expression:	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
Class:	L_1	L_2	L_2	L_1	L_1	L_1	L_2	L_2	L_2	L_2

In our example the samples having neighbours with different classes are S_3, S_8 , S_6, S_4 S_2 and S_5. Their gene expressions are considered as being candidate attributes. In the rest of the example we consider the reduction parameter r = 1, meaning that all the thresholds will be used.

Calculating the maximum information gain

The last step is to calculate the information gain for all the remaining attributes. The maximum information gain obtained in this step is the result for gene g in combination with the selected batch effect removal method.

The information gain is calculated for all the candidate attributes (r = 1):

$$
\begin{aligned}
InformationGain(S, g > 0.3) &= 0.1444843 \\
InformationGain(S, g > 0.4) &= 0.0074034 \\
InformationGain(S, g > 0.5) &= 0.0058022 \\
InformationGain(S, g > 0.6) &= 0.0058022
\end{aligned}
$$

$$InformationGain(S, g > 0.8) = 0.4199731$$
$$InformationGain(S, g > 0.9) = 0.4199731$$

The maximum information gain in this example is 0.4199731 for attributes$g > 0.8$ and $g > 0.9$. If r was a small value, the result would have been completely different, therefore it is important to choose a good r (not too small and not too large if possible).

6.5.3 Experimental setup

We performed four experiments by using the full potential of the enriched In-Silico DB architecture. The first three experiments are very similar microarray experiments that only differ in the preprocessing method that is used. The last experiment uses RNA-Seq data in combination with microarray data.

The above described method is applied 20 times to the data and every time we leave a random subset of 10 percent out of the computation. Our data are the gene expressions from the four batch effect removal methods, namely, ComBat, BMC, GENENORM and DWD. In the last experiment we do not use the DWD batch effect removal method. The final result is the intersection of the 100 bests genes for every application of the method. We decided to do 20 runs since FRMA did not seem very stable compared to the other preprocessing methods. By taking the intersection we make sure that only genes that splits the data well are in our results. We run the method on each preprocessing method (fRMA, SCAN and UPC) separately. The microarray and RNASeq experiment uses the UPC preprocessing technique.

The homogeneity parameter we use is $h = 0.4$, resulting in a computation time of approximatively 3-4 minutes per run (less than a minute per batch effect removal method). With $h = 0.5$ a run is immediately much longer.

The reduction parameter r We performed correlation tests between the results of the the method with r = 0.5 and r = 1. The correlation was always above 0.99 with a p-value smaller than 2.2^{-16}. Therefore we used 0.5 as reduction parameter.

6.5.4 Results experiments

The results of the microarray experiments are similar for all the preprocessing techniques. Therefore we reference first the list of discovered differentially ex-

pressed genes for each preprocessing method and then explain some conclusions about the discovered genes.

Resulting differentially expressed genes for the experiment using the fRMA preprocessing technique:

```
 [1] "ABCA8" "ACADL" "ADAMTS8" "ADARB1" "ADH1B" "ADIRF"
[7] "ADRB2" "AGER" "AOC3" "ASPA" "CA4" "CAV1"
[13] "CAV2" "CD36" "CD93" "CDH5" "CFD" "CLDN18"
[19] "CLIC5" "COL10A1" "COX7A1" "DENND3" "DLC1" "DOCK4"
[25] "DPEP2" "DUOX1" "EDNRB" "EMCN" "EMP2" "EPAS1"
[31] "FABP4" "FAM107A" "FAM189A2" "FCN3" "FGFR4" "FHL1"
[37] "FHL5" "FOXF1" "FXYD1" "GGCT" "GPM6A" "GRIA1"
[43] "GRK5" "HIGD1B" "HN1L" "JAM2" "KAL1" "KANK3"
[49] "KCNK3" "LAMP3" "LDB2" "LIMCH1" "LIMS2" "MFNG"
[55] "PAFAH1B3" "PECAM1" "PTPRB" "PYCR1" "RAMP2" "RAMP3"
[61] "RASIP1" "RGCC" "S1PR1" "SASH1" "SDPR" "SGCG"
[67] "SLC39A8" "SPOCK2" "SPTBN1" "STARD13" "STARD8" "TACC1"
[73] "TCF21" "TGFBR3" "TIE1" "TNNC1" "TNXB"
```

Resulting differentially expressed genes for the experiment using the SCAN preprocessing technique:

```
 [1] "ABCA8" "ACADL" "ADAMTS8" "ADAMTSL3" "ADH1B" "ADIRF"
[7] "AGER" "AOC3" "ARHGAP6" "ASPA" "CA4" "CAV1"
[13] "CD36" "CD93" "CDH5" "CFD" "CFP" "CLDN18"
[19] "CLIC5" "COL10A1" "DENND3" "DLC1" "DOCK4" "DPEP2"
[25] "EDNRB" "EFCC1" "EMCN" "EMP2" "EPAS1" "ERG"
[31] "FABP4" "FAM107A" "FAM189A2" "FCN3" "FHL1" "FHL5"
[37] "FOXF1" "FXYD1" "GPM6A" "GPX3" "GRIA1" "GRK5"
[43] "HIGD1B" "JAM2" "KANK3" "KCNK3" "LDB2" "LIMCH1"
[49] "LIMS2" "MFNG" "PECAM1" "PRX" "PTPRB" "PYCR1"
[55] "RAMP2" "RAMP3" "RASIP1" "RGCC" "ROBO4" "S1PR1"
[61] "SDPR" "SGCG" "SPOCK2" "SPTBN1" "STARD13" "STX11"
[67] "TACC1" "TAL1" "TCF21" "TEK" "TMEM100" "TNNC1"
[73] "TNXB"
```

Resulting differentially expressed genes for the experiment using the UPC preprocessing technique:

```
 [1] "ABCA8" "ACADL" "ADAMTS8" "ADIRF" "AGER" "ARHGAP6"
[7] "CA4" "CASKIN2" "CDH5" "CLIC5" "COL10A1" "DENND3"
```

```
[13] "DUOX1" "EDNRB" "FABP4" "FAM107A" "FAM189A2" "FCN3"
[19] "FOXF1" "FXYD1" "GIMAP6" "GPM6A" "GRK5" "HIGD1B"
[25] "HMGB3" "KANK3" "KCNK3" "KIAA1462" "KL" "LDB2"
[31] "LIMS2" "LINC00312" "LYVE1" "MELK" "NPR1" "PDLIM2"
[37] "PYCR1" "RAMP2" "RAMP3" "RASIP1" "S1PR1" "SLC2A1"
[43] "SLIT2" "SPOCK2" "SPP1" "STARD13" "TCF21" "TEK"
[49] "TMEM100" "TNNC1" "TOP2A" "WIF1"
```

Each of these set of genes have been entered in the *Enrichr* API (Chen et al., 2013) for analyzing the co-expression of the list of differentially expressed genes. The analysis using Enrichr gave some interesting results back. The API noticed for each group of DEGs that it was mainly composed of genes typically expressed in lung cells. For fRMA 25 genes are related to lung cells, 23 for SCAN and 20 for UPC.

The differentially expressed genes for fRMA and SCAN were linked to significant biological pathways. *A biological pathway is a series of actions among molecules in a cell that leads to a certain product or a change in a cell* (National Human Genome Research Institute, 2014). The three most significant pathways linked to the genes for fRMA and SCAN are the leukocyte transedonthelial migration, PPAR signaling pathway and the cell adhesion molecules. From these three pathways the PPAR signaling pathway and the cell adhesion molecules have already been related to lung cancer. The genes related to these two pathways are ACADL, CD36 and FABP4 for the PPAR signaling pathway and JAM2, CLDN18, CDH5 and PECAM1 for the cell adhesion molecules. The only significant pathway linked to the differentially expressed genes obtained in the UPC experiment is the PPAR signaling pathway. Only this time there are only two genes related to this pathway, namely, ACADL and FABP4.

The expression of the CD36 gene is directly related to the activation of PPARγ (Nagy et al., 1998). PPARγ is one of the subtypes of the PPAR signaling pathway and is linked by different studies to lung cancer (Tsubouchi et al., 2000; Chang and Szabo, 2000). CDH5 is a cell adhesion molecule of the Cadherin family, a family that has already been related to lung cancer (Smythe et al., 1999). CDH5 has also been reported with PECAM1 as being possible genetic markers for non-small cell lung carcinoma, which is a type of lung cancer (Tran, 2013). This enumeration of relations to lung cancer has for purpose to show that the methodology has positive results for the fRMA and SCAN preprocessing techniques. We investigated only the genes related to the significant pathways which represent a small subset of the discovered DEGS.

We found more DEGs in the experiments using the fRMA and SCAN preprocess-

Preprocessing	#genes	#genes in DEGs Taminau
fRMA	77	37
SCAN	73	32
UPC	52	26

Table 6.5: Number of differentially expressed genes found for each experiment and the number of gene present in the DEG reference for the datasets.

ing methods than for the experiments using the UPC preprocessing method. It is clear that our method has positive results for these two preprocessing methods, it is less clear for UPC since only one pathway was linked to resulting DEGs and that the gene playing an important role in this pathway was not present anymore. When comparing the DEGs found by our method with the results of Taminau (2012), we notice that the half of the genes in our lists are not present in his results. Our method captures perhaps other relevant genes and/or the simplicity of using information gain is not powerful enough to discover some of the differentially expressed genes in the merged dataset. Determining the cause of the differences could be part of future work on the method. The number of genes are referenced in Table 6.5.

Microarray and RNA-Seq experiment

Resulting differentially expressed genes for the experiment combining microarray and RNA-Seq data:

```
 [1] "AP003068.6" "C16orf59" "CACNG3" "DBNDD1" "DCAF4" "DLX2
[7] "DYRK1B" "FANCA" "FGF14" "KCNQ4" "KLK15" "LPPR3" "MAPK8IP2"
[14] "MATK" "MFSD7" "MUC2" "NR2C2" "OGT" "ONECUT2" "OTUB2"
[21] "RIMS1" "SCNN1D" "SLC13A4" "TBX1" "TPO"
```

We proceeded the same way for analyzing the results obtained by the combination of microarray and RNA-Seq data. Note that the amount of DEGs is much lower in this example, probably due to the smaller number of samples. We gave the list of differentially expressed genes as input to the Enrichr API. This time the API did not find the correct origin of the genes, but it found one very significant pathway for prostate cancer, namely, the MAPK signaling pathway. The genes related to this pathway are CACNG3, FGF14 and MAPK8IP2. The activation of the MAPK pathway is known to increase with the progression of prostate cancer (Gioeli et al., 1999).

6.6 Conclusions

In this chapter we used the methods implemented in the InSilico DB platform for analyzing the 6 lung cancer datasets and 2 prostate cancer datasets. We first validated the use of batch effect removal on the each preprocessing method by applying ComBat on the datasets. The batch effect free datasets were visualized by applying Principal Component Analysis on the gene expressions and plotting the first two principal components. We did not observe any problems with the results. The same approach was used for validating the results of applying batch effect removal on the combination of microarray and RNA-Seq data. The result was that batch effect seemed to be removed from the data. The diseases were not as well split as for the lung cancer datasets, but batch effect removal does not remove biological information. This means that the prostate cancer data has less differences in the gene expression values for prostate cancer and controls.

Our second step was to benchmark the results of the different combinations of batch effect removal and preprocessing techniques. Our benchmarking approach uses decision trees for measuring how well samples are separated for the diseases and studies after batch effect removal. A complex decision tree indicates that it is difficult to separate the samples and vice versa. Studies should always be difficult to separate after batch effect removal, meaning their decision tree should be complex. The inverse is true for diseases, they should be easily separable and resulting in simple decision trees.

Therefore the lung cancer datasets are classified on their diseases and studies for building the decision trees. The results of the benchmarking indicates that the preprocessing technique performing the best for our criteria is fRMA. The SCAN preprocessing technique also performs very well for most batch removal methods, except for DWD. We concluded that UPC was not suitable for our benchmark because of its probabilistic values. For the batch effect removal techniques we concluded that ComBat performed the best because it estimates the batch effect factors and adjusts the gene expression values using these estimates. GENENORM and BMC performed similarly and DWD was still influenced by batch effect. Those results are specific to the lung cancer datasets we used, but the methodology can be applied to other datasets.

The analysis of the stability of the trees showed that genes having a possible biological importance were most of the time the roots of the decision trees classifying the diseases. This information motived us to try an experiment for finding differentially expressed genes. We used the information gain as metric for differentiating the gene expressions. For each gene we used the combined information gain of the different batch effect removal methods for finding if a gene is differentially

expressed. Some optimizations have been made in the method for speeding up the computations. One example of such optimizations is that the homogeneity of the samples is checked before starting the computation. The gene is not considered a DEG if it is not homogeneous enough.

The method was applied to each preprocessing technique for microarray data and to the combination of microarray and RNA-Seq data. We did a brief research for finding possible biological relevance in the list of differentially expressed genes we obtained. We found that some genes in the list were co-expressed for multiple pathways related to lung cancer. The analysis of the microarray and RNA-Seq data was also successful since we found genes related to a pathway that is linked to prostate cancer. Even if the analysis in of microarray and RNA-Seq data was rather small-scale, it was important for us to show that the experimental pipeline we build in previous chapter makes UPC values for RNA-Seq data that is mergeable with microarray data. This experiment also proved that it is feasible to analyse both legacy microarray data with newer RNA-Seq data in one experiment and find biological relevant information.

CHAPTER 7

Conclusions

7.1 Problem Description

One of the biggest challenges in genomics is to make the data of experiments available to people having the expertise to interpret the data. These people generally do not have the IT background for taking care of all the data preparation steps enabling the data interpretation. It has been years now that scientist use microarray technology for experiments. Millions of dollars have been invested in this technology and cannot be thrown away because of the emergence of new technologies such as RNA Sequencing. Therefore it is crucial to find methods to cope with both data types, otherwise microarray data will be lost.

Therefore solutions have been made for enabling the people with the expertise to be able to interpret the results. InSilico DB is an example of such solution. It provides multiple services such as a user-friendly web interface for the combination and analysis of the processed and biologically curated data. This makes it possible for biologists and biomedical experts to easily analyze the data without being faced with the difficulties that must be tackled by Computer Science.

7.2 Contributions

7.2.1 Enriching the InSilico DB Platform

We enriched the InSilico DB platform with new preprocessing techniques enabling the analysis of lots of new data. The addition of the SCAN preprocessing technique to InSilico DB permits to analyze thousands of Affymetrix microarray samples that were not supported by the fRMA preprocessing method. This enables researchers to perform cross-platform analysis with those samples.

We enriched InSilico DB with another preprocessing technique called Universal exPression Code(Piccolo et al., 2013). This preprocessing technique enables to combine microarray and RNA-Seq data. New technologies keep emerging, but old technologies and data types need to be combinable in experiments otherwise this data is lost. UPC enables to combine various data types, among others microarray and RNA Sequencing data.

The last addition to the InSilico DB platform is the experimental RNA-Seq pipeline for UPC preprocessing. The problem with RNA-Seq data is that it consists of huge files of sequence reads and that their alignment takes a lot of time. Therefore the Bowtie2 aligner has been integrated to a distributed environment for speeding up the computation. Apache Hadoop is the distributed environment that has been used for processing and managing RNA-Seq Big Data. Hundred samples have been preprocessed in less than a week when normally it should have taken almost one month.

7.2.2 Benchmarking

After having the preprocessing methods included, we assessed the results of the different combinations of preprocessing and batch effect removal methods. We benchmarked the results using the complexity of generated decision trees. The target variables for growing the trees are respectively the studies and the disease. The more complex a decision tree for studies, the better the combination performs. The simpler the decision tree for the diseases, the best the combination removes the batch effect on the data. The combination of the assumptions defines a good combination as being a combination that makes it difficult to retrieve the original studies of the samples and that preserves their biological information while removing the undesirable variance.

7.2.3 Model for finding DEGs

Motivated by the observations of the results obtained when analyzing the stability of the decision trees generated for the benchmarking experiment, we decided to make a simple method for finding differentially expressed genes. The method is based on the the principle of information gain and searches the genes that splits the best the combination of gene expressions for all batch effect removal methods. Parameters have been added to the method for enabling a faster computation. For assessing the method we ran it 20 times on each preprocessing method. We found that the results were potentially biologically significant implying that the method has the potential of determining genes playing an important role in the disease. We performed the same experiment on a combination of microarray and RNA-Seq data and we also found that the results had a probable biological explanation. This also validates the usage of UPC for combining microarray and RNA-Seq data types in an experiment and offers many perspectives for reusing legacy microarray data with recent RNA-Seq data.

7.3 Future work

7.3.1 Optimizing the RNA-Seq pipeline

The Bowtie2 integration in Hadoop has speeded the computation up, but it is still possible to do better. Some other components of the pipeline can be time consuming. For example, a distributed implementation of eXpress (Roberts et al., 2013) has been released for still faster computation. It runs on Apache Spark (Zaharia et al., 2012), a framework which can potentially process large quantities of data faster than Hadoop.

7.3.2 Improving the model for finding DEGs

Like mentioned in the results chapter 6, the model finds differentially expressed genes but only a subset of them. We do not have an idea of the biological importance of all the genes found by the model since we only analyzed the genes having some significance in the results obtained by Enrichr. A first step could be to look at the genes that were not at first sight important and see if some biological information could be inferred from them. Also it could be interesting to look at the genes that were not found by the model, but that were present in other

experiments using the same datasets. This could shed light on the potential of the method.

CHAPTER 8

Bibliography

Affymetrix (2002). Statistical algorithms description document. Technical report, Santa Clara, CA, USA.

Benito, M., Parker, J., Du, Q., Wu, J., Xiang, D., Perou, C. M., and Marron, J. S. (2004). Adjustment of systematic microarray data biases. *Bioinformatics*, *20*(1):105–114.

Bolstad, B. M. (2004). *Low-level analysis of high-density oligonucleotide array data: background, normalization and summarization*. PhD thesis, University of California, Berkeley.

Borthakur, D. (2007). The Hadoop Distributed File System: Architecture and Design. `http://hadoop.apache.org/core/docs/current/hdfs_design.pdf`. Last checked: 25 august 2014.

Chang, T.-H. and Szabo, E. (2000). Induction of differentiation and apoptosis by ligands of peroxisome proliferator-activated receptor γ in non-small cell lung cancer. *Cancer Research*, *60*(4):1129–1138.

Chen, C., Grennan, K., Badner, J., Zhang, D., Gershon, E., Jin, L., and Liu, C. (2011). Removing batch effects in analysis of expression microarray data: an evaluation of six batch adjustment methods. *PloS one*, *6*(2):e17238.

Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G. V., Clark, N. R., and Ma'ayan, A. (2013). Enrichr: interactive and collaborative HTML5 gene list enrichment analysis tool. *BMC Bioinformatics*, *14*(1):128.

Cock, P. J. A., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, *38*(6):1767–1771.

Coletta, A., Molter, C., Duque, R., Steenhoff, D., Taminau, J., de Schaetzen, V., Meganck, S., Lazar, C., Venet, D., Detours, V., Nowe, A., Bersini, H., and Weiss Solis, D. (2012). InSilico DB genomic datasets hub: an efficient starting point for analyzing genome-wide studies in GenePattern, Integrative Genomics Viewer, and R/Bioconductor. *Genome Biology*, 13(11):R104.

Crick, F. et al. (1970). Central dogma of molecular biology. *Nature*, 227(5258):561–563.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.

Dai, M., Wang, P., Boyd, A. D., Kostov, G., Athey, B., Jones, E. G., Bunney, W. E., Myers, R. M., Speed, T. P., Akil, H., Stanley J, W., and Fan, M. (2005). Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data. *Nucleic Acids Research*, *33*(20):e175–e175.

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Fransisco, CA.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*(1):1–38.

Edgar, R., Domrachev, M., and Lash, A. E. (2002). Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Research*, *30*(1):207–210.

Ein-Dor, L., Zuk, O., and Domany, E. (2006). Thousands of samples are needed to generate a robust gene list for predicting outcome in cancer. *Proceedings of the National Academy of Sciences*, *103*(15):5923–5928.

Farrell, J. J. (2013). Genomic Analysis on Hadoop. http://www.bumc.bu.edu/gsi/files/2013/12/Farrell_slides.pdf. Last checked: 18 august 2014.

Fayyad, U. M. (1992). *On the Induction of Decision Trees for Multiple Concept Learning*. PhD thesis, University of Michigan, Ann Arbor, MI, USA.

Gentleman, R., Carey, V., Huber, W., Irizarry, R., and Dudoit, S. (2005). *Bioinformatics and computational biology solutions using R and Bioconductor*, volume 746718470. Springer New York.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google File System. In *ACM SIGOPS Operating Systems Review*, volume *37*, pages 29–43. ACM.

Gioeli, D., Mandell, J. W., Petroni, G. R., Frierson, H. F., and Weber, M. J. (1999). Activation of mitogen-activated protein kinase associated with prostate cancer progression. *Cancer Research*, 59(2):279–284.

Haas, B. J. and Zody, M. C. (2010). Advancing RNA-seq analysis. *Nature Biotechnology*, *28*(5):421–423.

Hou, J., Aerts, J., Den Hamer, B., Van Ijcken, W., Den Bakker, M., Riegman, P., van der Leest, C., van der Spek, P., Foekens, J. A., Hoogsteden, H. C., Grosveld, F., and Philipsen, S. (2010). Gene expression-based classification of non-small cell lung carcinomas and survival prediction. *PloS one*, *5*(4):e10312.

Irizarry, R. A., Bolstad, B. M., Collin, F., Cope, L. M., Hobbs, B., and Speed, T. P. (2003a). Summaries of Affymetrix GeneChip probe level data. *Nucleic Acids Research*, *31*(4):e15–e15.

Irizarry, R. A., Hobbs, B., Collin, F., Beazer-Barclay, Y. D., Antonellis, K. J., Scherf, U., and Speed, T. P. (2003b). Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, *4*(2):249–264.

Johnson, W. E., Li, C., and Rabinovic, A. (2007). Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics*, *8*(1):118–127.

Johnson, W. E., Li, W., Meyer, C. A., Gottardo, R., Carroll, J. S., Brown, M., and Liu, X. S. (2006). Model-based analysis of tiling-arrays for ChIP-chip. *Proceedings of the National Academy of Sciences*, *103*(33):12457–12462.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer, 2 edition.

Kannan, K., Wang, L., Wang, J., Ittmann, M. M., Li, W., and Yen, L. (2011). Recurrent chimeric RNAs enriched in human prostate cancer identified by deep sequencing. *Proceedings of the National Academy of Sciences*, *108*(22):9172–9177.

Landi, M. T., Dracheva, T., Rotunno, M., Figueroa, J. D., Liu, H., Dasgupta, A., Mann, F. E., Fukuoka, J., Hames, M., Bergen, A. W., Murphy, S. E., Yang, P., Pesatori, A. C., Consonni, D., Bertazzi, P. A., Wacholder, S., Shih, J. H., Caporaso, N. A., and Jen, J. (2008). Gene expression signature of cigarette smoking and its role in lung adenocarcinoma development and survival. *PloS one*, *3*(2):e1651.

Langmead, B. and Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature Methods*, *9*(4):357–359.

Lazar, C., Meganck, S., Taminau, J., Steenhoff, D., Coletta, A., Molter, C., Weiss-Solís, D. Y., Duque, R., Bersini, H., and Nowé, A. (2012). Batch effect removal methods for microarray gene expression data integration: a survey. *Briefings in bioinformatics*, page bbs037.

Leek, J. T., Scharpf, R. B., Bravo, H. C., Simcha, D., Langmead, B., Johnson, W. E., Geman, D., Baggerly, K., and Irizarry, R. A. (2010). Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, *11*(10):733–739.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, *25*(14):1754–1760.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, *25*(16):2078–9.

Lipshutz, R. J., Fodor, S. P., Gingeras, T. R., and Lockhart, D. J. (1999). High density synthetic oligonucleotide arrays. *Nature genetics*, *21*(20–24).

Lockhart, D. J., Dong, H., Byrne, M. C., Follettie, M. T., Gallo, M. V., Chee, M. S., Mittmann, M., Wang, C., Kobayashi, M., Norton, H., and Brown, E. L. (1996). Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, *14*(13):1675–1680.

Lu, T.-P., Tsai, M.-H., Lee, J.-M., Hsu, C.-P., Chen, P.-C., Lin, C.-W., Shih, J.-Y., Yang, P.-C., Hsiao, C. K., Lai, L.-C., and Chuang, E. Y. (2010). Identification of a novel biomarker, SEMA5A, for non-small cell lung carcinoma in nonsmoking women. *Cancer Epidemiology Biomarkers & Prevention*, *19*(10):2590–2597.

Luo, J., Schumacher, M., Scherer, A., Sanoudou, D., Megherbi, D., Davison, T., Shi, T., Tong, W., Shi, L., Hong, H., Zhao, C., Elloumi, F., Shi, W., Thomas, R., Lin, S., Tillinghast, G., Liu, G., Zhou, Y., Herman, D., Li, Y., Deng, Y., Fang,

H., Bushel, P., Woods, M., and Zhang, J. (2010). A comparison of batch effect removal methods for enhancement of prediction performance using MAQC-II microarray gene expression data. *The Pharmacogenomics Journal*, *10*(4):278–191.

Marron, J. S., Todd, M. J., and Ahn, J. (2007). Distance-weighted discrimination. *Journal of the American Statistical Association*, *102*(480):1267–1271.

McCall, M. N., Bolstad, B. M., and Irizarry, R. A. (2010). Frozen robust multiarray analysis (fRMA). *Biostatistics*, *11*(2):242–253.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, 1 edition.

Nagalakshmi, U., Wang, Z., Waern, K., Shou, C., Raha, D., Gerstein, M., and Snyder, M. (2008). The transcriptional landscape of the yeast genome defined by RNA sequencing. *Science*, *320*(5881):1344–1349.

Nagy, L., Tontonoz, P., Alvarez, J. G., Chen, H., and Evans, R. M. (1998). Oxidized LDL regulates macrophage gene expression through ligand activation of PPARγ. *Cell*, *93*(2):229–240.

National Human Genome Research Institute (2014). Biological pathways.

O'Driscoll, A., Daugelaite, J., and Sleator, R. D. (2013). 'Big data', Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics*, *46*(5):774–781.

Parkinson, H., Kapushesky, M., Kolesnikov, N., Rustici, G., Shojatalab, M., Abeygunawardena, N., Berube, H., Dylag, M., Emam, I., Farne, A., et al. (2009). ArrayExpress update—from an archive of functional genomics experiments to the atlas of gene expression. *Nucleic Acids Research*, *37*(suppl 1):D868–D872.

Piccolo, S. R., Sun, Y., Campbell, J. D., Lenburg, M. E., Bild, A. H., and Johnson, W. E. (2012). A single-sample microarray normalization method to facilitate personalized-medicine workflows. *Genomics*, *100*(6):337–344.

Piccolo, S. R., Withers, M. R., Francis, O. E., Bild, A. H., and Johnson, W. E. (2013). Multiplatform single-sample estimates of transcriptional activation. *Proceedings of the National Academy of Sciences*, *110*(44):17778–17783.

Rebhan, M., Chalifa-Caspi, V., Prilusky, J., and Lancet, D. (1997). GeneCards: integrating information about genes, proteins and diseases. *Trends in Genetics*, *13*(4):163.

Roberts, A., Feng, H., and Pachter, L. (2013). Fragment assignment in the cloud with eXpress-D. *BMC Bioinformatics*, *14*(1):358.

Roberts, A. and Pachter, L. (2013). Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature Methods*, *10*(1):71–73.

Sanchez-Palencia, A., Gomez-Morales, M., Gomez-Capilla, J. A., Pedraza, V., Boyero, L., Rosell, R., and Fárez-Vidal, M. (2011). Gene expression profiling reveals novel biomarkers in nonsmall cell lung cancer. *International Journal of Cancer*, *129*(2):355–364.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*(3):379–423.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE.

Sims, A. H., Smethurst, G. J., Hey, Y., Okoniewski, M. J., Pepper, S. D., Howell, A., Miller, C. J., and Clarke, R. B. (2008). The removal of multiplicative, systematic bias allows integration of breast cancer gene expression datasets–improving meta-analysis and prediction of prognosis. *BMC Medical Genomics*, *1*(1):42.

Smythe, W. R., Williams, J. P., Wheelock, M. J., Johnson, K. R., Kaiser, L. R., and Albelda, S. M. (1999). Cadherin and catenin expression in normal human bronchial epithelium and non-small cell lung cancer. *Lung Cancer*, *24*(3):157–168.

Su, L.-J., Chang, C.-W., Wu, Y.-C., Chen, K.-C., Lin, C.-J., Liang, S.-C., Lin, C.-H., Whang-Peng, J., Hsu, S.-L., Chen, C.-H., and Huang, C.-Y. F. (2007). Selection of DDX5 as a novel internal control for Q-RT-PCR from microarray data using a block bootstrap re-sampling scheme. *BMC Genomics*, *8*(1):140.

Taminau, J. (2012). *Unlocking the potential of public available gene expression data for large-scale analysis*. PhD thesis, Vrije Universiteit Brussel.

Taminau, J., Meganck, S., Lazar, C., Steenhoff, D., Coletta, A., Molter, C., Duque, R., de Schaetzen, V., Solís, D. Y. W., Bersini, H., et al. (2012). Unlocking the potential of publicly available microarray data using inSilicoDb and inSilicoMerging R/Bioconductor packages. *BMC Bioinformatics*, *13*(1):335.

Taminau, J., Steenhoff, D., Coletta, A., Meganck, S., Lazar, C., de Schaetzen, V., Duque, R., Molter, C., Bersini, H., Nowé, A., et al. (2011). inSilicoDb:

an R/Bioconductor package for accessing human Affymetrix expert-curated datasets from GEO. *Bioinformatics*, *27*(22):3204–3205.

Therneau, T., Atkinson, B., and Ripley, B. (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8.

Therneau, T. M., Atkinson, E. J., et al. (1997). An introduction to recursive partitioning using the RPART routines.

Tran, Q.-N. (2013). A novel method for finding non-small cell lung cancer diagnosis biomarkers. *BMC Medical Genomics*, *6*(Suppl 1):S11.

Trapnell, C., Pachter, L., and Salzberg, S. L. (2009). TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, *25*(9):1105–1111.

Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., van Baren, M. J., Salzberg, S. L., Wold, B. J., and Pachter, L. (2010). Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, *28*(5):511–515.

Tsubouchi, Y., Sano, H., Kawahito, Y., Mukai, S., Yamada, R., Kohno, M., Inoue, K.-i., Hla, T., and Kondo, M. (2000). Inhibition of human lung cancer cell growth by the peroxisome proliferator-activated receptor-γ agonists through induction of apoptosis. *Biochemical and Biophysical Research Communications*, *270*(2):400–405.

Turkington, G. (2013). *Hadoop Beginner's Guide*. Packt Publishing, 1 edition.

Wallace, T. A., Prueitt, R. L., Yi, M., Howe, T. M., Gillespie, J. W., Yfantis, H. G., Stephens, R. M., Caporaso, N. E., Loffredo, C. A., and Ambs, S. (2008). Tumor immunobiological differences in prostate cancer between African-American and European-American men. *Cancer Research*, *68*(3):927–936.

Wang, Z., Gerstein, M., and Snyder, M. (2009). RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, *10*(1):57–63.

Wei, G., Abraham, B. J., Yagi, R., Jothi, R., Cui, K., Sharma, S., Narlikar, L., Northrup, D. L., Tang, Q., Paul, W. E., Zhu, J., and Zhao, K. (2011). Genome-wide analyses of transcription factor GATA3-mediated gene regulation in distinct T cell types. *Immunity*, *35*(2):299–311.

White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media, 1 edition.

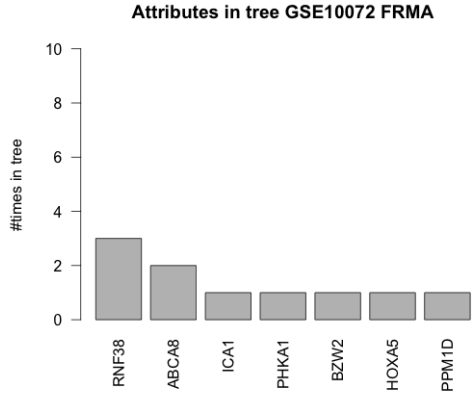
Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.

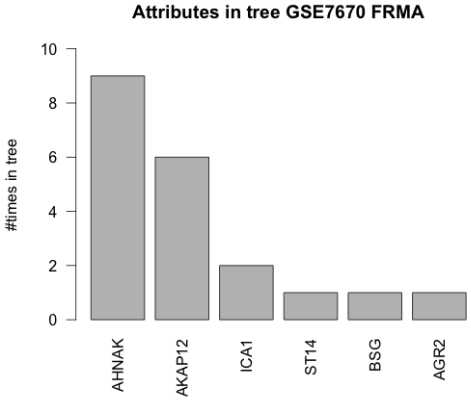
Zvelebil, M. J. and Baum, J. O. (2007). *Understanding Bioinformatics*. Garland Science, 1 edition.

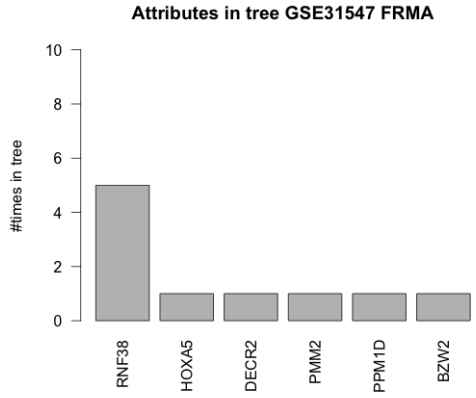
APPENDIX A

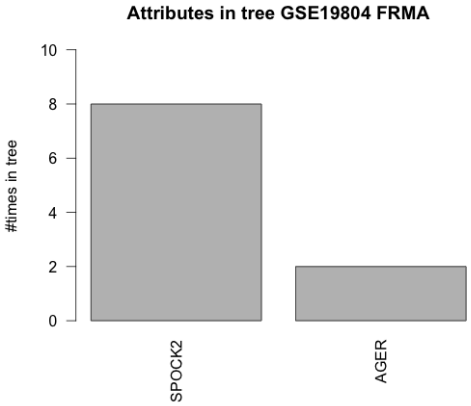
Figures: stability decision trees

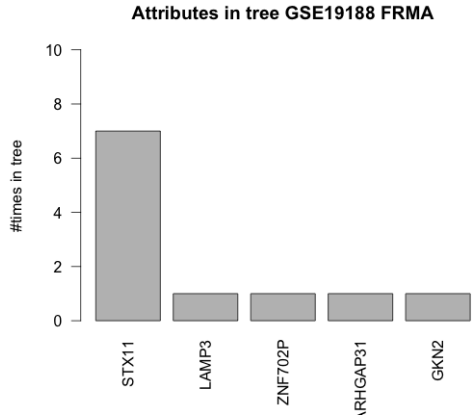
A.1 Stability individual datasets

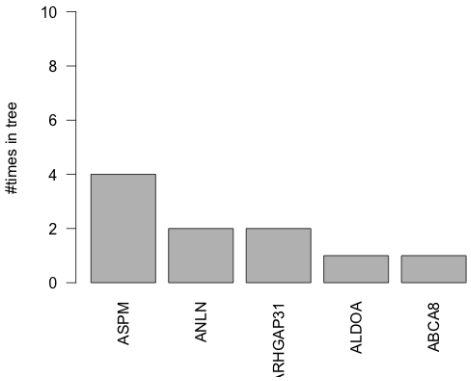

Attributes in tree GSE10072 FRMA
#times in tree
0
2
4
6
8
10
RNF38
ABCA8
ICA1
PHKA1
BZW2
HOXA5
PPM1D

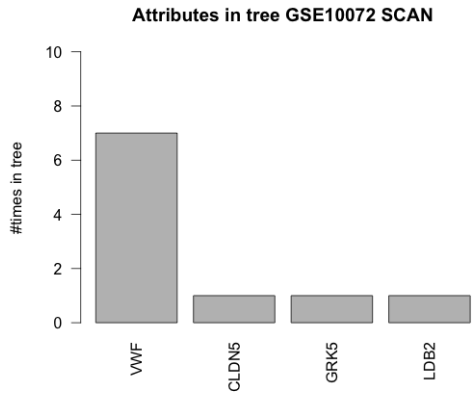

Attributes in tree GSE7670 FRMA
#times in tree
0
2
4
6
8
10
AHNAK
AKAP12
ICA1
ST14
BSG
AGR2

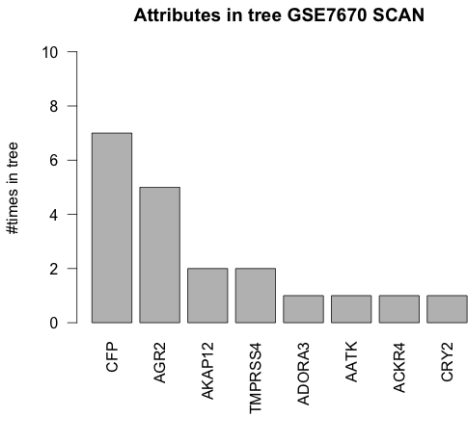

Attributes in tree GSE31547 FRMA
#times in tree
0
2
4
6
8
10
RNF38
HOXA5
DECR2
PMM2
PPM1D
BZW2

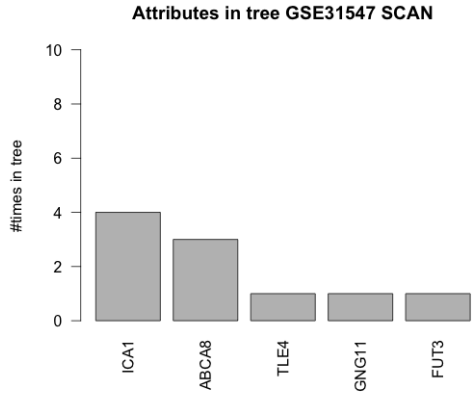

Attributes in tree GSE19804 FRMA
#times in tree
0
2
4
6
8
10
SPOCK2
AGER

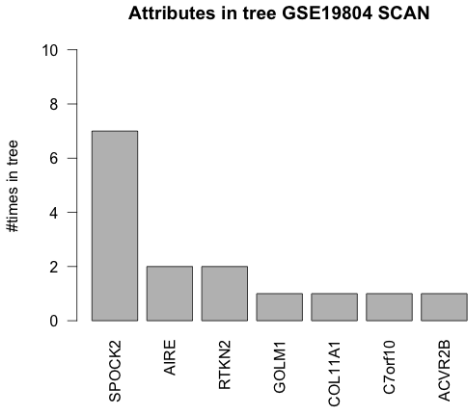

Attributes in tree GSE19188 FRMA
#times in tree
0
2
4
6
8
10
STX11
LAMP3
ZNF702P
GKN2

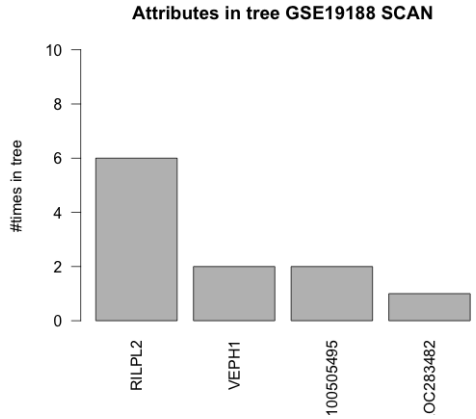

Attributes in tree GSE18842 FRMA
#times in tree
0
2
4
6
8
10
ASPM
ANLN
ALDOA
ABCA8

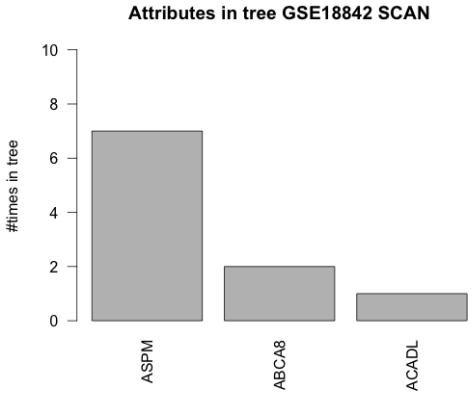

Attributes in tree GSE10072 SCAN
#times in tree
0
2
4
6
8
10
VWF
CLDN5
GRK5
LDB2

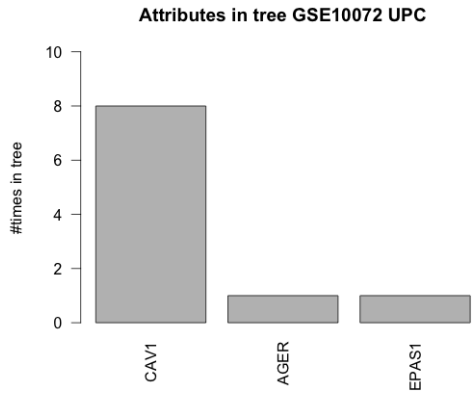

Attributes in tree GSE7670 SCAN
#times in tree
0
2
4
6
8
10
CFP
AGR2
AKAP12
TMPRSS4
ADORA3
AATK
ACKR4
CRY2

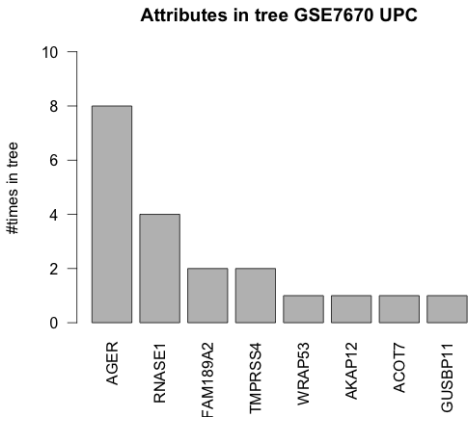

Attributes in tree GSE31547 SCAN
#times in tree
0
2
4
6
8
10
ICA1
ABCA8
TLE4
GNG11
FUT3

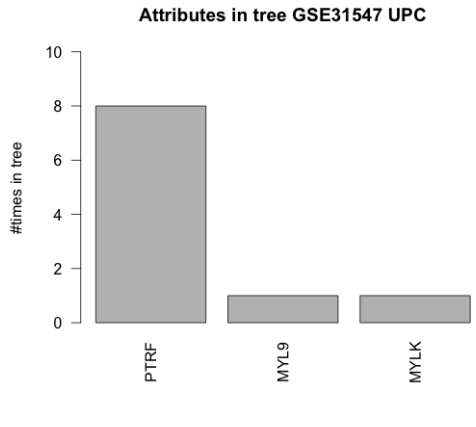

Attributes in tree GSE19804 SCAN
#times in tree
0
2
4
6
8
10
SPOCK2
AIRE
RTKN2
GOLM1
COL11A1
C7orf10
ACVR2B

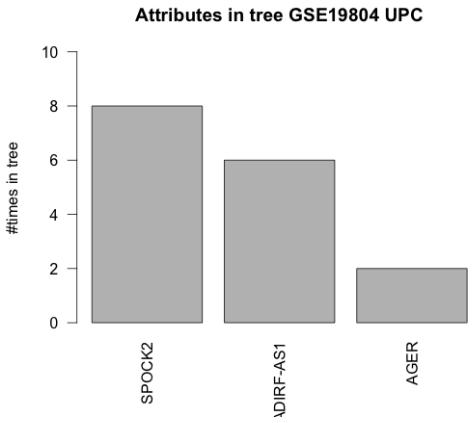

Attributes in tree GSE19188 SCAN
#times in tree
0
2
4
6
8
10
RILPL2
VEPH1
100505495
OC283482

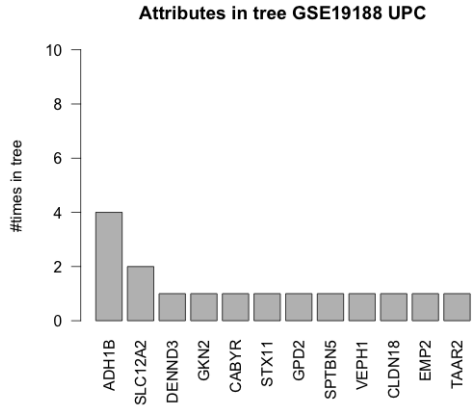

Attributes in tree GSE18842 SCAN
#times in tree
0
2
4
6
8
10
ASPM
ABCA8
ACADL

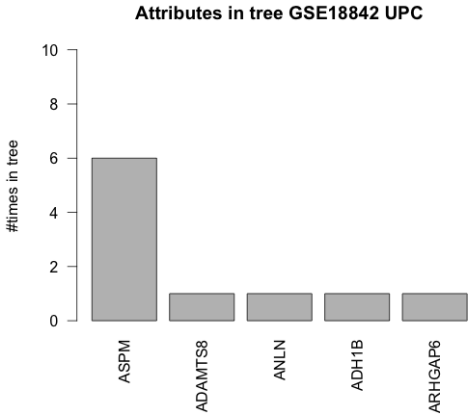

Attributes in tree GSE10072 UPC
#times in tree
0
2
4
6
8
10
CAV1
AGER
EPAS1


Attributes in tree GSE7670 UPC
#times in tree
0
2
4
6
8
10
AGER
RNASE1
FAM189A2
TMPRSS4
WRAP53
AKAP12
ACOT7
GUSBP11

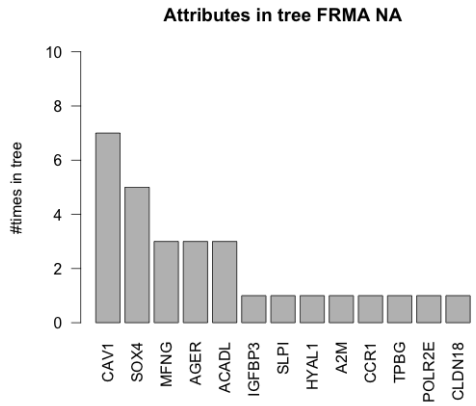

Attributes in tree GSE31547 UPC
#times in tree
0
2
4
6
8
10
PTRF
MYL9
MYLK

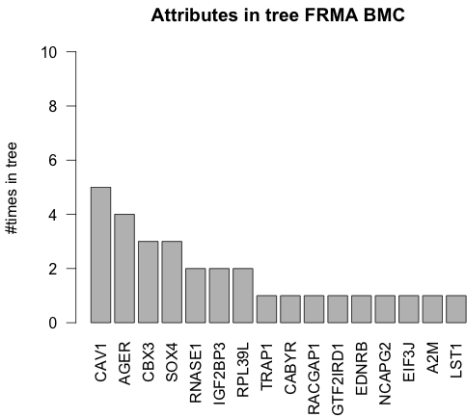

Attributes in tree GSE19804 UPC
#times in tree
0
2
4
6
8
10
SPOCK2
AGER

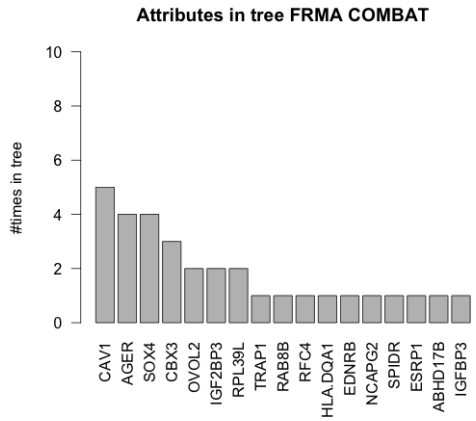

Attributes in tree GSE19188 UPC
#times in tree
0
2
4
6
8
10
ADH1B
SLC12A2
DENND3
GKN2
CABYR
STX11
GPD2
SPTBN5
VEPH1
CLDN18
EMP2
TAAR2

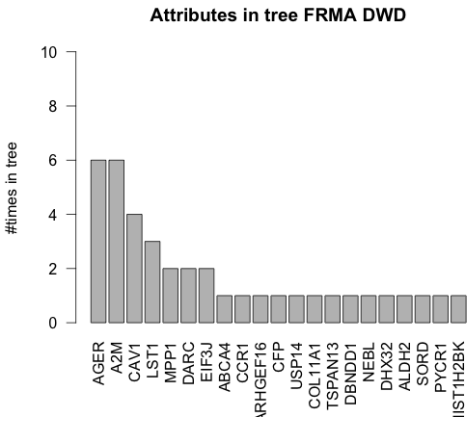

Attributes in tree GSE18842 UPC
#times in tree
0
2
4
6
8
10
ASPM
ADAMTS8
ANLN
ADH1B
ARHGAP6

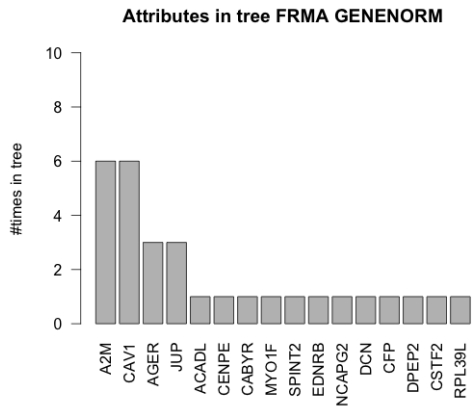
A.2 Stability batch effect removal methods

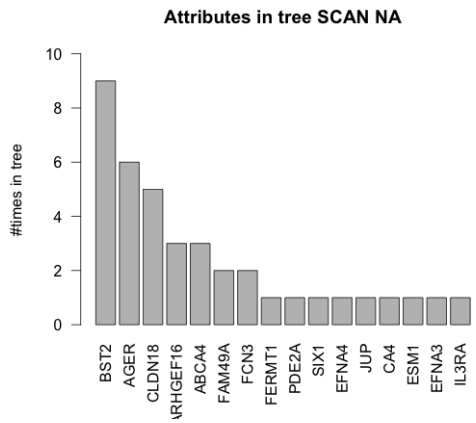

Attributes in tree FRMA NA
#times in tree
0
2
4
6
8
10
CAV1
SOX4
MFNG
AGER
ACADL
IGFBP3
SLPI
HYAL1
A2M
CCR1
TPBG
POLR2E
CLDN18

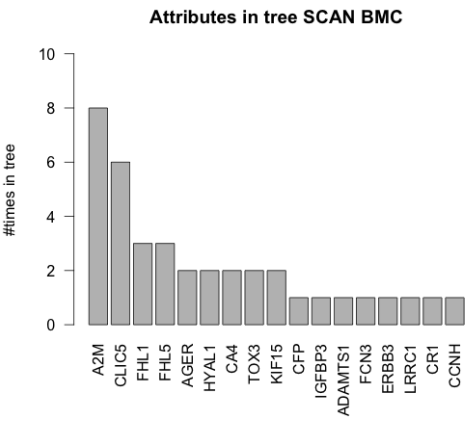

Attributes in tree FRMA BMC
#times in tree
0
2
4
6
8
10
CAV1
AGER
CBX3
SOX4
RNASE1
IGF2BP3
RPL39L
TRAP1
CABYR
RACGAP1
GTF2IRD1
EDNRB
NCAPG2
EIF3J
A2M
LST1

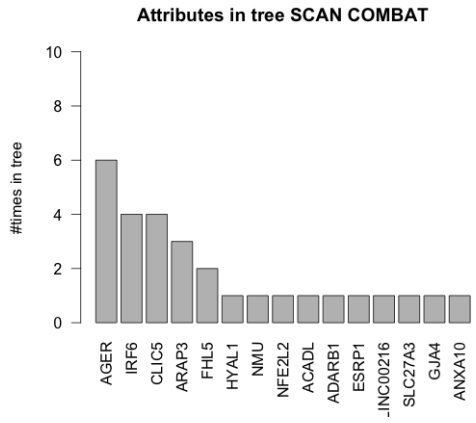

Attributes in tree FRMA COMBAT
#times in tree
0
2
4
6
8
10
CAV1
AGER
SOX4
CBX3
OVOL2
IGF2BP3
RPL39L
TRAP1
RAB8B
RFC4
HLA.DQA1
EDNRB
NCAPG2
SPIDR
ESRP1
ABHD17B
IGFBP3

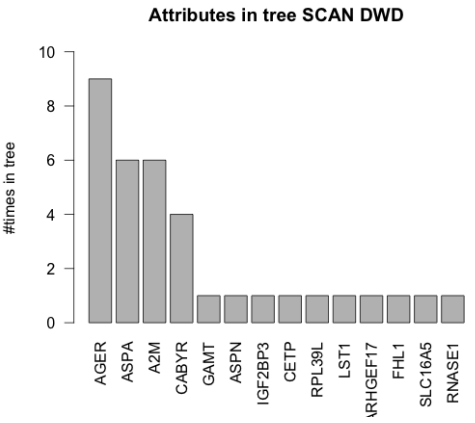

Attributes in tree FRMA DWD
#times in tree
0
2
4
6
8
10
AGER
A2M
CAV1
LST1
MPP1
DARC
EIF3J
ABCA4
CCR1
ARHGEF16
CFP
USP14
COL11A1
TSPAN13
DBNDD1
NEBL
DHX32
ALDH2
SORD
PYCR1
HIST1H2BK

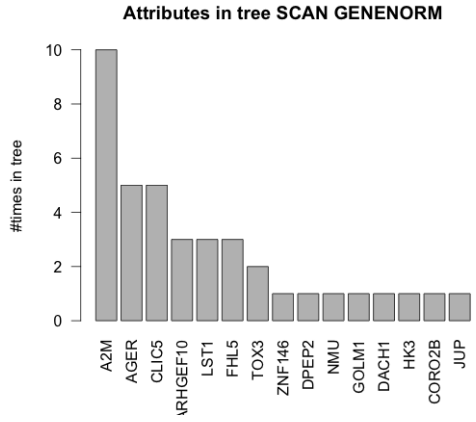

Attributes in tree FRMA GENENORM
#times in tree
0
2
4
6
8
10
A2M
CAV1
AGER
JUP
ACADL
CENPE
CABYR
MYO1F
SPINT2
EDNRB
NCAPG2
DCN
CFP
DPEP2
CSTF2
RPL39L

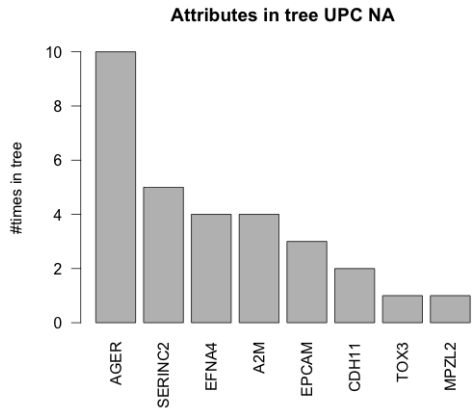

Attributes in tree SCAN NA
#times in tree
0
2
4
6
8
10
BST2
AGER
CLDN18
ARHGEF16
ABCA4
FAM49A
FCN3
FERMT1
PDE2A
SIX1
EFNA4
JUP
CA4
ESM1
EFNA3
IL3RA

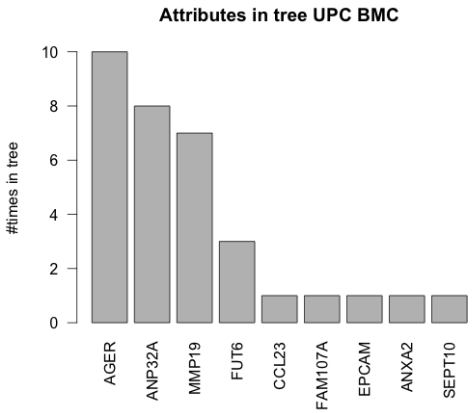

Attributes in tree SCAN BMC
#times in tree
0
2
4
6
8
10
A2M
CLIC5
FHL1
FHL5
AGER
HYAL1
CA4
TOX3
KIF15
CFP
IGFBP3
ADAMTS1
FCN3
ERBB3
LRRC1
CR1
CCNH

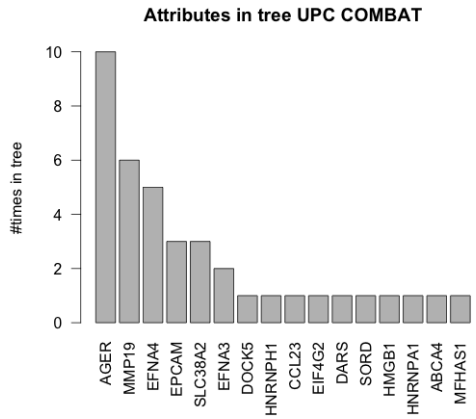

Attributes in tree SCAN COMBAT
#times in tree
0
2
4
6
8
10
AGER
IRF6
CLIC5
ARAP3
FHL5
HYAL1
NMU
NFE2L2
ACADL
ADARB1
ESRP1
LINC00216
SLC27A3
GJA4
ANXA10

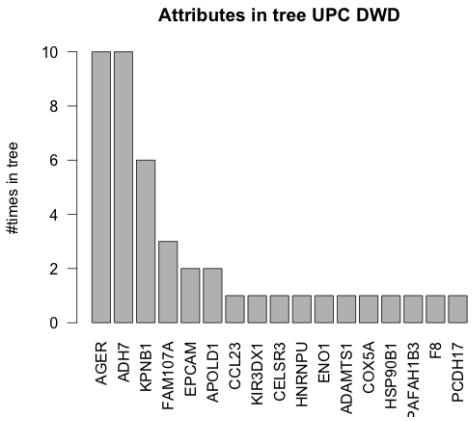

Attributes in tree SCAN DWD
#times in tree
0
2
4
6
8
10
AGER
ASPA
A2M
CABYR
GAMT
ASPN
IGF2BP3
CETP
RPL39L
LST1
ARHGEF17
FHL1
SLC16A5
RNASE1

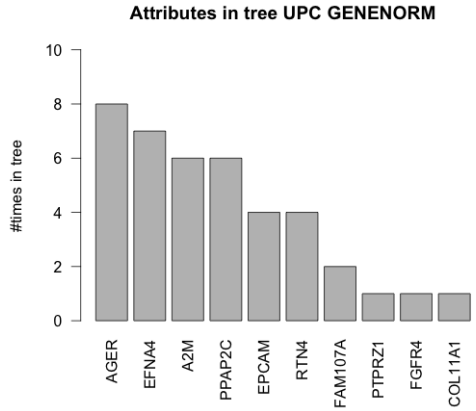

Attributes in tree SCAN GENENORM
#times in tree
0
2
4
6
8
10
A2M
AGER
CLIC5
ARHGEF10
LST1
FHL5
TOX3
ZNF146
DPEP2
NMU
GOLM1
DACH1
HK3
CORO2B
JUP


Attributes in tree UPC NA
#times in tree
0
2
4
6
8
10
AGER
SERINC2
EFNA4
A2M
EPCAM
CDH11
TOX3
MPZL2


Attributes in tree UPC BMC
#times in tree
0
2
4
6
8
10
AGER
ANP32A
MMP19
FUT6
CCL23
FAM107A
EPCAM
ANXA2
SEPT10


Attributes in tree UPC COMBAT
#times in tree
0
2
4
6
8
10
AGER
MMP19
EFNA4
EPCAM
SLC38A2
EFNA3
DOCK5
HNRNPH1
CCL23
EIF4G2
DARS
SORD
HMGB1
HNRNPA1
ABCA4
MFHAS1


Attributes in tree UPC DWD
#times in tree
0
2
4
6
8
10
AGER
ADH7
KPNB1
FAM107A
EPCAM
APOLD1
CCL23
KIR3DX1
CELSR3
HNRNPU
ENO1
ADAMTS1
COX5A
HSP90B1
F8
PCDH17


Attributes in tree UPC GENENORM
#times in tree
0
2
4
6
8
10
AGER
EFNA4
A2M
PPAP2C
EPCAM
RTN4
FAM107A
PTPRZ1
FGFR4
COL11A1