FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Learning for genomic data analysis

Vítor Filipe Oliveira Teixeira

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Camacho (FEUP)

Second Supervisor: Pedro Ferreira (I3S)

July 23, 2017

Deep Learning for genomic data analysis

Vítor Filipe Oliveira Teixeira

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Name of the President
External Examiner: Doctor Name of the Examiner
Supervisor: Doctor Name of the Supervisor

July 23, 2017

Abstract

Since the Human Genome Project that the availability of genomic data has been increasing. With the huge investments in research, genome sequencing technologies and techniques have been improving at a fast rate, resulting in a cheaper yet faster genome sequencing. Such amount of data enables an advanced and more detailed analysis, which leads to advances in research. Data generated by sequencing methods, like RNA-Seq, can be used to generate gene expression data which contain key information about the molecular basis of dangerous diseases such as cancer. However, the publicly available gene expression data related to cancer has some issues related to its nature, namely the low number of available samples to study, as well as a huge class imbalance problem. Moreover, this data is highly complex as it has a high dimensionality due to the number of genes, and, because of that, a considerable computational power and efficient algorithms are mandatory in order to extract useful information and perform it in reasonable time, which can represent a constraint on the extraction and comprehension of such information.

In this work, we focus on the biological aspects of RNA-Seq and in the analysis of the created representations by deep learning methods, given the recent and increasing number of records broken by this approach. We divided our study into two main branches. First, we built and compared the performance of several feature extraction methods as well as data sampling methods using classifiers that were able distinguish the RNA-seq samples of thyroid cancer patients from samples of healthy persons. Secondly, we have investigated the possibility of building comprehensible descriptions of gene expression data by using Denoising Autoencoders and Stacked Denoising Autoencoders as feature extraction methods. After extracting information related to the description built by the network, namely the connection weights, we devised post-processing techniques to extract comprehensible and biologically meaningful descriptions out of the constructed models.

Resumo

Desde o Human Genome Project que os dados genómicos se têm tornado de mais fácil acesso. Com os inúmeros investimentos na área, as tecnologias de sequenciação de genomas tornam-se mais avançadas e sofisticadas, permitindo assim uma sequenciação mais barata e mais rápida. Tal quantidade de dados permite uma melhor e mais avançada pesquisa, o que leva a novas descobertas na área. Através dos dados de sequenciação, como RNA-Seq, podem ser obtidos dados de expressão genética, que contêm informações importantes para perceber a base molecular de doenças perigosas, como por exemplo cancro. No entanto, os dados disponiveis publicamente possuem problemas no que diz respeito ao número de exemplos para a análise, e também uma grande disparidade no que diz respeito ao número de exemplos de diferentes tipos num determinado conjunto de dados. Para além disso, estes dados possuem uma elevada dimensionalidade devido ao número de genes, e, por isso, são necessários algoritmos eficientes e um grande poder computacional de maneira a analisar e extrair informação útil num tempo aceitável, o que representa uma barreira no que diz respeito à extração e interpretação da informação.

Neste trabalho focamo-nos principalmente nos aspectos biológicos do RNA-Seq e na análise das representações criadas usando métodos de deep learning, dado o número de recordes que estes métodos têm batido recentemente. O trabalho foi dividido em duas vertentes principais. Na primeira construímos e comparamos a performance de vários métodos de extração de *features* e métodos de *sampling* de dados usando classificadores que foram capazes de distinguir amostras de RNA-Seq de pacientes com cancro da tiróide de amostras de pessoas saudáveis. Em segundo lugar, foi investigada a possibilidade de construir boas descrições dos dados de expressão genética usando *Denoising Autoencoders* e *Stacked Denoising Autoencoders* para extracção de features. Após o treino dos modelos foi realizado um pós-processamento dos pesos extraídos dos modelos de maneira a conseguir retirar informação importante acerca da função e relação entre os genes.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Rui Camacho, for his promptness and guidance, that always pointed me to the right direction whenever I needed the most, and for all the help and freedom he gave throughout this thesis.

I would like to thank Pedro Ferreira, my second supervisor, for all the valuable advices he gave me, for helping me with all the questions I had concerning molecular biology and for helping me validate the results. Without his passionate participation this thesis would not have the same value.

I also would like leave a special thanks to my friend Joel Mendes, for being such a good friend and as well as to all my friends at FEUP, for being with me throughout all these years and making it one of the best periods of my life.

To my parents, Joaquim Teixeira, Maria Oliveira and to my sister, Liliana Teixeira, my deepest heartfelt thank you, for always providing me with unfailing support and encouragement. I owe them what I am today and will be in the future.

Last but not least, a truly heartfelt thank you to my girlfriend, Bárbara Pires. For always being there when I needed the most, for standing by me through the best and the worst times and for always putting a smile on my face and giving me the strength to face all the problems.

Vítor Teixeira

”The greatest obstacle to discovery is not ignorance
- it is the illusion of knowledge.
Never tell people how to do things. Tell them what
to do and they will surprise you with their ingenuity.”

George S. Patton

Contents

List of Figures

LIST OF FIGURES

List of Tables

List of Algorithms

Abbreviations

AdaGrad	Adaptative Gradient
ADAM	Adaptative Moment estimation
ADASYN	Adaptative Synthetic
API	Application Programming Interface
BAM	Binary Alignment Map
CNN	Convolution Neural Network
CPU	Central Processing Unit
DAE	Denoising Autoencoder
DL4J	Deep Learning For Java
DNA	Deoxyribonucleic acid
ENCODE	Encyclopedia of DNA Elements
GEO	Gene Expression Omnibus
GFF	General Feature Format
GO	Gene Onthology
GPU	Graphical Processing Unit
GTF	General Feature Format
GUI	Graphical User Interface
HTML	HyperText Markup Language
JVM	Java Virtual Machine
KEGG	Kyoto Encyclopedia of Genes and Genomes
KPCA	Kernel Principal Component Analysis
MRDM	Multi-Relational Data Mining
mRNA	Messenger RNA
MSE	Mean Squared Error
NCBI	National Center for Biotechnology Information
ND4J	N-Dimensional Arrays For Java
NGS	Next-Generation Sequencing
NHGRI	National Human Genome Research Institute
PCA	Principal Component Analysis
POM	Project Object Model
RAM	Random Access Memory
RBM	Restricted Boltzman Machine
RDM	Relational Data Mining
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RNA	Ribonucleic acid
RNA-Seq	RNA Sequencing
RNN	Recurrent neural Networks
SAM	Sequence Alignment Map

ABBREVIATIONS

SDAE	Stacked Denoising Autoencoder
SMOTE	Synthetic Minority Over-sampling Technique
SQL	Structured Query Language
Tanh	Hyperbolic Tangent
TCGA	The Cancer Genome Atlas
tRNA	Transfer RNA
TSV	Tab Separated Values
VCF	Variant Call Format
YARN	Yet Another Resource Negotiator

Chapter 1

Introduction

In this chapter the context of this work is briefly explained as well as the underlying motivations and issue that arrive when dealing with the problem described. Then we explain what we aim to obtain with this research and end with a brief explanation of the structure of this report.

1.1 Context

Cancer is a group of diseases that involve an uncontrolled growth and spread of abnormal cells and is known to be responsible for the death of millions every year [1]. While there is still no cure for these diseases, there is still the possibility of enhancing the quality of medical diagnosis and disease prognosis. There is, however, an inherent difficult that is specific to the cancer type. For instance, in thyroid cancer, the tumors are expressed as thyroid nodes, and, among them, 95% are benign, which raises the difficulty of the diagnosis [Uti05].

A biomarker is any substance, structure or process that can be measured in the body or its products and influence or predict the incidence of outcome or disease [$O^{+}93$]. Biomarkers play a critical role in understanding molecular and cellular mechanisms that drive tumor initiation, maintenance and progression. Early disease detection by biomarkers offers an effective opportunity for enhancing disease detection, improving patient prognosis and optimizing the use of drug therapy to each case and assessing clinical outcomes of treatment. Hence biomarkers are known to be useful in several phases of the disease [Pfa13]:

- Before diagnosis, they provide the potential for screening and risk assessment.
- As part of the diagnostic process, biomarkers can determine staging, grading, and selection of initial therapy.
- In the treatment phase, they can be used to monitor therapy success, select additional therapies or monitor recurrent diseases.

[1] http://www.cancerresearchuk.org/health-professional/cancer-statistics/worldwide-cancer

That said, it is important that we continue the pursuit of such biomarkers in order to decrease the number of fatalities caused by the disease. Last decade's advances in sequencing techniques had an huge impact on genomics and enabled an inexpensive production of large amounts of sequencing data [Mar08]. As such, RNA biomarkers have been the choice in cancer research [PSM15][BM13][Kis15] and can be discovered by analysing RNA-Seq data. RNA-Seq uses next-generation sequencing to reveal the presence and quantify the amount of RNA present in a cell at a given moment and can be used to determine differences in gene expression over different groups [WL09]. However, the analysis and interpretation of gene-expression data still presents to be a significant challenge due to the nature of the data. There are three main challenges that are often faced when trying to extract any meaning from gene-expression cancer data: the low sample size of the dataset, the high dimensional noisy data, and how to extract the information from it. As such, a careful data processing and efficient algorithms are a must in order to extract meaningful information from it.

1.2 Motivation and objectives

Next-generation sequencing techniques led to the sequencing of cancer and normal genomes within a matter of weeks at a low price. Given the high mortality associated with cancer, the research for prevention and cure should be of high priority. However, it is still a challenge for researchers to analyse the data and extract useful information from it given the high complexity and sparsity of the data.

Deep learning is getting more and more attention after being known to outperform commonly used methods for classification [HSK$^+$12]. Although these methods are not fully understood they are known to perform well in various situations if tuned well, which is proven to be a rather difficult task [Ben12]. That said, it is important to study and assess their performance in important fields like genomics that can revolutionize nowadays molecular biology knowledge and lead to potential discoveries that can help on clinical diagnosis and disease prognosis.

Of the more than 50000 genes that a gene-expression dataset can contain, only a few are relevant to the problem. Extracting the ones that are relevant and studying their influence on the disease is the main goal of this thesis. Methods like Denoising Autoencoders, that aim to reduce the dimensionality of the data by being forced to compress the data into a lower feature space by minimizing the difference between the input data and the reconstruction of an intentionally corrupted input data, have been used to extract the most important features from gene-expression data, making it easier to analyse a lower subset of genes [TUCG15]. In this work we will use papillary thyroid carcinoma gene expression data from The Cancer Genome Atlas. We will study the best ways to deal with the inherent problems related to the nature of the data and will assess the performance of Stacked Denoising Autoencoders for feature extraction of gene-expression data. Stacked Denoising Autoencoders are composed of many Denoising Autoencoders and are able to extract more non-linearities within the data.

In sum, in this work we will start by dealing with the challenges mentioned above, namely, the low number of samples, data noise and high dimensionality, and how to extract important information from such generated features. First, we will deal with the low number of samples by comparing the performance of different data sampling methods given different conditions. Then, we deal with the high number of features by using and comparing different feature reduction methods, including Denoising and Stacked Denoising Autoencoders, that are the main focus in this thesis. Finally, after having the number of features reduced to a sensible number, we want to be able to extract comprehensible biological meaning from them in order to be able to help biologists discover novel cancer biomarkers. We will use several ways to extract the genes that most contributed for the generated features by analysing the final weights from the trained models. After having a list of what we call *high weight genes* we will try to cluster the genes using a functional annotation clustering tool in order to detect patterns and relations between the extracted genes and conclude if the generated features from the autoencoders have any biological meaning.

1.3 Structure of the thesis

Apart from this introductory chapter, the report has four additional chapters. Chapter 2 contains a detailed explanation on the state-of-the-art in topics relevant for the thesis work. We start by introducing the biological and genomic concepts needed to understand the given problem as well as RNA-Seq analysis and the most commonly used tools to generate and analyse gene-expression data. Furthermore, we also explain the concepts behind data mining and deep learning methods that we will use to analyse the data and present some of the most commonly used tools. The chapter ends with the description of relevant distributed frameworks that can be used in order to accelerate generation and analysis of gene-expression data.

In Chapter 3 we give a more deep overview of the problem and the challenges that arise when trying to solve it. We finish by explaining what was our rational behind our devised solution. Chapter 4 describes and explains all of the implementation details, from the choice of the used tools and the pre-processing of the data to the way we implemented each step necessary to perform a given experiment. Chapter 5 concludes the thesis work and explain the difficulties that were found found during development stage. Future work is also part of Chapter 5.

Chapter 2

Biological and technological concepts

In this chapter begin by making an introductory approach to RNA-Seq and its underlying knowledge. Then, we will briefly review data mining algorithms and lastly introduce deep learning main concepts and the most commonly used deep learning and distributed frameworks.

2.1 Molecular Biology

DNA

DNA is known to be the key molecule in every living organism as it carries the genetic information concerning each individual. It can be found at a cell's nucleus [1] wrapped in a thread-like structure called chromosome in eukaryotic organisms or in the cytoplasm for prokaryotes.

DNA stands for Deoxyribonucleic Acid. DNA molecules are formed by two strands that form a double-helix. Those strands are composed of nucleotides. Each nucleotide contains a sugar (deoxyribose), a phosphate group and one nitrogenous base. There are four bases that can be present on DNA: Cytosine, Adenine, Guanine and Thymine. These bases are held onto each other by hydrogen bonds, connecting nucleotides thus forming the double stranded shape. There are, however, some base pairing rules. Each base cannot be paired with any other. Adenine can only be paired with Thymine and Guanine with Cythosin. [DNA] (Figure 1)

RNA

RNA stands for Ribonucleic Acid and is a molecule responsible for the coding, decoding, regulation and expression of genes [Cla08]. RNA and DNA have a similar structure, however, RNA only has a single strand that folds onto itself and its sugar is Ribose. It is composed of four types of ribonucleotide bases: Adenine, Cytosine, Guanine and Uracil. (Figure 2.1)

[1] and also in mitocondria but not wrapped in chromosomes

[2] Image taken from: http://www.differencebetween.net/science/difference-between-dna-and-rna/

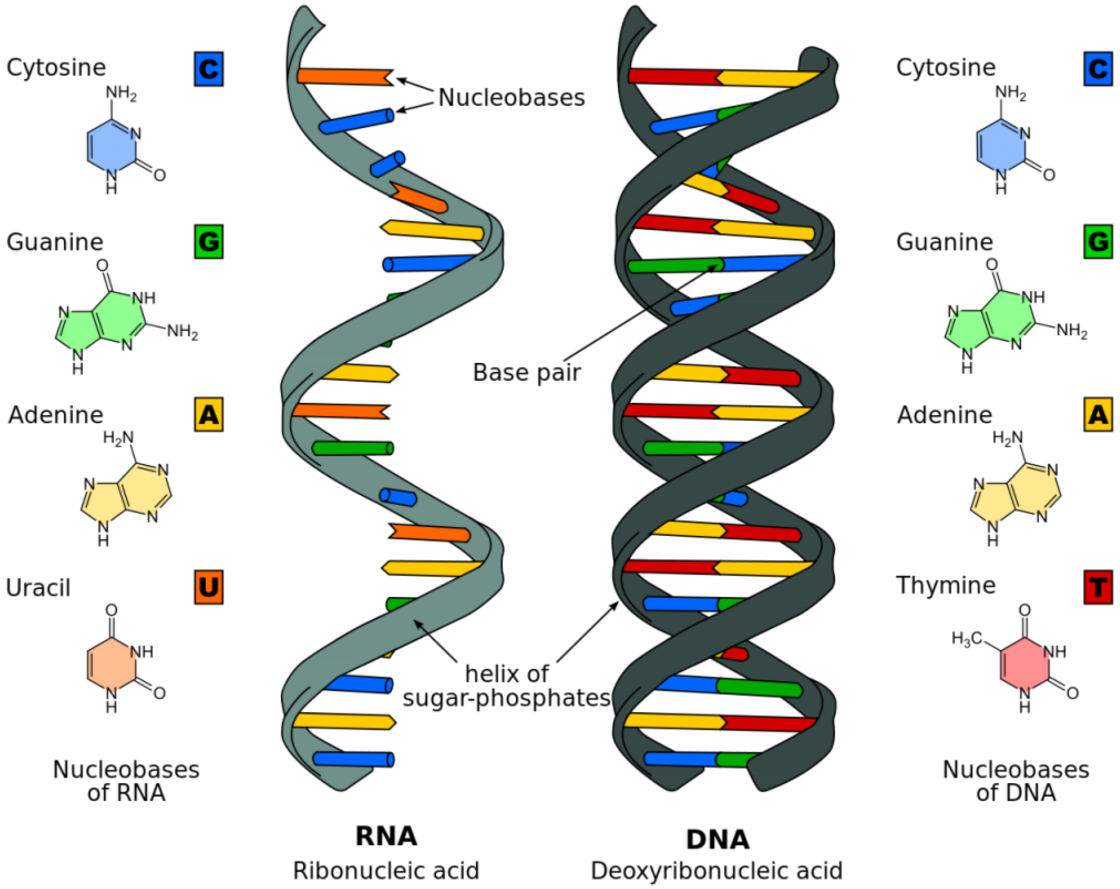


Figure 2.1: DNA and RNA structures[2]

Gene expression – from DNA to RNA to Proteins

DNA determines the structure of a cell, meaning whether it is meant to be an eye cell, a skin cell and so forth [RS08]. DNA contains genes and those genes are used to produce RNA (the transcription process) that has the information needed to synthesize a protein in a process called gene expression [CH16].

A gene is a continuous string of nucleotides that begins with a promoter and ends with a terminator. They can also contain regulatory sequences that can increase or decrease the expression of the specific gene. (Figure 2).

The process of transforming DNA into proteins is divided in two stages: transcription, where RNA is produced using DNA templates and translation, where proteins are synthesized using RNA templates.

Transcription occurs inside the nucleus and itself can be divided in three stages: initiation, elongation and termination. In the initiation stage, the RNA polymerase binds to the promoter region of the gene where the majority of the gene expression is controlled. As the RNA polymerase binds to the DNA, it separates the two strands. Then, in elongation the RNA polymerase slides along the DNA while adding complementary nucleotides to the new forming RNA. Finally, in ter-

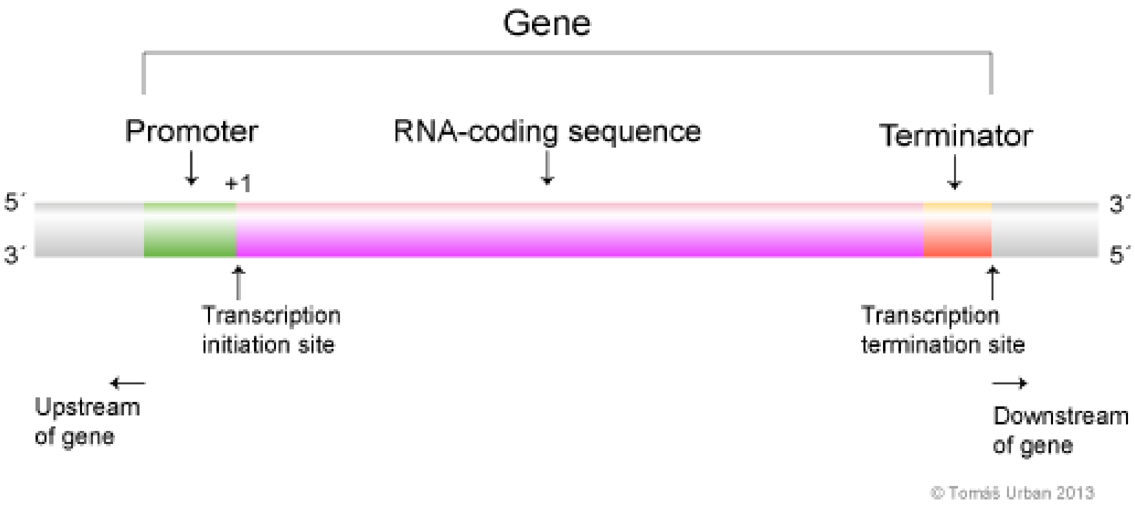


Figure 2.2: Protein coding gene structure[3]

mination, the RNA polymerase reaches the terminator, pre-mRNA is completed and it disconnects from both RNA polymerase and DNA.

The mRNA that is formed during transcription contains coding and non-coding sections, exons and introns, respectively. Since only exons contain information on how to synthesize a protein, introns are then removed by spliceosomes in a process called splicing [CH16]. (Figure 2.3)

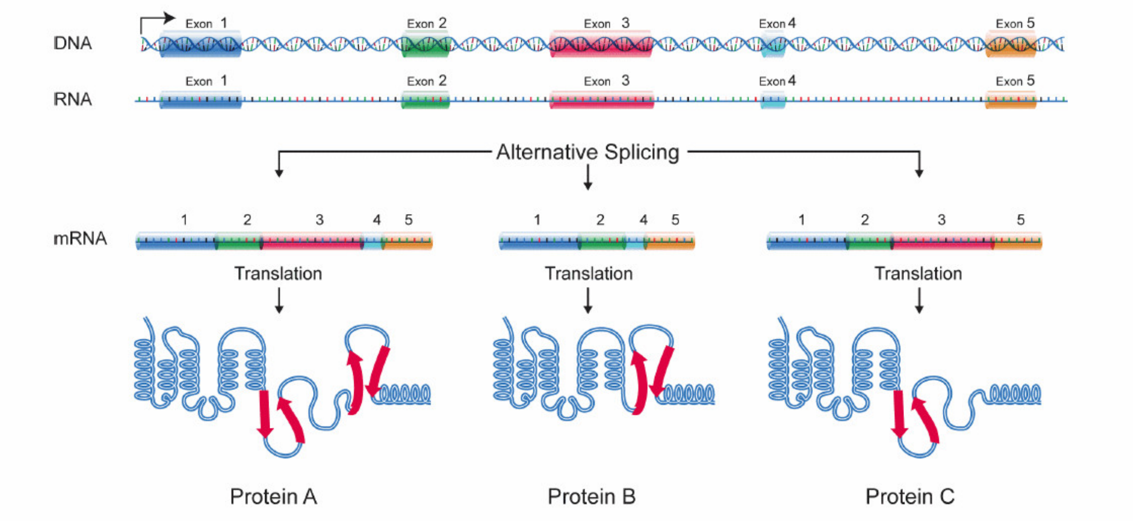


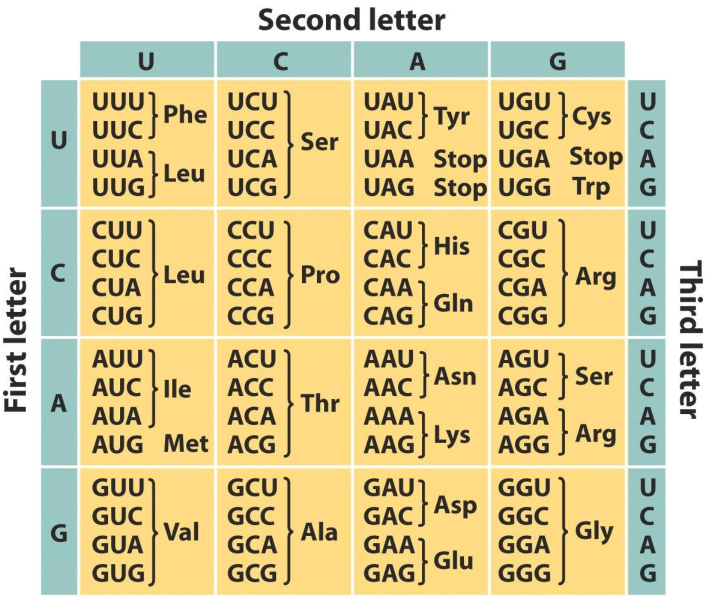
Figure 2.3: DNA Splicing[4]

After this stage, the pre-mRNA (precursor messenger RNA) is now matured and contains only coding information that is ready to be translated. This new formed mRNA contains groups of three nucleotides called codons. Each codon translates into a specific amino acid according to the

[3] Image taken from: http://web2.mendelu.cz/af_291_projekty2/vseo/print.php?page=307&typ=html
[4] Image taken from: https://en.wikipedia.org/wiki/Alternative_splicing

genetic code table, except for four codons: AUG, the start codon, and UAA, UAG, UGA, the stop codons [CH16]. (Figure 2.4)

The translation process is also divided into three stages: Initiation, elongation and termination. The initiation stage begins with the small subunit ribosome scanning the mRNA to find the start codon. Then, the initiator tRNA, which contains the amino acid corresponding to the codon, connects to the start codon and the large ribosomal subunit connects to form the initiation complex. After the initiation is complete, the elongation starts. In this stage, tRNA (transfer RNA) connects to the subsequent codons, one at a time, and a chain of amino acids is formed as the ribosome moves along the strand. When the ribosome reaches a stop codon, the polypeptide is released and the complex is dissociated so that the process can start again at initiation.



First letter	Second letter								Third letter
	U		C		A		G		
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	Stop	UGA	Stop	A
	UUG		UCG		UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

Figure 2.4: Genetic code table[5]

2.2 Biological Pathways

A biological pathway is a sequence of interactions between molecules in a cell that result in a certain product of changes in a cell [Bio]. Those interactions aim to control the flow of information, energy and biochemical compounds in the cell and the ability of the cell to change its behavior in response to a stimuli.

There are several types of biological pathways with the most commonly known ones being metabolic, signaling and gene-regulation pathways. These are very important to understand the mechanisms that originate a disease as they provide clues on what genes, proteins and other molecules are involved in the pathway. Comparing two pathways, from a healthy person and from a person with a disease, researchers can find what triggered such disease.

[5] Image taken from: http://www.bio.miami.edu/dana/250/250SS13_9.html

2.3 Sequencing RNA

Sequencing can be defined as figuring out the order of the nucleotides in an orgamism's DNA. Sequencing methods have been continuously improving throughout the years [HC16]. From taking years and billions of US$ to sequence a human genome using Sanger's methods like in the Human Genome Project, which took thirteen years to be completed and US$ 3 billion [Hay14], to being able to sequence a human genome in a day for less than US$ 5000 using next-generation sequencing [SLS+10].

Next-generation sequencing (NGS), otherwise known as deep or massively parallel sequencing refers to the technological advances in sequence techniques that enable a huge number of sequence reads[6] per run [Met09]. By allowing an high throughput and decreased cost compared to other sequencing technologies [Hay09], this method gained popularity among researchers.

Such advances gave birth to a new technique called RNA-Seq (RNA Sequencing) [WL09]. RNA is of huge importance when it comes to gene expression, combining that with NGS techniques enables the profiling of whole transcriptomes while offering a cost-effective way of measuring genome-wide expression..There are plenty of next-generation sequencing platforms, each one with its own use However, the most commonly used is the Illumina/Solexa[7].

These advances also enabled new projects like [AA+15] to be completed, resulting in more data available for analysis. Nevertheless, it is still a challenge to analyse the huge datasets we get from the above mentioned technology as seen further ahead in this thesis.

2.4 Transcriptome assembly

A transcriptome is the collection of all RNA present in a cell or tissue and the corresponding quantity. RNA sequences are "mirrors" of the DNA from which they were transcribed, considering Thymine is replaced by Uracil, and such, by analysing all RNA sequences in a cell (transcriptome), we can estimate which genes are more expressed in a cell or tissue [WGS09]. That information is of major importance, for instance, if a gene is highly expressed in a cancer cell that can mean that it could be associated with the growth of the cell, giving the researchers some insight on gene functions that were not previously known like in [TJWJ11].

RNA sequences are obtained experimentally using sequencing methods and are assembled in partial or complete transcriptomes.

For many years, microarrays were the standard tool for genomic-wide transcriptome analysis. This approach, however, has some downsides compared to the recent RNA-Seq methods. This is mainly because RNA-Seq works with both reference-based assembly and *de novo* assemblies [WGS09]. In reference-based methods the aligning of the short reads is made with a reference genome or transcriptome while in *de novo* method we can obtain a transcriptome without the aid of a reference genome. This approach gained a particular interest among researchers because

[6] A read is a fragment of a a genome/transcriptome obtained by sequencing methods

[7] Illumina: http://www.illumina.com/

it widened the possibilities of DNA analysis [CLS+11] and the creation of new transcriptomes for organisms that have an incomplete reference genome or don't have one yet at all. Hence, researchers developed various ways of improving this approach [EB+11].

2.5 RNA-Seq Analysis

A single run from any sequencing platform generates a quite big amount of data. For instance, the average human genome size can go up to 200GB of data [GEN] which represents a challenge when it comes to computational resources. That said, powerful computational resources and good algorithms are a must when performing the analysis.

RNA-Seq can be used for various research, such as findings in alternative splicing [PSL+08], gene-expression quantification and transcripts [CLS+11, MWM+08], detecting gene fusions [EMK+11, PTS+10] and some more variations.

RNA-Seq Analysis workflow

The RNA-Seq analysis workflow consists, broadly, of the steps shown in (Figure 5).

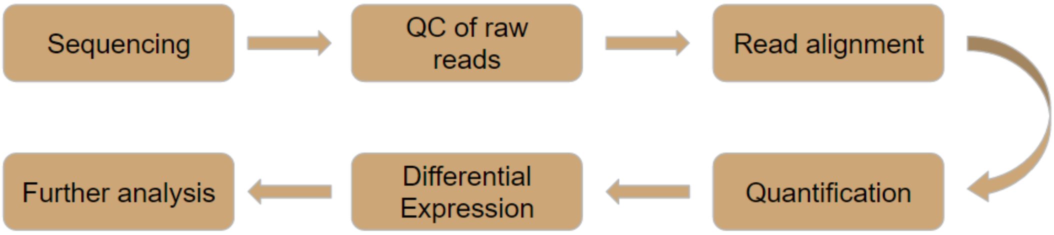


Figure 2.5: Typical RNA-Seq Analysis workflow

Before explaining the concepts of each step, one should have in mind that these steps are not always followed. Nodes can be added and/or removed depending on the goal of the analysis.

After obtaining the raw reads in the sequencing process the reads are subjected to a quality control step. In this step tools are used in order to "clean" the data and improve further stages performance. These tools are used to discard low-quality reads, trim adaptor sequences[8] and eliminate poor-quality bases [CMT+16].

After the quality control step, the reads will be aligned with a *de novo* assembly, if the reference genome is not present, or with the reference genome or transcriptome otherwise.

Following the alignment step a quantification is performed. In this step reads are aggregated in order to calculate gene expression values to be then compared with other samples values in the differential expression step in order to know what genes are most expressed.

[8] Single or double-stranded chemically synthesized short fragments of nucleic acids that can be ligated to the ends of DNA or RNA molecules

Gene analysis

After having identified which genes are differentially expressed, performing an analysis is the core step to extract any meaning from the result. Gene Ontology (GO) project provides a set of structured, controlled vocabularies for the community use in annotating genes and gene product attributes across all species [GO07]. These, so called GO terms, have a name, a unique alphanumeric identifier, a definition with cited sources, and a namespace with the domain where it belongs. Gene Ontology is structured as a direct acyclic graph where each term is connected to one or more terms in the same, or other domain. These terms cover three domains:

Cellular component, the parts of a cell or its extracellular environment.

Molecular function, the elemental activities of a gene product at the molecular level, such as binding or catalysis.

Biological process, operations or sets of molecular events with a defined beginning and end, pertinent to the functioning of integrated living units: cells, tissues, organs, and organisms.

The assignment of a GO term to a gene product is called annotation which are useful to understand the relationships between genes on a gene set. This process is called gene enrichment analysis and can be performed using various tools that provide enrichment capabilities, for instance, PANTHER[9], DAVID[10] or gProfiler[11].

Tools for RNA-Seq

In this subsection we will present some of the most relevant tools for the RNA-Seq steps described previously.

FastQC

FastQC[12] is a quality control tool used to clean data coming from next-generation sequencing methods. It accepts any FastQ, SAM and BAM files, which are file formats that will be described with more depth below in this thesis, and provides an overview of the problematic areas in a form of graph or table as well as the option to export the results as an HTML (HyperText Markup Language) report.

T-Coffee

T-Coffee is a multiple sequence alignment tool that can be used to align sequences or join multiple sequences from different alignment methods [NHH00]. It uses a progressive approach that consists in aligning the two most closely related sequences and then successively align the next most related

[9] http://pantherdb.org/

[10] https://david.ncifcrf.gov/

[11] http://biit.cs.ut.ee/gprofiler/

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

ones. Although it does not guarantee an optimal solution, this approach is efficient enough to implement on large scale datasets.

BLAST

BLAST, which stands for Basic Local Alignment Search Tool [AGM+90], and is a widely used tool for sequence searching. It finds regions of similarity between sequences by comparing those sequences to sequence databases and calculates a statistical significance between them.

Bowtie

Bowtie is a short read alignment tool. It is meant to be a memory-efficient option that works best when aligning large sets of short reads to large genomes. Bowtie indexes the reference genome using a method called Burrows-Wheeler transform to perform the alignment to make the alignment memory efficient, that can take advantage of multiple processors in order to speed up the alignment process.

Bowtie 2 [LS12] is a new improved version of Bowtie that achieves an higher speed, sensivity and accuracy by using dynamic programming algorithms and a full-text minute index.

TopHat

Tophat [TPS09] is a mapping tool for RNA-Seq reads. It uses Bowtie as a short read aligner and then analyses the mapping results to identify splice junctions between exons. While other alignment tools rely on known splice sites, TopHat enables the discovery of new ones by mapping to a reference genome after the alignment.

Cufflinks

Cufflinks [TRG+12] is a transcript assembler that estimates their abundance and tests for differential expression and regulation in RNA-Seq samples. Its package contains several different internal tools such as: Cufflinks, that is used to assemble the transcripts, Cuffcompare, used to compare the assembly to known transcripts, Cuffmerge, used to merge the transcript assemblies, Cuffquant, used to quantify gene and transcript expression in RNA-Seq, having the option to save the results as a file that can be latter analysed with Cuffdiff, used to compare expression levels of genes and transcripts and Cuffnorm used to normalize them.

DESeq

DESeq [AH10] is an R package used to analyse count data from RNA-Seq and test differential gene expression. It uses a negative binomial distribution to infer differential signal correctly and with good statistical power as count data is discrete and skewed therefore is not well approximated by a normal distribution.

EdgeR

EdgeR [RMS09] is an R package included in the Bioconductor software packages. It used to perform differential analysis of RNA-Seq expression profiles and implements a wide range of statistical methods based on negative binomial distributions.

iRAP - Integrated RNA-seq Analysis Pipeline

Each step in the RNA-Seq Analysis workflow can be performed with multiple tools, each one having its pros and cons. Integrating them can also be a problem since the input/output of each tool may not be compatible, which can make it hard for someone who does not have the necessary skills to apply these techniques.

In order to solve that problem the iRap pipeline will be desribed. iRap is a pipeline for RNA-Seq that integrates various tools needed for filtering and mapping reads, quantifying expression and testing for differential expression [FPMB14].

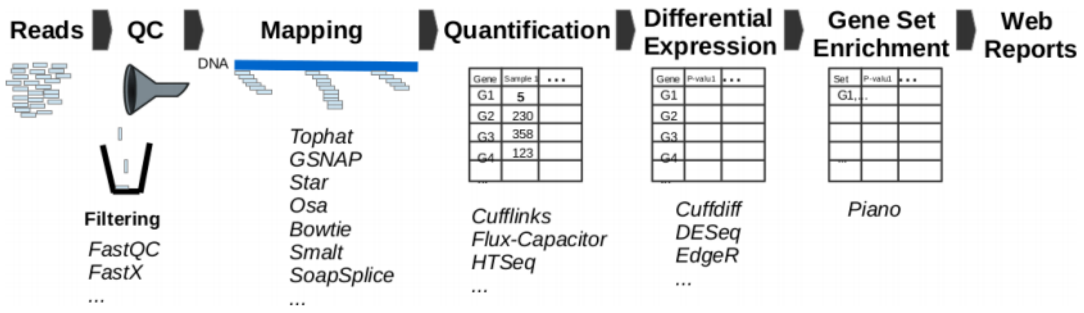


Figure 2.6: iRap pipeline [FPMB14]

As seen in Figure. 6, iRap uses two more steps to complement the analysis of RNA-Seq, gene set enrichment and web reports. The gene set enrichment step is useful in order to understand gene expression data. This method focus on gene sets instead of focusing in single genes [STM^{+}05] which adds more meaning to the results. Web reports make it easier to understand and observe the results produced in each stage of the analysis [FPMB14].

Tools supported by iRap

Table 2.1 lists the tools supported by iRap in each step of the RNA-Seq.

Quality Control	FastQC, FastX
Mapping	TopHat 1, TopHat 2, Osa, Star, GSNAP, Bowtie 1, Bowtie 2, Smalt, BWA1, BWA2, GEM, SOAPSplice, HISAT2, MapSplice
Quantification	HTSeq, Cufflinks 1, Cufflinks 2, NURD, Kallisto, NURD, RSEM, StringTie, Salmon, Flux-capacitator
Differential Expression	Cuffdiff 1, Cuffdiff 2, DESeq, DESeq2, EdgeR, VOOM
Gene set enrichment	Piano

Table 2.1: Available RNA-Seq tools

2.6 File Formats

FASTA

A FASTA format [FAS] file contains text file information concerning nucleotides or peptide sequences. It consists of a description line followed by the correspondent sequence representation. The description line begins with a ">" followed by the name and/or the sequence identifier. This format is extremely simple making the file parsing and conversion easy to do.

FASTQ

The FastQ format is quite similar to FASTA, it also contains text file information about nucleotides. However, it has an extra complementary sequence with the associated per base quality scores [CFG+09].

Each sequence is usually represented in four lines. The first line starts with an "@" followed by its ID. The next line corresponds to the raw sequence letters followed by the third line, containing a "+" with an optional ID. The last line contains the quality scores for the sequence in line two.

This format can be seen as an extension of the FASTA format since it also provides the ability to store quality scores as well as avoiding the big title lines without any line wrapping that would often make common parsers crash [CFG+09]. This format has become the standard for storing the output of high-throughput sequencing instruments like Illumina.

SAM and BAM

SAM stands for Sequence Alignment/Map. This format is tab-delimited and it is used to store sequence data aligned to a reference sequence [LHW+09]. The file consists in a header section and alignment section, and, since it is stored in plain-text it is easy to read and parse. The only thing that differs from BAM to SAM files is their encoding. Instead of using plain text, BAM files store the same information in binary format, thus increasing performance by allowing compression and fast random access [LHW+09].

VCF

Variant Call Format (VCF) is a text file format used to store information about variants found at certain positions in a reference genome [DAA+11]. It was first developed for the 1000 Genomes Project in order to deal with the problem of storing all the information about the genome.

A VCF file contains the header, which includes the file format version and the variant caller version, and the data lines which include the information about the variant.

BED

The BED (Browser Extensible Data) format is used to represent genomic features and annotations [USC]. BED files are tab-delimited and each line contains three mandatory fields, the chromosome name, its starting and ending position, and nine optional fields containing more information about each feature.

GFF and GTF

The Generic Feature Format (GFF) is a text file format used to store gene features. These files are tab-delimited and consist of one line per feature, each one containing nine fields and an optional track line [USC].

Gene Transfer Format (GTF) files are similar to GFF, that is, GTF files are a refinement of the GFF format. The first eight fields are the same but the last one is now expanded into a list of attributes that provide more information about each feature.

2.7 Data repositories

In order to test and validate the results we should use real data to assess the performance of our solution. For that we can use public databases containing genome information such as TCGA[13](The Cancer Genome Atlas), which contains data from hundreds of cancer samples using NGS techniques. This genomic information helps researchers to increase the knowledge about cancer prevention, diagnosis and treatment. There is also another huge source of information called The ENCODE Project[14] which is funded by the NHGRI (National Human Genome Research Institute) and its goal it contains a list of the functional elements in the human genome. In addition to these last repositories we also have GEO[15](Gene Expression Omnibus) that is a public repository for high throughput data at NCBI. It has available tools to help users query and download experiments and curated gene expression profiles.

[13] https://cancergenome.nih.gov/

[14] https://www.encodeproject.org/

[15] https://www.ncbi.nlm.nih.gov/geo/

2.8 Data mining

Data mining can be seen as the process of extracting knowledge and discovering important patterns from data [HPK11]. Data is getting more and more abundant and it seems ever-increasing. Nowadays, due to the ease of acquiring more storage and more computational power we store every little bit of information we would otherwise throw away some years ago. Consequently, as the abundance of data grows and the machines that can perform these type of tasks become commonplace, also does the need of understanding what that data means, as important information can be extracted from it. That same information is what attracted people and companies into using data mining, the search of something new and nontrivial in large amounts of data.

Most existing data mining approaches look for patterns in single data tables, called propositional data mining. While they are useful for not very complex datasets, for datasets which have multi table relations that approach will result in loss of meaning or information. RDM (Relational Data Mining), often referred to as MRDM (Multi Relational Data Mining) is an approach that aims to solve that problem. RDM looks for patterns that involve multiple table relations [Dže03] in order to deal with the complex real world data. These approaches have been applied to a variety of areas, remarkably in the area of bioinformatics like in [FCC11].

Two of the main goals in data mining are considered to be prediction and description [FPSS96, Kan12]. Prediction is when the learning algorithm uses the current data to make future predictions whilst description tries to characterize the properties of the data in a given dataset [HPK11].

There are plenty of ways to achieve these goals and extract useful information from data which will be briefly explained below.

Classification algorithm tries to classify data into finite number of predefined labels (class values) by learning a function that maps objects to the labels.

Regression tries to build a predicting learning function that maps an element to the real-value predicted by the function.

Summarization is a descriptive task that tries to represent data in a more concise way while maintaining the main features of the dataset.

Clustering is a descriptive task that tries to identify a set of clusters or categories to define similar data. There is a clustering subfield called conceptual clustering that aims to not only cluster the data but also discover and explain the meaning behind each cluster.

Anomaly detection tries to find unusual values, that is, values that deviate from what is normal in the dataset. These values are considered outliers and can be either considered errors or interesting values that should be further investigated.

Model-evaluation tries to find dependencies between variables or values of a feature in a dataset.

General approach on problem solving

There is a general procedure to data mining problem solving that involve the five following steps (Figure 2.7):

We first start by stating the problem and formulating an hypothesis followed by collecting the needed data to perform the mining. Before performing it the data should be cleaned in a step called pre-processing. For the extraction to be efficient we should gather only the needed information in order to get better and faster results. After we remove outliers or select the features that we want in pre-processing we can implement and use the appropriate data-mining technique. After the data mining the results need to be validated in order to then draw conclusions on them. We need to be sure that the results that we got from the analysis are relevant and not wrong, otherwise they will be useless and misused.

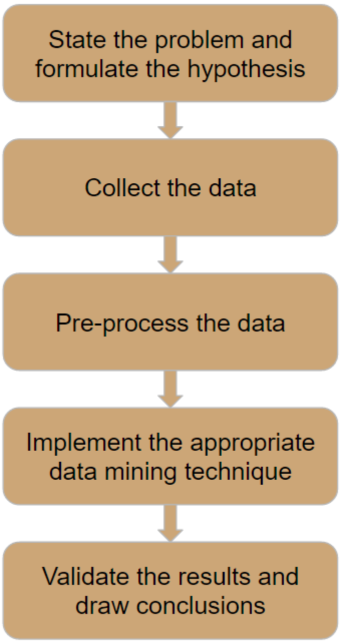

State the problem and formulate the hypothesis
Collect the data
Pre-process the data
Implement the appropriate data mining technique
Validate the results and draw conclusions

Figure 2.7: Data-mining process (Adapted from [Kan12])

Pre-processing

One of the steps in the problem solving pipeline that is of huge importance is data pre-processing. Missing this step would cause the results of the analysis to be misleading.

TCGA gene expression data is of huge dimensionality, given the number of analysed genes, and imbalanced, as it often has a much bigger number of cancer samples compared to the samples from healthy patients.

That leads us to two known problems in data science, dimensionality reduction, and imbalanced datasets.

Dimensionality reduction

Dimensionality reduction is the process of discovering compact representations of high-dimensional data [RS00]. Genomic data is of very high dimensions so a feature extraction method can be useful to extract and analyse its compact representation in comparison with the raw data. The methods that appear to be useful for the problem are the following:

PCA (Principal component analysis) is a linear technique whose goal is to encode high-dimensional data into a lower dimensional representation. It finds the most important information within the data and generates a set of orthogonal variables called principal components that best differentiate the data points [AW10]. In sum, the principle behind this method is the maximization of the variance of the orthogonal transformation of the raw data.

KPCA, or **Kernel PCA**, is an extension of PCA, however, unlike PCA, KPCA tries to find a low-dimensional nonlinear space [SSM98]. It uses kernel methods to compute the principal components in high-dimensional feature spaces.

Autoencoders are a type of artificial neural network that is commonly used for unsupervised learning. Autoencoders try to encode the data and reconstruct it while minimizing the error, finding compact representations of the data. They will be covered in more depth in 2.9.

Imbalanced datasets

More and more we deal with an issue that is often present in nowadays data, namely imbalanced datasets, the lack of samples of one class. When presented with an imbalanced dataset classifiers will often be biased towards the majority class, failing at learn the difference between the majority and minority class. While there is no silver bullet that can solve this challenge entirely, there are some techniques that can be used to help [HG09].

Algorithm modification is a procedure that is more oriented to the adaptation of the base learning methods to be more sensitive to imbalanced classes.

Cost-sensitive learning can be useful when the data is well-known as samples of the minority class can be given more importance compared to sample classes that appear more often.

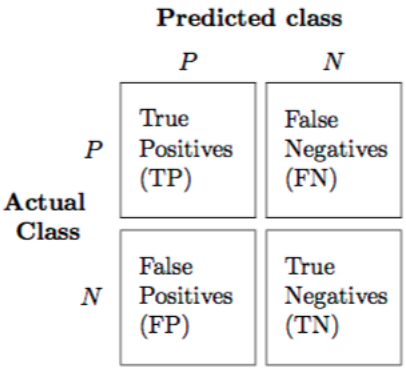
Data sampling is a way of modifying the dataset by some mechanisms in order to balance it. Resampling can be done in several ways, either by undersampling, oversampling or hybrid methods that combine both strategies [LFG+13].

The basic methods involve random oversampling and undersampling where samples are duplicated and removed, respectively. However, these strategies were proven not to be the best since they have no heuristic, as such, downsides arrive with the mentioned problem, random oversampling often leads to overfitting, while random undersampling can lead to the elimination of important samples [HG09].

That said, better approaches have been developed. One such example is SMOTE, synthetic minority oversampling technique [CBHK02]. With this technique, the minority class is oversampled based on the feature space similarities between its samples. To create a synthetic sample, it considers the K-nearest neighbors and randomly selects one to multiply its feature vector difference by a random number between 0 and 1 and adds this vector to the selected neighbor. This approach, however, has some drawbacks as when it is generating synthetic samples it doesn't consider neighboring examples, which can lead to over generalization and increases the overlapping between classes [HG09][LFG$^+$13]. In order to deal with this drawback, adaptive sampling methods such as Adaptive Synthetic Sampling (ADASYN) have been proposed. ADASYN uses a method to create data according to their distributions, that is, more samples are generated for minority class samples that are harder to learn compared to samples that are easier to learn [HBGL08]. In addition to ADASYN, there are also other ways to deal with the issues of SMOTE with one of them being using it with data cleaning techniques like SMOTE + Tomek or SMOTE + ENN [BPM04] that clean the unwanted overlapping between classes after the oversampling, each one using different methods for the task.

Performance measures

Evaluating the performance of a model is one of the core steps in a data mining process. It indicates how successful predictions of a dataset have been by a trained model. The performance of a classification can be evaluated by four different values: the number of correct predictions of class examples (true positives, correctly identified), the number of correct predictions that don't belong to the class (true negatives, correctly rejected), the number of examples that were incorrectly assigned to a class (false positives, incorrectly identified) and the number of examples that were not recognized as being of the class (false negatives, incorrectly rejected)[SL09]. These four values form a matrix, often called confusion matrix that is shown in 2.8 as an example for binary classification.



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Figure 2.8: Example of confusion matrix for binary classification[16]

There are some metrics that are used that are based on the values of the confusion matrix that are listed below:

Accuracy

Accuracy is the overall effectiveness of a classifier.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision

Precision is the proportion of the classes that were correctly identified by the classifier.

$$Precision = \frac{TP}{TP+FP}$$

Recall

Recall is the effectiveness of a classifier to identify positive labels.

$$Recall = \frac{TP}{TP+FN}$$

F1 Score

F1 Score is the harmonic mean of both precision and recall. It is an important metric to use when assessing the performance of classifiers in unbalanced datasets.

$$F_1 = \frac{2TP}{2TP+FP+FN}$$

[16] Image taken from http://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/ (Accessed in 21/06/2017)

Classification techniques

In this subsection we will briefly explain some of the commonly used classification algorithms.

Logistic Regression

Logistic Regression is a statistical method used for analysing a dataset in which there are one or more independent variables that output a binary outcome. Its goal is to find the best fitting model as a linear combination of the predictor variables.

Support Vector Machines

Support Vector Machines are used as a classifier algorithm where a separating hyperplane is defined in order to separate the classes. The hyperplane is optimal when it has the largest distance to the nearest point in both training classes.

Artificial Neural Networks

Artificial Neural Networks are commonly used for classification using a special activation function on the last layer called softmax. In order to perform classification using artificial neural networks the last layer needs to have as many neurons as the number of classes and the activation of the layer will be the probability of the input being of a given class. Softmax function will be discussed further ahead in this thesis.

Decision trees

Decision trees are a method used for classification. The goal is to create a model that can predict the value of a variable given several input variables. In decision trees, each non-leaf node represents input features and each leaf node on the tree represents the resulting value for that variable given the path from the root to the node. There are many algorithms for building decision trees such as ID3 or C4.5.

Naive Bayes

Naive Bayes are a commonly used classification algorithm based on Bayes Theorem. It lies on the principle that every feature that is being classified is independent of the value of any other feature, being that the reason behind being called naive. They are pretty simple to implement and one of the main reasons to use them is the speed. By being given a set of features being probabilities it can predict what is the class that has the highest probability for the given input.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Figure 2.9: Bayes theorem

Data Mining tools

In this subsection we will list some of the most used tools for data mining purposes.

WEKA

WEKA[17] stands for Waikato Environment for Knowledge Analysis. It is an open source tool that contains a collection of machine learning used for data mining tasks. It is written in Java and it is extensible, meaning that we can easily implement new functionalities given the need. It can be used through a command line or directly by its GUI.

RapidMiner

RapidMiner[18] is a complete solution for data mining and machine learning problems. It is used for business and commercial purposes as well as for educational and research ones. It supports all steps of the data mining process detailed in figure 2.7 and its functionalities can be extended through the use of new plugins. This software platform is priced by the amount of data used by the models and its only free up to 10000 rows of data for non-educational purposes while for educational purposes its free to use.

R

R[19] is a statistical computing language for data analysis and graphics. This language can be considered a different implementation of the S language. It contains several statistical and graphical techniques and can be easily extended. It is commonly used among statisticians and data miners given its wide variety of functionalities.

KNIME

KNIME[20] is an open source data analytics that integrates various components for machine learning and data mining. It is popular for its modular pipelining concept. KNIME allows processing of large data volumes and integrates with various open-source projects like Weka and the R project that we mentioned before.

2.9 Deep learning

Deep Learning is a subfield of machine learning, so, before being able to understand deep learning it is mandatory to know what is machine learning and how are computers able to learn. Machine learning is a computer science field that tries to give a machine the ability to learn. Learning is the

[17] http://www.cs.waikato.ac.nz/ml/weka/
[18] https://rapidminer.com/
[19] https://www.r-project.org/
[20] https://www.knime.org/

process of learning something either by experimentation/studying or getting taught. There are two types of learning, supervised and unsupervised.

Supervised and unsupervised learning

Supervised Learning

Supervised is the type of learning where we help the computer determine what it needs to achieve by giving it the desired results along with the input in the training data. The goal of this type of learning is to generalize, that is, it needs to know how to get the correct results for inputs that were not present in the training set. A common problem of this kind of approach is overfitting. Overfitting is when the model learns the data instead of learning the function, so when presented with new data that was not present in the training set it performs poorly because it is too adapted to the training set. It usually happens when the dataset is too small or homogeneous.

Unsupervised Learning

Unsupervised learning is when the desired output is not given during the training. It is used to find the structure or the relationship between the given data, like, for instance, clustering, which is one of the most commonly used unsupervised learning methods.

Deep Neural Networks

Deep learning is a buzzword that derives from artificial neural networks. An artificial neural network is an information processing paradigm that was inspired by the way a biological brain works. Artificial neurons are connected to many others by links that act like axons, thus connecting the output of one neuron to the input of another one (Figure 2.10). These links have numeric weights that represent the strength of the connection between two nodes. Those weights can be tuned based on experience, making neural networks capable of learning based on the input.

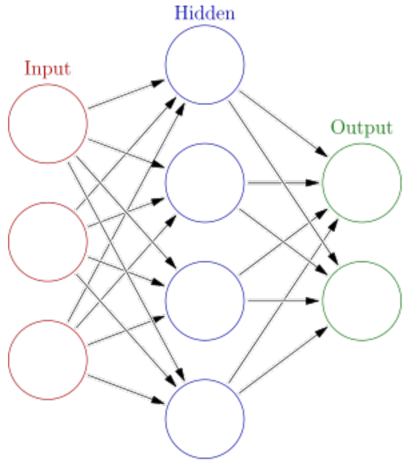


Figure 2.10: Simple Feedforward Neural Network[21]

There is no general concensus on the definition of deep neural network but it can be seen as any artificial neural network that has several hidden layers, more than two most of the times. By having multiple layers these networks are able to learn representations of the data with multiple levels of abstraction [GBC16]. Over the last years this approach has been gaining a lot of interest due to the fact that it outperforms the common used methods for classifying on various datasets [HSK+12] and improved the state-of-art in various fields like speech and image recognition [LBH15] and also in drug discovery and genomics [DGH16].

Training the models - Backpropagation

Backpropagation is the common method to use when training an artificial neural network. It is commonly used with an optimization method called gradient descent. Gradient descent is an optimization algorithm that is used to find a local minimum of a function, called the cost or error functions. There are three variations of gradient descent [Rud16]:

Batch gradient descent

This approach is the standard gradient descent. It computes the computes the gradient of the cost function with respect to the parameters θ as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta)$$

Where we calculate the gradients of the full training set before performing one update, which can be rather slow for big datasets or even very difficult to do when datasets don't fit in memory.

Stochastic gradient descent

With stochastic gradient descent, instead of passing through the whole training set for a single update, it performs a parameter update for each training sample like follows:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; x^{(i)}, y^{(i)})$$

Where $x^{(i)}, y^{(i)}$ are a given sample of the training set.

This approach solves the problem where the dataset is too big to fit in memory and, in addition to this, has a faster convergence, since it avoids the computation of similar values on each parameter update. Nevertheless, it can lead to some fluctuations on the minimization of the objective function.

Mini-batch gradient descent

The mid-term between the above two approaches is mini-batch gradient descent. Instead of updating for every training set sample the updates are performed in mini-batches, a small set of training samples:

[21] Image taken from: https://en.wikipedia.org/wiki/Artificial_neural_network

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; x^{(i:i+n)}, y^{(i:i+n)})$$

Where $x^{(i:i+n)}, y^{(i:i+n)}$ are elements from a batch with size n.

By choosing a small batch of data, this approach reduces the fluctuations of the parameter updates in comparison to the updates from a stochastic approach.

A forward pass and a backward pass of all training examples is called epoch. In other words, an epoch is completed after the network as seen all dataset samples. On the other hand, an iteration is completed after each mini-batch is processed. The number of iterations on each epoch depends on the size of the mini-batch.

In the case of neural networks gradient descent is used to find the optimal value for each weight, since its optimal value is at the global minimum, which sometimes can't be satisfied because gradient descent can get stuck in a local minimum, being one of its limitations. Choosing a learning rate, that is, how quickly the weights are updated, can be rather difficult. A high learning rate can make the loss function diverge, and a small learning rate will lead to a slow convergence. Learning rate schedules [DCM] try to solve this problem by adjusting the learning rate according to some schedule, however, since they have to be defined beforehand, they cannot adapt to the characteristics of the data set. In addition to this, we might want to vary the learning rate according to the frequency of samples of a given dataset, that is, if we have few samples that map to a characteristic we might want to have a larger update in that case. Algorithms with adaptive learning rates deal with the challenges aforesaid and will be briefly described below.

In sum, backpropagation tries to find the minimum of the error function in the weight space using gradient descent. The general steps of the algorithm in a neural network are as follows:

1. Initialize network weights and biases
2. Weights are propagated forward through the network
3. The output error is calculated
4. Compute hidden and input layers weights by calculating the partial derivative of the error function with respect to the given weight
5. Update network weights by multiplying the negative of the computed partial derivatives with the learning rate
6. Repeat until stop condition is met

Gradient Descent Optimization Algorithms

As we mentioned earlier, there are some challenges with respect to the learning rate hyperparameter. In order to deal with that, many improvements on the basic stochastic gradient descent algorithm have been proposed throughout the years. We will briefly explain two of the most commonly used methods below.

Nesterov accelarated gradient

When in areas where the surface is steeper in one dimension more than in another, Stochastic Gradient Descent has some troubles and oscillates around that area making small progress, thus taking more time to converge. Momentum [Qia99] is a method that helps stopping the "zig-zag" when we go down to a local minimum. The current step now depends on both the current gradient and the change on the last step so that it accelerates the convergence and pushes it in the right direction while minimizing the oscillation.

$$\upsilon_{t+1} = \gamma\upsilon_t - \eta\nabla_\theta L(\theta)$$

$$\theta = \theta + \upsilon_{t+1}$$

Where υ_t is the current velocity vector and β is the momentum parameter.

However, when reaching towards the minimum, momentum is often high and it doesn't slow down causing it to miss the minimum entirely and going further to a not so good solution. In order to solve that issue, Nesterov accelerated gradient [Nes83] was created.

$$\upsilon_{t+1} = \gamma\upsilon_t - \eta\nabla_\theta L(\theta + \gamma\upsilon_t)$$

$$\theta = \theta + \upsilon_{t+1}$$

Instead of using gradient at the current location and then taking a big step in the direction of momentum, it first takes a big step in the direction of the accumulated gradient and then makes a correction based on the gradient.

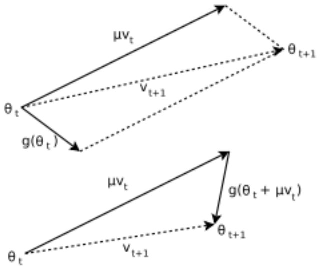


Figure 2.11: (Top) Classical Momentum (Bottom) Nesterov Accelerated Gradient [SMDH13]

AdaGrad

AdaGrad [DHS11] is an adaptive gradient algorithm that adapts the learning rate to the parameters. This is very useful for sparse data where features that are highly informative are not very abundant in the training data. When they appear in the training process they are going to be weighted equally comparing to a feature that is present in a lot of training samples and is not very informative.

AdaGrad tries to solve that imbalance by increasing the learning rate for more sparse parameters and decreasing for less sparse ones.

$$g_t = \nabla_\theta L(\theta)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \odot g_t.$$

Where G_t is a diagonal matrix that contains the sum of the squares of the past gradients with respect to all the θ parameters and ε a small value, usually $10^(-8)$, to avoid division by zero.

ADAM

ADAM [KB14], or Adaptive Moment Estimation is an adaptive gradient algorithm that can be seen as a generalization of AdaGrad. The update rule for Adam is based on the estimation of first (gradient mean) and second (uncentered variance) order moments of past gradients.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_1 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Where m_t and v_t are estimations of first and second order moments, respectively and $\hat{m}_t$ and $\hat{v}_t$ are the correction bias.

Then, the correction bias are used to update the parameters like follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

Cost functions

Cost functions can also be viewed as the loss, or error of a model. In a simplistic way, they are the difference between what the model think is the correct result and the actual correct result. The objective of backpropagation is to minimize this function in order to infer the right weights. Here we will list some of the most common cost functions.

Mean Squared Error

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where n is the total number of items in the training data, y is the true value and ŷ is the estimated value.

The mean squared error function measures the average of the squares of the errors, that is, the difference between the output value we got and what we estimated.

Cross-Entropy

$$C = -\frac{1}{n}\sum_{x}[y\ln\alpha + (1-y)\ln(1-\alpha)]$$

Where n is the total number of items in the training data, α is the output from the neuron, the sum over all training inputs, x, and y is the desired output.

This function is often used when the data is normalized as its results are bounded between 0 and 1 and can be represented as probabilities.

Kullback-Leibler Divergence

$$D_{KL}(P||Q) = \sum_{i=1}^{n} P(i)\ln\frac{P(i)}{Q(i)}$$

Where $D_{KL}(P||Q)$ is a measure of the information lost when Q is used to approximate P, in other words, it measures the "distance" between two distributions. Unlike other Euclidean distances, KL-Divergence is not symmetric, that is, when $Q(i) = 0$ and $P(i)$!= 0.

Activation functions

For any hidden layer to provide any useful information we need to use a non-linear activation function, otherwise it doesn't matter how deep the network is, the results will always yield a linear transformation, which won't produce any useful information given the non-linearity in real-world problems. Here we list some of the commonly used activation functions in this field.

Sigmoid

$$f(x) = \frac{1}{1+e^{-x}}$$

The sigmoid function was known to be the most used activation function. One of the reasons for it is that the derivatives needed for the gradient descent are easy to calculate. Other reason is that they are bounded between 0 and 1 and it can be used as a probability estimation.

There are however some downsides of using this function like the vanishing gradient problem where the gradient gets exponentially smaller in the early layer leading to a slow training process. To solve this problem researchers are using ReLU activation function that we will see below. Another factor to take into account is that it contains exponentials which is an expensive operation and slows down the process.

Tanh

$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

Tanh function, commonly known as hyperbolic tangent, is another type of activation function that is quite similar to the sigmoid. However, this function's derivatives are higher than sigmoid's so it has stronger gradients [LBOM98b]. It also has a greater range [-1,1] than the sigmoid's [0,1], avoiding bias in the gradients. As well as the sigmoid function, it also has the downside of having to calculate exponentials.

ReLU

$$f(x) = max(0,x)$$

ReLU, also known as Rectified Linear Unit is an activation function that has no bounds. It translates any negative input to 0 and all positive values are kept. This function is now the most popular non-linear activation function in use [LBH15]. Two of the main reasons is that it is fast, and it avoids the vanishing gradient problem that the sigmoid and tanh functions have. It has, however, another problem which is usually called the "dying ReLU". This occurs when some neuron always outputs the same value, that is, it gets stuck at 0 because the function gradient at 0 is also 0. One approach to deal with this problem is the the use of a "Leaky ReLU" or the "Exponential ReLU" [CUH15]. Instead of translating a value to 0 if it is negative we can modify the flat side of the function for it to have a gradient and give the neuron a chance to recover.

LeakyReLU:

$$f(x) = \begin{cases} x & ifx > 0 \\ 0.01x & ifx \leq 0 \end{cases}$$

ELU:

$$f(x) = \begin{cases} x & ifx > 0 \\ \alpha(e^x - 1) & ifx \leq 0 \end{cases}$$

Softmax

$$S_j = \frac{e^{z_i}}{\sum_{K=1}^{N} e^{a_k}} \forall j \in 1..N$$

The softmax function is mainly used as the activation function of an output layer using cross-entropy loss for multi-class classification. It takes an N dimensional vector of real values and produces another N dimensional vector of values in the range 0 to 1 that can be interpreted as the probability of a given input being of the given class.

Deep Network architectures

In this subsection we will present some of the most common used deep network architectures.

Deep-Belief Networks

Deep Belief Networks are formed by many Restricted Boltzmann Machines (RMBs). RBMs are composed of only visible and hidden layers. Unlike feedfoward neural networks, the connections between visible and hidden layers are bidirectional, that is, the values can be propagated in both directions. It learns by backpropagation, or more recently by a faster approach called contrastive divergence introduced in [Hin02].

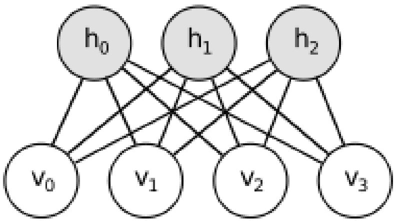


Figure 2.12: Architecture of a Restricted Boltzmann Machine[22]

Deep-Belief Networks can be seen as multiple RBMs stacked on top of each other. Each hidden layer of one RBM is connected to the input layer of the next one. This type of network was first introduced in [HOT06] and are primarily used to recognize, cluster and generate images [HBL+07, BLP+07] and video-sequences [SH07].

Convolutional Neural Networks

Convolutional Neural Networks (CNN) were first based on the organization of the cat's visual cortex. That same organization was replicated as a form of artificial neural net in [LBBH98], establishing the basis of the first CNN. The most popular implementation is called LeNet, named after Yann LeCun. They are mainly used for image recognition and classification. The main steps can be summarized in the following main steps: Convolution, sub-sampling, and full connectedness (Figure 2.13). Its important to note that are plenty of different CNN architectures and the following explanation focus on a simple approach.

Convolution

In this step features are extracted from the input images. A set of filters will be looped through the image to produce feature maps containing different detected features like edges, curves and so on. The more filters we have, the better the CNN gets at recognizing patterns in unseen images. ReLU is typically used after every convolution since convolutions are linear and we need to introduce some non-linearity in order to make it able to learn real-world data. ReLU replaces all negative

[22] Image taken from: http://deeplearning.net/tutorial/rbm.html

[23] Image taken from: http://deeplearning.net/tutorial/lenet.html

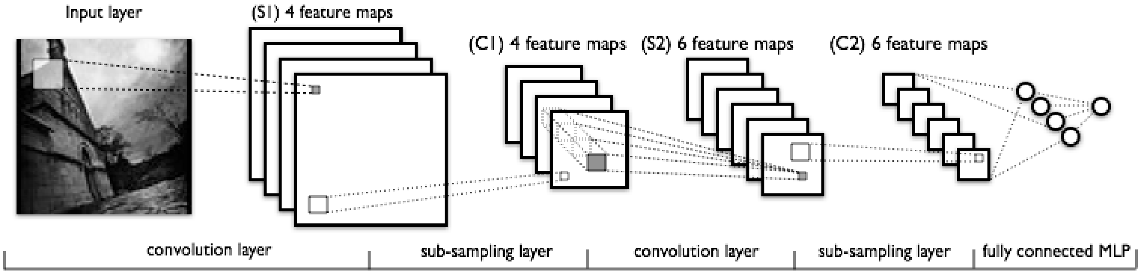


Figure 2.13: Architecture of a simple Convolutional Neural Network[23]

pixel values in the feature map by zero, it has proven to be perform the best in most situations compared to other non-linear functions [KSH12].

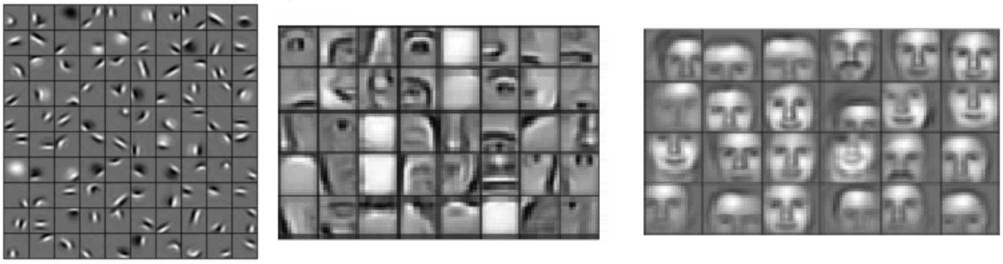


Figure 2.14: Example features learned in a face detection CNN[24]

Sub-sampling

Sub-sampling or spatial pooling is used to reduce the dimensionality of each feature map while retaining the main information. This step can be done using different types of pooling like Max, Average, Sum and so on. For instance, if we have a 2x2 pixel filter and a pixel matrix A = $\begin{bmatrix} 3 & 3 \\ 1 & 5 \end{bmatrix}$ and use Max Pooling we will end up with Â = [5] with Â being the result of the pooling operation on the feature map.

Full connectedness

The fully connected layer is used to classify the high-level features of the input image like we see in the rightmost image in figure 2.14. By adding this fully-connected layer the model can learn non-linear combinations of the learned features.

Recurrent Neural Networks

Recurrent neural Networks (RNNs) belong to a class of artificial neural networks that, unlike traditional neural networks, can contain directed cycles. The idea behind RNNs is to make use

[24] Image taken from: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/

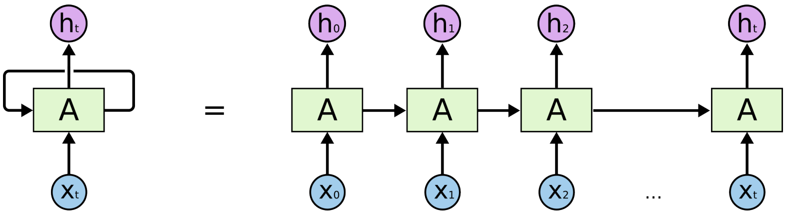


Figure 2.15: An unrolled Recurrent Neural Network[25]

of sequence data. They are used, for instance, on speech recognition problems, translation, gene expression and so on. These networks are called recurrent because they perform the same task for every element of a sequence, the output of a computation depends on the last computations. Image 2.15 shows an example of how a simple recurrent neural networks works in practice. For instance, if we have a four words sentence, the network would be unrolled into a 4-layers neural network, one for each word. Each layer is composed of an input, corresponding to a time step, an hidden state, which can also be interpreted as the "memory" of the network, and an output.

There are several types of RNNs, with the most used one being Long Short-Term Memory (LSTM) networks. LSTMs are very similar to RNN except in the way the hidden state is calculated. Common RNNs have the issue of being incapable of learn such "long-term dependencies", or, more specifically, have a gradient vanishing problem [BSF94]. LSTMs solve the vanishing gradient by introducing structures called "gates" that gives them the ability to remove or add information to the state.

Autoencoders

Autoencoders are symmetric feedfoward neural networks that aim to reconstruct the input. The idea behind an autoencoder is to compress the information and then try to reconstruct it. The basic architecture of an autoencoder is shown in figure 2.16. The basic idea behind it lies on two steps: encode and decode. Encode happens when the features from the input layer are passed into the hidden layer, that usually has less neurons that the input/output layer that usually have the same. By encoding the information the neural network will learn an abstract representation of the input. Then, the information flows from the hidden to the output layer in the decode step where the abstract information is used to reconstruct the input.

Autoencoders extract both linear and non-linear relationships present in the input data, making them power tools for feature extraction [VLBM08]. Stacked Autoencoders are autoencoders stacked on top of each other. The encoder part gradually decreases the dimensionality of the input and learns more compact representations in each successive layer. By reducing the dimensionality step by step, there is a reduction on the loss of information when compared to single autoencoders where the dimension is reduced in one step [BLP$^+$07].

[25] Image taken from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[26] Image taken from: https://commons.wikimedia.org/wiki/File%3AAutoEncoder.png

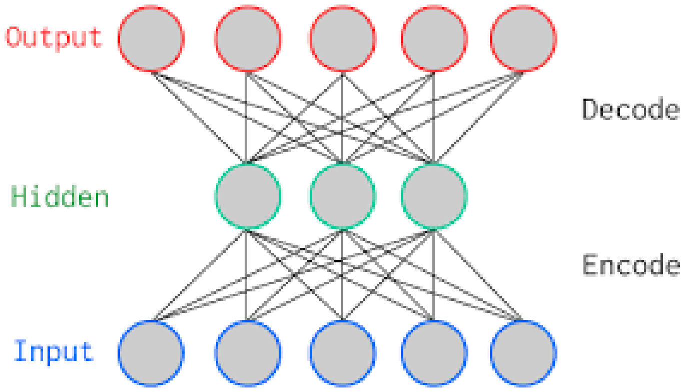


Figure 2.16: Basic architecture of an Autoencoder[26]

Stacked autoencoders have been the choice for pre-training deep networks using a method called greedy layerwise training [KSH12]. It is proven that pre-training the parameters in a deep architecture often leads to much better solutions in terms of generalization performance [VLL+10]. Instead of randomly initializing the weights and bias, a deep network is used to learn these parameters and initialize them near a good local minimum [HOT06][BLP+07].

To get more accurate generalized representation of the data we can use denoising autoencoders [VLL+10]. They are silimar to the ordinary autoencoders except it tries to intentionally corrupt the input data, forcing it to reconstruct the original input from the noisy data.

Deep Learning frameworks

In this section we present some commonly used deep learning frameworks.

Theano

Theano[27] is a Python library for deep learning. It was developed at the University of Montreal for research and development into state-of-the art deep learning algorithms. It handles operations on multidimensional arrays and have several optimizations including the use of GPU of computations. It is, however, low-level, making it harder to use. Nevertheless, there are some libraries that are built on top of Theano, for instance Keras[28], which ease the use of the library by abstracting some concepts.

TensorFlow

TensorFlow[29] is an open source machine learning library developed by researchers and engineers at Google that uses data flow graphs for numerical computations. Mathematical operations are represented by nodes, while the graph edges represent the multidimensional data arrays, as tensors,

[27] http://deeplearning.net/software/theano/
[28] https://github.com/fchollet/keras
[29] https://www.tensorflow.org/

that flow between them. It is cross-platform and has a C++ and Python interface. It also supports distributed computations in multiple CPUs and GPUs.

Caffe

Caffe[30] is a Python deep learning library developed at Berkeley Vision and Learning Center. Its primary focus is convolutional neural networks and one of its benefits is the number of pre-trained networks that can be downloaded and used right away. It supports C++, Python and MATLAB interfaces.

DeepLearning4j

DeepLearning4j[31] is a distributed deep learning framework developed in Java (and JVM languages) that can integrate with Hadoop and Spark (that will be discussed below) using multiple CPUs or GPUs. It supports multidimensional arrays by using ND4J[32] (N-Dimensional Arrays for Java) where its linear algebra computations are performed and it also provides a library, Datavec[33], to transform raw data into vector formats that can be used as input to the algorithms. It covers most of the state of the art deep learning algorithms.

Microsoft CNTK

Microsoft CNTK[34], that stands for Computational Network Toolkit, is a unified deep learning toolkit from Microsoft Research. CNTK scales to multiple GPU servers and is designed around efficiency. It supports C++ and Python interfaces and includes most of the state of the art deep learning algorithms.

2.10 Distributed computing technology and tools

Distributed computing is a model in which multiple computers communicate and work together to solve a single problem like they were a single one. Here we will present two of the main distributed big-data processing frameworks.

Hadoop

Hadoop[35] is a framework that allow the distributed processing of large data sets across clusters of computers. It was inspired by MapReduce[DG08] programming model that is used to tackle large distributed data processing. Hadoop runs of HDFS (Hadoop Filesystem), a distributed file system that is designed to scale up to thousands of machines while being fault-tolerant. Hadoop

[30] http://caffe.berkeleyvision.org/
[31] https://deeplearning4j.org/
[32] http://nd4j.org/
[33] https://github.com/deeplearning4j/DataVec
[34] https://www.microsoft.com/en-us/research/product/cognitive-toolkit/
[35] http://hadoop.apache.org/

also includes another module called YARN (Yet Another Resource Manager) that takes care of job scheduling and cluster resource management.

Spark

Spark[36] is an open source engine for Big Data processing with build-in modules for Streaming, SQL, Machine Learning and Graph processing. Spark can be ran on (EC2, Hadoop YARN or Apache Mesos) and in several distributed storage system (HDFS, Cassandra, Amazon S3, . . .). It can also be used locally, running on one single machine and with one executer per CPU core. Spark applications can be written in Java, Scala, Python and R.

Spark is usually faster than Hadoop MapReduce (up to 100x), one of the reasons is that Spark uses in-memory processing instead of persisting the data back to the disk after each operation (Spark can also preserve data in the disk if the memory is not enough). So as long as Spark has enough memory to fit the data it will usually outperform Hadoop MapReduce, even for Spark jobs that can also be a single Hadoop MapReduce job.

2.11 Conclusions

In this chapter we presented the basic biological and RNA-Seq notions that are necessary in order to understand the goal of the thesis. We also reviewed the concepts behind data mining strategies used to analyse RNA-Seq data and reviewed deep learning methods as well as distributed technologies that can enhance the performance of the analysis.

[36] http://spark.apache.org/

Chapter 3

Methodology

In this chapter we will give a brief overview of the problem as well as some challenges that arise with it. Then we explain our approach to the problem and how we plan to solve it.

3.1 Problem overview

Cancer is the name given to a collection of related diseases, all of them being related to the abnormal cell multiplication and spreading into surrounding tissues. It is known for affecting millions of people each year and continues to attract more and more researchers in the medical field due to the resulting number of deaths of the disease. Cancer research focus on prevention, detection and treatment for the disease and it focus more in some types of cancer that are considered to be more fatal.

Morphological and clinical based prediction of cancer has some limitations, tumors with similar characteristics can follow significantly different clinical paths and show different responses to therapy. To overcome this problem and gain a better insight into the issue, gene expression analysis has become the choice throughout the years, in terms of cancer research. Genomic data has a lot of potential when it comes to novel discoveries that can revolutionize the field of molecular biology. The genomics of tumors are studied in order to understand the molecular basis of the disease to obtain key information on prevention and treatment of the diseases or drug discovery. With the advances of sequencing techniques in the last decade, namely the next-generation-sequencing approach of high-throughput RNA sequencing, huge amounts of gene expression data have been collected and are publicly available.

Statistics, data mining and machine learning have given us the possibility of a deeper analysis and comprehension of gene expression data that is known to contain fundamental information related to cancer diagnosis and drug discovery. While classifiers are useful to differentiate between classes, they are not that useful in real scenarios as most of the them do not help in therapeutic decision making. On the other hand, some classifiers can have some biological relevance, that is, they can be used to find hidden patterns on the data that can be used to find predictive markers, which can be a lot more useful. A predictive marker, either a gene or a protein, provides information about how beneficial a certain therapy can be [Duf05]. The use of predictive markers is becoming more and more relevant in cancer therapy as it can give professionals information about whether a patient will respond positively to a certain therapy or not. There are however some challenges related to the nature of the data that difficult this analysis that will be discussed below.

Concluding, as deep learning is a rather recent approach to a great variety of problems and is getting more and more interest, as it continues to outperform state-of-the-art classifiers [HSK+12] it is important to assess its performance in the genomics field and try to extract biologically meaningful information from the high dimensional noisy data.

3.2 Challenges

While the raw reads are known to have a very large volume, the resulting gene expression values are well within the capability of modern technology. The bigger challenge is the proper analysis, interpretation and reporting of the findings in a way that it provides useful biological information that have real medical utility and can be used for diagnosis. Due to the nature of the data, several challenges arrive when performing a classification analysis on this data.

First challenge comes with the size and dimensionality of the available datasets. All the available datasets on TCGA have less than 560 samples, except for the breast invasive carcinoma, that is the biggest project and contains 1092 samples. On the other hand, the number of genes, or attributes, is huge: there can be more than fifty thousand genes for each sample. Mapping each sample in this very high dimensional space results in very high sparse data. Most classifiers were not designed to deal with this type of issues. The high dimensionality and sparseness of data presents a big challenge for most classification algorithms and often leads to overfitting. In deep learning, particularly, the low number of samples presents a big challenge as this method is very data hungry, that is, the more data it has, the better it performs. However, deep learning has been successfully applied in large-scale, distributed scenarios where a huge amount of very high dimensional data is analysed, for instance, in [Le13], a sparse autoencoder designed for feature detection with 10 million 200x200 pixel images and one billion parameters was trained in a distributed way for 3 days on 1,000 CPUs with a 16,000 cores or a Deep Belief Network like [DCM+12] that was trained for speech recognition on 1,000 CPUs using 42 million parameters for 3 days, both achieving state-of-the-art results. There is, however, the need of great computational power given the number of attributes in order to get results in an acceptable time. Therefore, developing an efficient and effective approach to deal with cancer classification is not an easy task.

The second challenge involves dealing with the huge amount of useless information within the data. From the thousands of genes that form a sample, only a few are relevant and contain useful information. Most of the genes are not cancer related and thus make the job of the classifiers more difficult by increasing the computational time and by making it more difficult to select the relevant genes. This problem can be handled by performing some type of gene selection to select the most relevant genes.

The third challenge, already mentioned before, is related to the usefulness of the classification and feature reduction methods. While accurate prediction is important, it is not the main point of the research. Being able to extract meaningful information from these methods is an important aspect as any biological information will give biologists key information on how the genes are related to a certain type of cancer or gain information on what genes are under or over-expression in a cancerous tissue. Its this information that researchers look for when studying gene expression and its why high accuracy classifiers themselves are not enough, the ability to reveal important biological information is the key aspect.

3.3 Research methodology

Given the importance of not only having a high accuracy, but also revealing important information during the process, we wanted to assess the performance of deep learning models on that objective.

Artificial neural networks have always been seen as "black boxes" as they are believed to deliver very little information on the contribution of the variables [OJD04]. However, there are some methods that aim to reveal the contribution of each variable in an artificial neural network. By knowing which variables contributed the most for the output we can create a subset of the highest ranked genes and analyse the subset in order to find hidden relations between them.

Before feeding the data to the models, we wanted to assess the performance of oversampling methods. For that we ran a quick experiment in which we tested their performance and used the method that resulted in the best performance in further experiments.

Then, the approach to the problem was the construction of a denoising autoencoder for unsupervised feature learning. By forcing the autoencoder to learn a compressed representation of the data it can discover interesting relations from data. Nevertheless, that information can't be directly interpreted as it won't have any real meaning. We tested the quality of the generated features by feeding them to a shallow artificial neural network in a supervised manner to test their accuracy. We then proceeded to construct a stacked denoising autoencoder using the same strategy. By stacking denoising autoencoders with successively small layers we can preserve more features and extract more non-linearities. We also tested the accuracy of other feature reduction methods like PCA and KPCA for comparison. In sum, our first step was to assess the accuracy of the generated features by different methods.

After having high accuracy models we wanted to extract meaningful biological information from them. We did that with both denoising autoencoders and stacked denoising autoencoders. In the first case, we extracted the genes which had the higher weights on the connection to the input layer. however, in a stacked denoising autoencoder we have several levels of connections. In this case, we used one of the aforementioned methods for finding what input variables (genes) had the most influence on the hidden representation constructed by the model.

After knowing which genes had higher importance to the hidden representations of both models we performed GO enrichment analysis to understand the underlying relationship between the subset of genes.

In addition to that, we also extracted all known homo sapiens biological pathways and their correspondent genes from the KEGG database and used the same number of hidden nodes as the number of pathways in several different architectures to see if any of the nodes of the generated features would map into the pathways. We also repeated this procedure with a higher number of hidden nodes to check for unknown pathways.

3.4 Chapter conclusion

In this chapter we started by giving a brief overview of the problem and the importance of the subject. Then, we discussed three of the main challenges in analysing genomic data, namely the low number of samples, noisy and high dimensional data, and the difficulty in extraction biological meaning from it. Finally, we explained the methodology we followed in order to solve each of the aforementioned issues.

Chapter 4

Implementation and Results

4.1 Implementation

This chapter presents the implementation details of the performed experiments and their corresponding results.

Tools

We used Maven to control our dependencies like DeepLearning4j, the Java library that we used to build and train our models. Maven [1] is a build automation tool mainly used for Java projects and one of its main features is the dependency management. It dynamically downloads Java libraries and Maven plugins from a Maven Central Repository which are specified in a configuration file called POM (Project Object Model) that provides all the configuration needed to run the project like the name, dependencies and plugins.

We choose DL4J to train our neural networks mainly because of GPU support and the integration with Spark, which could be a huge improvement on the speed of the experiments. However, due to hardware reasons we had to train our networks without Spark and on CPUs, which presented to be a big issue on performance.

We also used Python[2] as our scripting language for data pre and post-processing. Python is a general-purpose, dynamic and strongly typed programming language that emphasizes usability. It is open-source and runs a successful community based development model, which translates in more documentation, ease of use, and a huge number of powerful packages that were essential to the experiments, namely Numpy [vdWCV11], Pandas[3], Seaborn [4] and Sci-kit learn [PVG+11].

[1] https://maven.apache.org/

[2] https://www.python.org/

[3] http://pandas.pydata.org/

[4] seaborn.pydata.org/

Working environment

In this work we planned to use a Spark cluster to train our models using DL4J, since we had some limitations on the customization of the hardware we used the cluster machines as single working nodes.

We mainly worked with two machines, since the remaining machines of the cluster did not have enough RAM to run our experiments. We used the first machine to run our stacked denoising autoencoder experiments, and the second to run the experiments related with less complex architectures. Below we present a table containing the specs of the machines we used.

	Machine 1	**Machine 2**
Operating System	Scientific Linux 6.9 (Carbon)	Scientific Linux 6.9 (Carbon)
CPU	AMD Opteron™Processor 4180	Intel®Xeon®CPU E5520
CPU Speed	2.27Ghz	2.6Ghz
CPU cores	6	4
GPU	MGA G200eW WPCM450 32MB	MGA G200eW WPCM450 32MB
RAM	62Gb	47Gb

Table 4.1: Specifications of the machines used to perform the experiments

Gene expression data

Our first intention was to generate gene expression data from raw reads, however, since we had limited time and this step was not the main goal of this work, we will be working with gene expression data that is available from public repositories.

We used the papillary thyroid carcinoma dataset from TCGA repository. It contains 510 cancerous samples and 58 healthy samples and the dataset consists in three different TSV files. One with 568 samples with 60483 gene expression values each, one containing the metadata about the samples, and one containing the classification of all samples, whether it is a healthy or cancerous sample.

Data pre-processing

Before feeding the data into the models the data needs to be pre-processed and converted to numerical values. Figure 4.1 illustrates the pipeline that will be described below. These steps are reproduced five more times, one for each fold.

Data preparation

First, we started by joining the file with gene expression values with the corresponding labels by joining the samples on the experience identifier. Then, we changed the labels to numerical values so that the model could interpret them. We associated cancer samples with the label 0, and

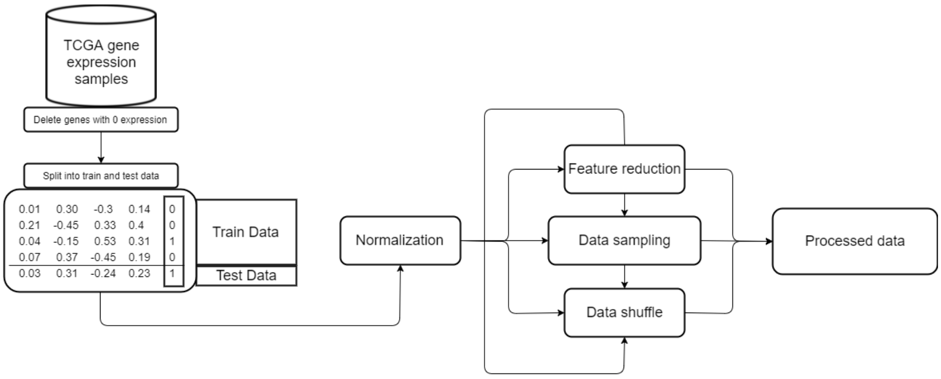


Figure 4.1: Data pre-processing pipeline

healthy ones with the label 1. After having the file assembled we started by removing all genes which expression values were null across all samples, reducing the number of genes from 60483 to 57490. We then divided the dataset in train and test data. However, since we had few healthy samples and a high number of cancer samples, we divided the dataset in four parts: two partitions for each train and test sets, one being the healthy samples, and other being the cancerous samples. The reason behind this division was that there was the possibility of all the minority class samples ending up in the train set or the test set, which could make learning impossible. After the division we distributed the samples equally between the main train and test sets being 80% and 20% of the whole dataset, respectively.

In the next step we transformed each set to $log_2(x+1)$ to represent relative changes in expression and normalized all the values using the training set estimation, as the test set is supposed to play the role of future unknown data. If we normalized all values and then split the dataset into train and test, we would be biasing the model.

Feature reduction and class imbalance

In this work we wanted to assess the performance of deep architectures feature reduction. For that we trained denoising autoencoders (DAE) and stacked denoising autoencoders (SDAE) on DL4J and used the generated features as input for classification. We compared that approach to other feature reduction methods like PCA and KPCA. For both PCA and KPCA we used the implementation from the scikit-learn python package.

When looking at the dataset we noticed that we were dealing with an imbalance problem, we have a high number of cancer examples compared to healthy ones, which can have negative effects on how the network learns. For that, we applied different sampling techniques such as SMOTE with TOMEK and ENN, ADSYN (using imbalanced-learning python package [LNA17]) and no sampling for future comparison.

Finally, we shuffled the data so that our sample classes appear in a random order instead of appearing in batches of the same class, which could result in slower convergence [Ben12].

Architecture details

In our experiments we used three different architectures for our models. A shallow artificial neural network, a Denoising Autoencoder and Stacked Denoising Autoencoder. We will give a brief review of some of the details that are common in the architectures used in our work.

Architectures

The shallow artificial neural network, that is, an artificial neural network without hidden layer, having the input directly connected to the output, was used to classify the approaches using PCA and KPCA for feature reduction in order to compare them to the use of an autoencoder to perform the task. We used denoising autoencoders to increase the generalization performance of our models. Introducing noise on data and forcing the autoencoder to reconstruct the original data from the noisy version can lead to better results when comparing to autoencoders with no noise [VLBM08]. We also used a deeper architecture to extract features. By using stacked denoising autoencoders, instead of reducing the dimensionality in one step like in a single layer autoencoder, we reduce it step by step, resulting in more rich features and less loss of information [BLP+07].

In addition to introducing noise to deal with overfitting, we also used a method called dropout in our experiments. Overfitting was found to be reduced by randomly omitting features on each training set when training large feedfoward neural networks on small training sets [HSK+12]. We used a small value of 0.9 to make sure our model generalizes well. In DL4J, a value of 0.9 translates in each feature having 10% chance of being omitted. Moreover, as autoencoders are prone to learning the identity function given a high enough number of nodes in the hidden layer, DL4J uses tied weights in their implementation of the autoencoder to also prevent overfitting by imposing the following constraint on the weights: $W_2 = W_1^T$.

As pre-training leads to a better generalization [VLL+10], we found that to be a good fit for the problem involving stacked autoencoders, as it would increase the accuracy of the results of our deep architecture. After the both the single and stacked autoencoder approaches learned their weights we ran a supervised fine-tuning step based on the activations of a softmax layer, updating the parameters of the network to better model the input data given its labels.

Hyperparameters and weight updates

Choosing the right combination of hyperparameters is known to be difficult when it comes to get the best out of deep learning experiments [Ben12]. Apart from the highly model dependent hyperparameters like learning rate or number of layers and the number of nodes of each layer, there are some common characteristics amongst the models used in this work like the weight initialization, the optimization algorithm and the correspondent updater.

For the weights initialization we used a method called Xavier weight initialization that was first introduced in [GB10]. Wrong weight initialization can make the learning harder by making gradients either too small or too large. By having really small weights, makes the signals passing through layers successively smaller until they are close to or even useless, and, by having large weights, which makes those signals grow and be too massive to be useful. Xavier initialization satisfies that condition by maintaining the same distribution of activations. In DL4J this initialization follows [GB10] like the following:

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}}$$

Where $Var[W^i]$ is the variance of the weights for a given layer i, initialized with a normal distribution and n_i and n_{i+1} are the amount of neurons in the parent and in the current layer. We used a random seed for weight initialization that was set to 12345. By using the same seed we can get reproducible results.

We used Mini-batch Gradient Descent as our optimization algorithm using 32 as the size of our mini-batch. For every mini-batch of 32 examples the parameters are updated. Compared to Stochastic Gradient Descent, this approach is often much faster [LBOM98a] and often results in better solutions.

As a loss function for the autoencoder layers we used Kullback-Leiber Divergence. We wanted to measure the difference between the original input and the reconstructed features, and, by minimizing Kullback-Leiber Divergence we can minimize the difference between two distributions, namely, the original input data and the reconstruction. The less difference between them the better the feature selection from the autoencoder.

To conclude, we also tested three different updaters when training our DAE and SDAE, those being Nesterov accelerated gradient, AdaGrad and ADAM. Of all three, ADAM was the one that resulted in a faster convergence. We used the default values of β_1 and β_2, being 0.9 and 0.999, respectively.

Model validation

In order to validate our models we used a method called K-fold cross-validation to make sure that they generalize well when given an independent dataset. The goal is to define a test dataset that is used to test the model after the training phase to avoid problems like overfitting, where the model fails to predict data that falls outside the similarities of the data it trained on.

In K-fold cross validation, the model is randomly partitioned into K equal sized datasets. In each iteration, one of the K datasets is used as a test set, with all the remaining k-1 sets being used as the training set. After all K iterations, the prediction results are averaged in order to produce a single estimation. A higher K will have more variance and less bias, however, the computational costs will be also higher [K+95], since the experiment will have to be rerun K times. On the contrary, a lower K will be cheaper, and have more bias and variance. We chose K to be 5 as we think that it is good enough to avoid any sampling bias and also because we had some hardware limitations as well as limited time.

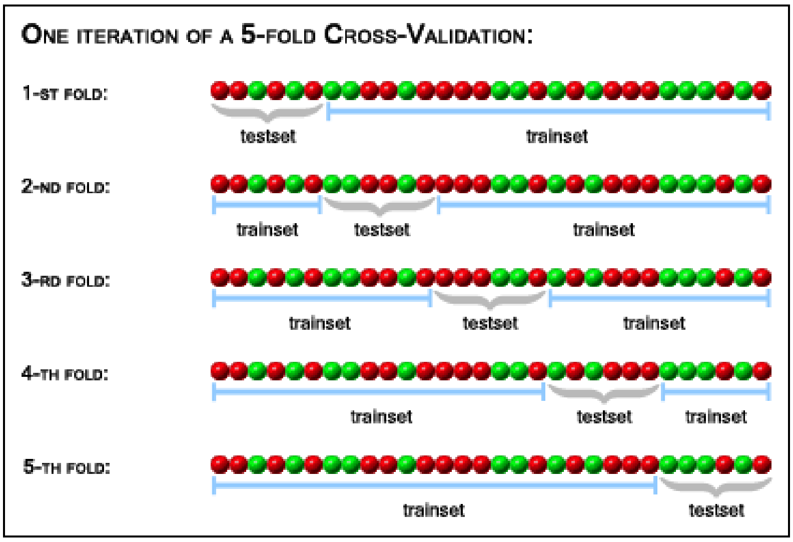


Figure 4.2: Example of 5-fold cross-validation on a dataset with 30 samples.[5]

Deep Learning platform details

As mentioned before, our deep learning platform was build in DL4J using Maven as a dependency manager. We used two plugins, Exec Maven Plugin, to execute our program, and Apache Maven Shade Plugin, to generate an uber-jar containing our project and all its dependencies in a single jar file. Below we list the versions of the core components used to build this project. More information can be found in the POM listed in Appendix A.1

	Version
Deep Learning For Java	0.8.0
N-Dimensional Arrays For Java	0.8.0
Datavec	0.8.0
Java	1.8
Maven	3.5.0

Table 4.2: Versions of the used tools in the platform

We ran the experiments using Exec Maven Plugin from the command line. The code to compile and run is as follows:

```
mvn compile
mvn exec:java -Dexec.mainClass=com.vitorteixeira.GeneExpDL -Dexec.
    cleanupDaemonThreads=false -Dexec.args=ARGS
```

Listing 4.1: Commands used to compile and run a desired experiment

[5] Available from: http://genome.tugraz.at/proclassify/help/pages/XV.html (Accessed 11/06/2017)

The arguments that are passed on ARGS to -Dexec.args argument are the configuration options to run a desired experiment. More detail on the available arguments is on Appendix A.1. Furthermore, depending on the experiment and the machine, we also needed to set the environment variable MAVEN_OPTS="-XmxNgb" with N being the number of gigabytes needed to run the experiments. N was usually set as 30 for simple experiments and 45 to run deep neural network architecture experiments.

After an experiment finished running, the results were saved into a file containing all the information about the used configuration and the respective accuracy, precision, recall and f1 score. If the cross-validation flag was set, the average scores were also logged.

Experimental details

Sampling methods comparison

Given the problem with class imbalance that is inherent to the dataset, it is important to compare the various Sampling methods performance. For that, we built a shallow artificial neural network and, using our pre-processing pipeline, we executed all possible permutations using the available methods to produce five different folds for each: SMOTE+TOMEK, SMOTE+ENN, ADASYN and no sampling method for comparison. We performed this experiment using three different feature reduction methods: PCA, KPCA, and a denoising autoencoder. After verifying which method yielded better performance on our dataset we used it in further experiments.

Assessment of the use of deep architectures for feature reduction

Given the importance of extracting meaningful information from classifiers we wanted to assess the quality of the generated features from various methods. We chose the number of encoded features to be 400 as it seemed good enough for not losing a lot of information and also because PCA and KPCA can only have $N-1$ features where N is the number of samples in the dataset.

We first started by constructing a denoising autoencoder architecture that is represented in Figure 4.3. X_1 and $\hat{X}_1$ are the input data and reconstructed input data, respectively, and H_1 is the node 1 of the hidden layer. The fine-tuning step is not contemplated in the image for simplicity. In DL4J, denoising autoencoders can be constructed using a MultiLayerConfiguration (see Appendix A.3). Apart from the common choices we mentioned before, the remaining hyperparameters for each experiment had to be tuned manually. To get the best configuration we chose the remaining hyperparameters by manually searching a configuration that resulted in the highest average values from cross-validation. For that, we first started by choosing a configuration that yielded satisfying results and then carefully tuned the parameters until we couldn't get any further improvement.

For the stacked denoising autoencoder experiment we followed the same approach as before for the network specific hyperparameters, however, this time, we had to also tune the number of hidden layers and respective nodes. We followed the guidelines presented in [Ben12]. For the number of nodes in the hidden layers, we started by choosing a "pyramid-like" decreasing number of hidden nodes, while successively adding more autoencoder layers until the model performance

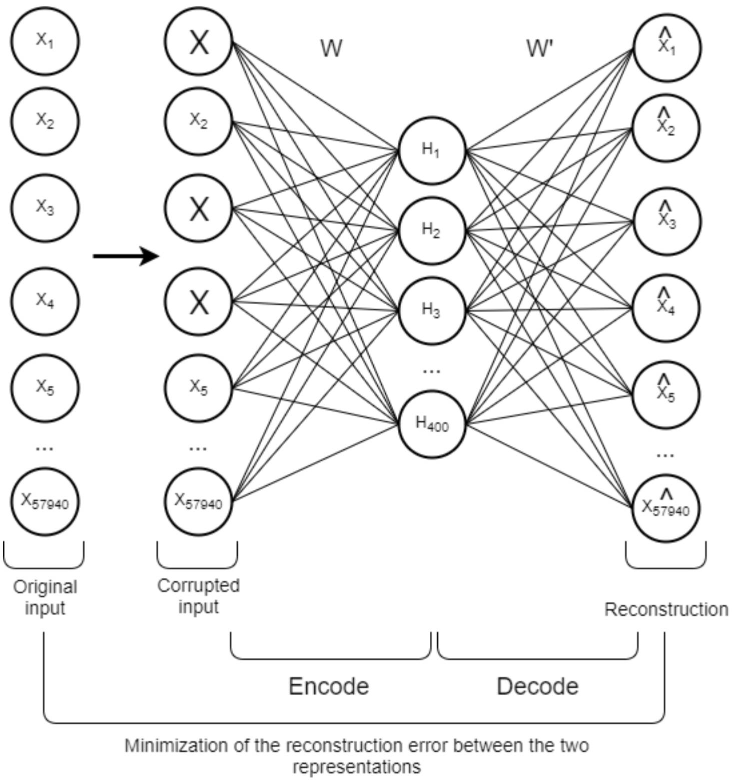


Figure 4.3: Example of a single Denoising Autoencoder

didn't increase anymore. The MultiLayerConfiguration for this experiment can be found in A.4. Figure 4.4 illustrates the chosen architecture for the stacked denoising autoencoder with X_1 and O_1 being the first input and output node, respectively, and H_{1_2} being the second hidden node of the first hidden layer.

For the PCA and KPCA approaches, we used scikit-learn to transform our data before feeding it to the single layer artificial neural network, that translates to the input layer fully connected to a single softmax layer just like in the autoencoder approaches.

After the unsupervised training and fine-tuning, we cross-validated the results and collected the results for further comparison.

Weights analysis

After having a good accuracy classifer we want to check if the learned features had any biological meaning. In order to do so, we extracted the weights matrix from every layer of the autoencoders, as they can help us figure out what were the one that contributed the most for the learned compact representation.

The weights matrix we extracted from the each layer had [$nInput$ x $nOutput$] dimensions, with $nInput$ and $nOutput$ being the number of input nodes for a layer, and the number of nodes for the hidden representation, respectively. In in the case of the denoising autoencoder, this translates into a weight matrix of dimensions 57489 x 400, that contains the weights between input and

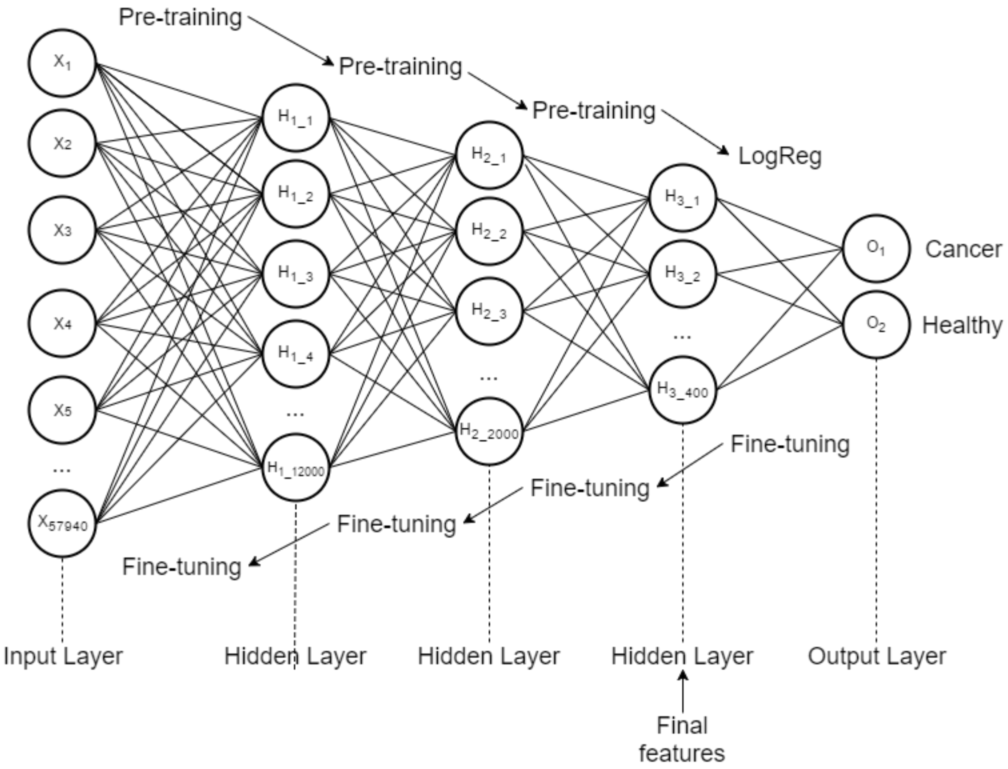


Figure 4.4: Example of Stacked Denoising Autoencoder architecture used in the experiments

output, ready for further analysis. However, in this case of the stacked denoising autoencoder, we have a weights matrix for each layer, so we had to devise a way to calculate the influence of input nodes on the final representation. To obtain a 57489 x 400 weights matrix from a stacked denoising autoencoder, we extracted all layers weight matrix and calculated the products between each successive layer. The product of n matrices $W_1, W_2, W_3, ..., W_n$ with sizes $s_0 \times s_1, s_1 \times s_2, s_2 \times s_3, ..., s_{n-1} \times s_n$ resulting in a $s_0 \times s_n$ matrix is:

$$W = \prod_{i=1}^{n} W_i$$

Our first approach, after we extracted the weights, was to fit the values in a normal distribution and calculate its mean and variance and extract the genes that fell off a certain positive or negative variance threshold. However, after making a more detailed analysis to the data, we verified that the weights connected to the output nodes did not follow a normal distribution but a unimodal skewed or multimodal distribution. Since our approach was not scientifically correct we had to follow other path.

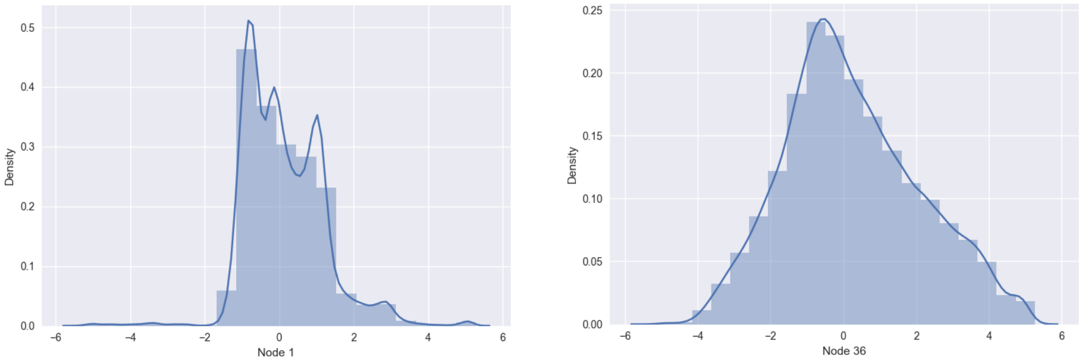


Figure 4.5: Data distribution fitting a kernel density estimation on two randomly chosen nodes

Since the above method didn't work, we now tried to follow [OJD04] method, called Connection Weight, for quantifying variable importance in artificial neural networks. They provided a robust comparison of the performance of different methods for quantifying variable contributions in artificial neural networks and concluded that Connection Weight approach was the best methodology for the task. In the Connection Weight approach we also calculate the matrices product like the above formula. However, this time we sum the products across all hidden neurons in order to maintain both magnitude and sign of the relative contribution of each node. The importance of a given input node X is calculated like follows:

$$Input_X = \sum_{Y=1}^{N} Hidden_{XY}$$

With X and Y being a given input and hidden node, respectively, and N the number of output nodes.

After having calculated the relative importance of each node, we ordered them in descending order so that we could extract the most important ones, using a chosen percentage.

Furthermore, we also used another strategy to calculate what we call high weight genes. This approach also uses the matrices product, however, this time, for each outer node we iterate through its incoming connections and add the top X nodes to a set, with X being a chosen percentage. The algorithm is represented in Algorithm 1.

Algorithm 1: Algorithm used to extract high weight genes

```
input      : Weights matrix of size $inNodes \times outNodes \leftarrow weights\_matrix$
             A list of size $inNodes$ containing the gene ids $\leftarrow gene\_ids$
output     : Set containing the most influential genes for output nodes $\leftarrow gene\_set$
parameters: Percentage of nodes to extract $\leftarrow trim\_percentage$

$gene\_set$;                                                       // Empty set
for $output\_node \leftarrow 0$ to $outNodes$ size do
    $gene\_weight\_tuples$ ;        // Empty list of (gene_id, weight) tuples
    for $input\_node \leftarrow 0$ to $inNodes$ size do
        Add gene ID and its corresponding weight to $gene\_weight\_tuples$;
    Sort $gene\_weight\_tuples$ by $weight$;
    Add $trim\_percentage \times inNodes$ nodes to $gene\_set$;
Return $gene\_set$;
```

After having performed all necessary calculations to have the weights analysed we can generate a list of genes, that we call high weight genes, which will enable a more in-depth analysis of the meaning behind the generated representations. The gene lists that derived from the aforementioned methods are newline separated, that is, they contain one gene per line.

To extract meaning from the resulting lists we used a DAVID[6] [HSL08a][HSL08b] tool called Functional Annotation Clustering. This tool is capable of organizing large gene lists into biologically meaningful groups. Functional annotation charts often have a lot of redundancy due to the nature of annotations. In order to avoid that, this tool cluster similar annotations together which makes the analysis more clear.

After having our newline delimited file, we headed to the tools page on the website and uploaded our data. After uploading the data we selected the type of gene identifiers in our dataset, which is ENSEMBL_GENE_ID, selected the list type as gene list and submitted the query. Then, we had to choose which tool we wanted to use with the dataset, in which we chose the Functional Annotation Tool, followed by the Function Annotation Clustering option. Lastly, a new window opened with the results of the clustering. If enough associations are found, a list of clusters and their corresponding enrichment score. The enrichment score is the geometric mean of all the p-values[7] (in log-scale) in a given annotation cluster. The higher the enrichment score the more important they can be considered for the given study [HSL08b]. DAVID tests significance by performing a hypergeometric test, that is, it tests the null hypothesis that the enrichment of an annotation is purely by chance. This test is measured by the p-value. The smaller the p-value is, the more unlikely the enrichment is purely by chance, meaning it is more significant. DAVID also

[6] The tool is available at `https://david.ncifcrf.gov/`.

[7] A p-value is used in hypothesis testing to help support or reject the null hypothesis. The p-value is the evidence against a null hypothesis. The smaller the p-value, the strong the evidence that one should reject the null hypothesis.

offers the option to show Benjamini corrected[8] p-values, which correct multiple test error when we perform multiple hypothesis tests.

The first two columns of a cluster represent the category and corresponding term in the annotation cluster. The third column indicates the number of genes that are included in that term. Finally, the last two columns represent the modified Fisher exact p-value and the Benjamini correction of the p-value. In [HSL08b] it is advised to give more attention to clusters with more than 1.3 enrichment score, as 1.3 enrichment score is equivalent to 0.05 in non-log scale. However, it is also stated that clusters with lower scores can also be potentially interesting. We chose that same threshold for our cluster analysis.

Pathways analysis

In addition to studying the weights and extract information, we also tried limited down the number of generated features to the number of existing pathways to see if they resembled any pathway.

We first started by downloading all known existing human pathways and its respective genes from KEGG. In order to achieve that, we created a script to parse the list we requested from the KEGG's REST API, which contained a list of tab separated pairs of pathways and their respective genes. Since our dataset contained a list of ENSEMBL genes, and the list we get is made of KEGG IDs, we also had to extract the ENSEMBL ID from each KEGG Gene ID using the KEGG REST API.

After gathering all that information, we extracted the number of existing pathways and how many genes each pathway had. Then, we started by building a denoising autoencoder having the number of hidden nodes equal to the number of existing pathways. After the training phase we extracted the weights and analysed the similarities between each pathway and generated feature from the autoencoder. The algorithm used for this experiment is listed below.

We repeated this experiment several times under several conditions:
First we used a denoising autoencoder with the number of existing pathways as number of features as mentioned above. Second, we repeated the last experiment while increasing the number of nodes. This step was performed to see if the similarities increased if we increased the number of features, which could indicate the presence of the still unknown pathways that could be further analysed. In addition to this, we also kept adding more layers to check if the pathways were mapped in a more high level layer in the network.

[8] The Benjamini-Hochberg procedure is used to control the false discovery rate. Adjusting the rate helps to take to account the fact that sometimes small p-values happen by chance, which could lead to incorrectly reject true null hypotheses.

Algorithm 2: Algorithm used to extract similarities between the extracted features and known human pathways

```
input  : Weights matrix of size $inNodes \times outNodes \leftarrow weights\_matrix$
         A list of size $inNodes$ containing the gene ids $\leftarrow gene\_ids$
         An hashtable containing KEGG pathways and genes $\leftarrow pathway\_table$
output: Number of similar genes between the each node and pathway

for $pathway$ in $pathway\_table$ do
    for $outer\_node \leftarrow 0$ to $outNodes\ size$ do
        Create a list of $(gene\_id, weight)$ using $gene\_ids$ and $weights\_matrix$ ;
        Sort the list on $weight \leftarrow gene\_weight\_tuples$ ;
        $num\_path\_genes \leftarrow pathway\_table[pathway]$ size ;
        Extract $num\_path\_genes$ from $gene\_weight\_tuples \leftarrow splitted\_gene\_list$;
        $count \leftarrow 0$ ;
        for $gene$ in $pathway\_table[pathway]$ do
            if $gene$ in $splitted\_gene\_list$ then
                $count \leftarrow count + 1$;
        Save the number of matches for each node and pathway combination ;
```

4.2 Results

Sampling methods comparison

In 4.4 we present a table containing the results from the experiments, containing the evaluation metrics for all combinations of sampling methods and feature reduction techniques, except for stacked denoising autoencoders. Given our hardware, tuning a stacked denoising autoencoder was rather hard not only because the nature of the algorithm but also because of the running time, excluding the time that took to get a good selection of hyperparameters, it would take almost a week to get the cross-validation results for each of the presented sampling methods.

All experiments were run under the conditions listed below:

Parameter \ Features	Raw	PCA/KPCA	DAE
Number of epochs	50	50	35
Number of output features	-	400	400
Mini-batch size	32	32	32
Learning rate	0.1	0.1	0.1
Updater	ADAM	ADAM	ADAM
Activation function	LeakyReLU	LeakyReLU	LeakyReLU
Dropout	0.9	0.9	0.9

Table 4.3: Hyperparameters used to run the experiment

Sampling method	Features	Accuracy	Precision	Recall	F1 Score
Raw	None	76.07	**65.04**	85.93	74.03
	PCA	98.95	96.77	97.87	97.29
	KPCA	98.95	96.77	97.87	97.29
	DAE	93.48	82.61	91.08	86.37
SMOTE+TOMEK	Raw	97.54	91.12	97.90	94.34
	PCA	99.12	98.04	97.23	97.63
	KPCA	**99.30**	**98.20**	**98.06**	**98.12**
	DAE	98.07	94.60	95.09	94.83
SMOTE+ENN	Raw	95.07	84.18	96.52	89.90
	PCA	**99.30**	**98.20**	**98.06**	**98.12**
	KPCA	98.95	97.87	96.39	97.11
	DAE	95.78	86.67	96.10	91.08
ADASYN	Raw	98.41	96.64	94.63	95.57
	PCA	98.95	97.87	96.39	97.11
	KPCA	98.77	97.77	95.48	96.61
	DAE	97.18	93.46	91.94	92.14

Table 4.4: Resulting performance of the different sampling methods (%)

By analysing the table 4.4 we can conclude that we get rather good results from every combination of feature reduction approach with sampling method, except when we choose not to use any, in which the accuracy and F1 Score results in 76.06% and 74.03%, respectively. We should also notice that while the use of no sampling methods on PCA and KPCA approaches don't result in a noticeable improvement, in the case of denoising autocoders it does result in some improvement due to the nature of the algorithm.

Finally, by observing the results we can conclude that both SMOTE+TOMEK with KPCA and SMOTE+ENN with PCA yield the best performance on the used dataset, having the same result in all four evaluation categories. We decided to chose to work with SMOTE+TOMEK since it also yielded the best overall performance of the denoising autoencoder experiments, which will be our primary focus in this work.

Feature reduction assessment

In this experiment we wanted to assess the richness of the generated features for each feature reduction method. After training and fine-tuning the model we measured the performance of each method on the test set using cross-validation. This time we added a stacked denoising autoencoder for comparison, which we expected to be more time consuming. The hyperparameters used to run the stacked denoising autoencoder experiment and its results are listed below:

Features / **Parameter**	**SDAE**
Number of epochs	5
Number of output features	400
Mini-batch size	32
Learning rate	0.1
Updater	ADAM
Activation function	LeakyReLU
Dropout	0.9

Table 4.5: Hyperparameters used to run the SDAE experiment

Features	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Avg. running time per fold**
Raw	97.77	91.62	97.99	94.66	16 minutes and 30 seconds
PCA	99.12	98.04	97.23	97.63	36 seconds
KPCA	**99.30**	**98.20**	**98.06**	**98.12**	**20 seconds**
DAE	98.07	94.60	95.09	94.83	58 minutes and 36 seconds
SDAE	97.36	94.70	92.05	93.30	3 hours and 31 minutes

Table 4.6: Resulting performance of the different feature extraction methods (%) and their average running time

From these results we can see that the feature reduction method that results in better overall classification as well as running time is KPCA. KPCA is a good choice to boost high dimensionality data classification performance, it does not give us, however, any comprehensible information about the relevant genes that can potentially play the role of biomarkers. Both autoencoder approaches yield more than acceptable results and we believe that they can be used to give some better insights on the data.

As expected, the running time scales either with the number of nodes in a layer, as well as the network size, in this scenario, the number of denoising autoencoders used. From our experience, assuming similar hardware, same hyperparameters and a pyramid like architecture, stacking another autoencoder almost doubles the average running time per fold, resulting in an increased difficulty for training more deep networks.

The denoising autoencoder approach yielded similar but better overall results. We believe that if we had better hardware we could tune it so that it yielded more similar or better results. Nevertheless, it could be that the data has not many non-linearities so a less complex model is more adequate to fit the data.

Weights analysis

Below we present the results of the functional annotation clustering tool from DAVID using the architectures that had the best classification scores and the neuron contribution extraction methods. We performed a small literature research on the information extracted from the method that resulted in the best overall enrichment scores.

Method 1 - Connection Weight approach

Denoising Autoencoder

Table 4.7: Functional analysis clustering using connection weights approach on the denoising autoencoder results having 303 DAVID IDs with a p-value threshold of 0.05

Annotation Cluster 1 **Enrichment Score:** 1.49

Category	**Term**	**Gene count**	**P-Value**	**Benjamini**
GOTERM_BP_DIRECT	DNA repair	6	1.2^{-2}	1.0^{0}
UP_KEYWORDS	DNA damage	7	3.2^{-2}	1.0^{0}
GOTERM_BP_DIRECT	interstrand cross-link repair	3	4.0^{-2}	1.0^{0}
UP_KEYWORDS	DNA repair	6	4.8^{-2}	1.0^{0}

This approach only resulted in one cluster with 1.49 enrichment score after applying the 0.05 p-value threshold. Not much information could be extracted using this combination.

Stacked Denoising Autoencoder

Table 4.8: Functional analysis clustering using connection weights approach on the stacked denoising autoencoder results having 419 DAVID IDs (0.015% of the total number of genes) with a p-value threshold of 0.05

Annotation Cluster 1 **Enrichment Score:** 2.14

Category	**Term**	**Gene count**	**P-Value**	**Benjamini**
UP_KEYWORDS	SH3 domain	11	3.4^{-3}	3.1^{-1}
UP_SEQ_FEATURE	domain:SH3 1	5	4.0^{-3}	9.8^{-1}
INTERPRO	Src homology-3 domain	11	4.2^{-3}	9.5^{-1}
SMART	SH3	11	4.2^{-3}	5.0^{-1}
	domain:SH3 2	5	4.4^{-3}	8.9^{-1}
INTERPRO	Variant SH3	5	2.0^{-2}	9.7^{-1}
UP_SEQ_FEATURE	domain:SH3 3	3	4.8^{-2}	1.0^{0}

Annotation Cluster 2 **Enrichment Score:** 2.06

Category	Term	Gene count	P-Value	Benjamini
GOTERM_CC_DIRECT	collagen trimer	7	4.8^{-3}	7.9^{-1}
UP_KEYWORDS	Collagen	7	6.0^{-3}	3.2^{-1}
UP_SEQ_FEATURE	calcium ion binding	4	2.4^{-2}	9.9^{-1}

Annotation Cluster 3 **Enrichment Score:** 1.65

Category	Term	Gene count	P-Value	Benjamini
GOTERM_MF_DIRECT	superoxide-generating NADPH oxidase activity	3	1.5^{-2}	9.8^{-1}
GOTERM_MF_DIRECT	NADPH oxidase complex	3	1.7^{-2}	9.4^{-1}
GOTERM_MF_DIRECT	electron carrier activity	6	2.0^{-2}	9.3^{-1}
GOTERM_MF_DIRECT	superoxide anion generation	3	2.4^{-2}	1.0^{0}
GOTERM_MF_DIRECT	superoxide metabolic process	3	4.7^{-2}	1.0^{0}

Annotation Cluster 4 **Enrichment Score:** 1.62

Category	Term	Gene count	P-Value	Benjamini
INTERPRO	Major intrinsic protein, conserved site	3	1.2^{-2}	9.8^{-1}
GOTERM_MF_DIRECT	glycerol channel activity	3	1.8^{-2}	9.6^{-1}
GOTERM_BP_DIRECT	cellular water homeostasis	3	2.1^{-2}	1.0^{0}
GOTERM_BP_DIRECT	glycerol transport	3	2.1^{-2}	1.0^{0}
UP_SEQ_FEATURE	short sequence motif:NPA 2	3	2.1^{-2}	1.0^{0}
UP_SEQ_FEATURE	short sequence motif:NPA 1	3	2.1^{-2}	1.0^{0}
INTERPRO	Major intrinsic protein	3	2.9^{-2}	9.7^{-1}
GOTERM_MF_DIRECT	water channel activity	3	3.0^{-2}	9.4^{-1}
INTERPRO	Aquaporin-like	3	2.4^{-2}	3.3^{-2}
GOTERM_BP_DIRECT	water transport	3	4.7^{-2}	1.0^{0}

Annotation Cluster 5 **Enrichment Score:** 1.59

Category	Term	Gene count	P-Value	Benjamini
UP_SEQ_FEATURE	domain:Beta/gamma crystallin 'Greek key' 4	3	2.2^{-2}	1.0^{0}
UP_SEQ_FEATURE	domain:Beta/gamma crystallin 'Greek key' 3	3	2.5^{-2}	9.9^{-1}
UP_SEQ_FEATURE	domain:Beta/gamma crystallin 'Greek key' 2	6	2.5^{-2}	9.9^{-1}
UP_SEQ_FEATURE	domain:Beta/gamma crystallin 'Greek key' 1	3	2.5^{-2}	9.9^{-1}
INTERPRO	Beta/gamma crystallin	3	2.6^{-2}	9.7^{-1}
INTERPRO	Gamma-crystallin-related	3	2.6^{-2}	9.7^{-1}
SMART	XTALbg	3	2.9^{-2}	9.1^{-1}

Annotation Cluster 6 **Enrichment Score:** 1.39

Category	Term	Gene count	P-Value	Benjamini
UP_SEQ_FEATURE	domain:FAD-binding FR-type	3	4.0^{-2}	1.0^{0}
INTERPRO	Riboflavin synthase-like beta-barrel	3	4.1^{-2}	9.7^{-1}
INTERPRO	Ferredoxin reductase-type FAD-binding domain	3	4.1^{-2}	9.7^{-1}

This approach resulted in 6 different clusters with the highest enrichment score of 2.14. Further analysis is needed in order to assess the performance of this method.

Method 2 - Union of the individual contributions (Algorithm 1)

Denoising Autoencoder

Table 4.9: Functional analysis clustering using algorithm 1 on denoising autoencoder results having 407 DAVID IDs (0.00075% of the genes with high weights in each node) with a p-value threshold of 0.05

Annotation Cluster 1 **Enrichment Score:** 2

Category	Term	Gene count	P-Value	Benjamini
GOTERM_MF_DIRECT	exonuclease activity	4	4.3^{-3}	9.0^{-1}
UP_KEYWORDS	Exonuclease	5	9.8^{-3}	4.8^{-1}

UP_KEYWORDS	Nuclease	7	2.4^{-2}	6.3^{-1}

Annotation Cluster 2 **Enrichment Score:** 1.79

Category	**Term**	**Gene count**	**P-Value**	**Benjamini**
UP_SEQ_FEATURE	domain:ABC transmembrane type-1 1	4	2.2^{-3}	7.1^{-1}
UP_SEQ_FEATURE	domain:ABC transmembrane type-1 2	4	2.2^{-3}	7.1^{-1}
INTERPRO	ABC transporter, transmembrane domain, type 1	4	1.2^{-2}	1.0^{0}
UP_SEQ_FEATURE	domain:ABC transporter 1	4	2.1^{-2}	1.0^{0}
UP_SEQ_FEATURE	domain:ABC transporter 2	4	2.1^{-2}	1.0^{0}
UP_SEQ_FEATURE	nucleotide phosphate-binding region:ATP 1	4	2.6^{-2}	1.0^{0}
UP_SEQ_FEATURE	nucleotide phosphate-binding region:ATP 2	4	2.6^{-2}	1.0^{0}
INTERPRO	AAA+ ATPase domain	4	2.7^{-2}	1.0^{0}
SMART	AAA	7	2.8^{-2}	9.9^{-1}
INTERPRO	ABC transporter, conserved site	7	3.7^{-2}	1.0^{0}
GOTERM_MF_DIRECT	ATPase activity, coupled to transmembrane movement of substances	4	4.3^{-2}	9.2^{-1}

Annotation Cluster 3 **Enrichment Score:** 1.67

Category	**Term**	**Gene count**	**P-Value**	**Benjamini**
KEGG_PATHWAY	Fc epsilon RI signaling pathway	6	1.4^{-2}	7.8^{-1}
KEGG_PATHWAY	B cell receptor signaling pathway	6	1.5^{-2}	6.6^{-1}
KEGG_PATHWAY	Osteoclast differentiation	8	2.1^{-2}	5.2^{-1}
KEGG_PATHWAY	Natural killer cell mediated cytotoxicity	7	4.5^{-2}	7.0^{-1}

Annotation Cluster 4 **Enrichment Score:** 1.66

Category	Term	Gene count	P-Value	Benjamini
GOTERM_BP_DIRECT	L-amino acid transport	3	1.8^{-2}	1.0^{0}
GOTERM_MF_DIRECT	L-amino acid transmembrane transporter activity	3	2.1^{-2}	8.5^{-1}
UP_KEYWORDS	Amino-acid transport	4	2.8^{-2}	5.7^{-1}

Annotation Cluster 5 **Enrichment Score:** 1.53

Category	Term	Gene count	P-Value	Benjamini
KEGG_PATHWAY	Homologous recombination	4	2.3^{-2}	5.0^{-1}
UP_KEYWORDS	DNA repair	11	2.4^{-2}	5.9^{-1}
GOTERM_BP_DIRECT	double-strand break repair	5	2.8^{-2}	9.9^{-1}
UP_KEYWORDS	DNA damage	12	3.2^{-2}	5.6^{-1}
UP_KEYWORDS	DNA recombination	5	4.5^{-2}	6.1^{-1}

This approach resulted in 5 different clusters having the highest enrichment score of 2.

Stacked Denoising Autoencoder

We used the method that resulted in the best overall enrichment scores, which was the method presented in algorithm 1.

Table 4.10: Functional analysis clustering using algorithm 1 on stacked denoising autoencoder results having 378 DAVID IDs (0.0015% of the genes with high weights in each node) with a p-value threshold of 0.05

Annotation Cluster 1 **Enrichment Score:** 3.88

Category	Term	Gene count	P-Value	Benjamini
UP_SEQ_FEATURE	repeat:CSPG: 1	4	3.1^{-5}	2.8^{-2}
UP_SEQ_FEATURE	repeat:CSPG: 2	4	3.1^{-5}	2.8^{-2}
	...			
UP_SEQ_FEATURE	repeat:CSPG: 12	4	3.1^{-5}	2.8^{-2}
UP_SEQ_FEATURE	domain:Calx-beta 3	3	1.3^{-3}	3.2^{-1}
UP_SEQ_FEATURE	domain:Calx-beta 1	3	4.3^{-3}	6.3^{-1}
UP_SEQ_FEATURE	domain:Calx-beta 2	3	4.3^{-3}	6.3^{-1}
SMART	Calx_beta 3	3	6.4^{-3}	6.2^{-1}
INTERPRO	Na:Ca exchanger/integrin-beta4	3	7.7^{-3}	7.9^{-1}

Annotation Cluster 2 **Enrichment Score:** 1.75

Category	Term	Gene count	P-Value	Benjamini
UP_SEQ_FEATURE	domain:EF-hand 4	5	1.3^{-2}	8.7^{-1}
UP_SEQ_FEATURE	domain:EF-hand 3	6	1.4^{-2}	8.4^{-1}
GOTERM_MF_DIRECT	calcium ion binding	20	1.4^{-2}	9.5^{-1}
INTERPRO	EF-hand domain	9	2.3^{-2}	1.0^{0}
INTERPRO	EF-hand-like domain	10	2.7^{-2}	1.0^{0}

Annotation Cluster 3 **Enrichment Score:** 1.42

Category	Term	Gene count	P-Value	Benjamini
UP_KEYWORDS	Cell division	12	2.9^{-2}	7.3^{-1}
UP_KEYWORDS	Mitosis	9	4.0^{-2}	7.2^{-1}
GOTERM_BP_DIRECT	cell division	11	4.8^{-2}	9.9^{-1}

This approach resulted in 3 different clusters. Since it had the highest enrichment score we decided to search for connections between the terms and thyroid cancer in literature. From a small research we found that CSPGs, which are present on the Annotation Cluster 1, are known to be involved in certain cell processes such as cell growth, cell migration, cell adhesion, receptor binding and interaction with other extracellular matrix constituents [RF04]. In [WM06], the role of CSPGs in tumor progression was studied and it was concluded that they play a role in tumor growth and metastasis. Furthermore, in cluster 2, most of terms are related to calcium-binding regions or to EF-hand domain, which is the most common calcium-binding motif found in proteins. EF-hand prototeins are known to be related to the growth and metastasis of various types of cancer, including papillary thyroid carcinoma [JTL+15] [ZYH+16]. Finally, the last cluster is related to cell division and mitosis, which is directly related to abnormal cell proliferation.

It should be noted that even though we extracted information that can be considered to be interesting in the given problem, we still need the help of a specialist in thyroid cancer to validate the usefulness of the results.

Pathways analysis

In this experiment we tried to see if there was any relation between the encoded features and pathways. After training our models, while increasing the number of features and the number of layers between experiments, we verified that this hypothesis was not proven to be true. We used the following architectures to verify the hypothesis:

	Number of nodes in each autoencoder layer
Architecture 1	57490-317-57490
Architecture 2	57490-350-57490
Architecture 3	57490-450-57490
Architecture 4	57490-12000-317-12000-57490
Architecture 5	57490-12000-6000-317-6000-12000-57490

Table 4.11: Architectures used in this experiment

In each experiment we assessed the similarity between the high weight genes that contributed to the encoded representation and the genes involved in known human pathways. From that, we concluded that the nodes had no apparent relation with the pathways as the number of similar genes in every combination were not above 0.1% of the total of genes involved in a specific pathway in all experiments.

4.3 Chapter conclusions

In this chapter we presented all the details of the built testing platform as well as all the details about each performed experiment. We ended by reporting and analysing the results for each experiment.

Chapter 5

Conclusions and future work

In this thesis we researched the usefulness of feature reduction methods using stacked denoising autoencoders. We used various approaches for data sampling as well as feature reductions methods, comparing their performance and usefulness. We were able to extract information by studying the weights extracted from the stacked denoising autoencoder.

There is still the need for a specialist in thyroid cancer to analyse the results from the analysis in order to ensure that they have some logical meaning, and to compare and assess the usefulness of each neuron contribution method and type of network. From a small research on our findings we can conclude that this approach has a lot of potential since the knowledge we extracted from the approach that had the best enrichment score had literature relating the information to the type of cancer we studied. If our findings are proven to be useful they will give cancer researchers a shape and direction for a further study even in other types of cancer.

5.1 Objective Fulfillment

Our objective was the assessment of the usefulness and performance of deep learning methods for the extraction of biologically meaningful information from gene expression data. We successfully built a customizable analysis pipeline and analysed the contribution of deep architectures for the extraction of important biological information. That said, we can say we completely fulfilled the objective, but, since this was a research work, there is a lot more work that can be done that will be described below.

5.2 Future work

There is still a lot of work that can and should be done, as this approach for extracting information from gene expression data can be useful to give researchers a direction for investigation. We could first start by trying more combinations of hyperparameters while using better hardware. Hardware was one of the biggest issues we found in this work. The better we could get would be a distributed GPU cluster with high-end GPUs for fast deep network training. Better hardware would also ease the training of other types/ architectures deep networks for the study of other ways to extract information from gene expression data. Moreover, we could also include more neuron contribution algorithms to assess which one produces the most biologically accurate representation, and also use and compare gene enrichment tools to maximize the amount of information we can get from this approach. We could also integrate iRap in our pipeline and use the cluster in order to produce gene expression data from raw sequencing data and build a full pipeline for gene expression data analysis using deep learning methods. Finally, we could perform the experiences with other types of cancers or even build a dataset containing several cancer types to see if we could extract some common biomarkers amongst all types of cancer.

References

[AA$^+$15] Adam Auton, Gonçalo R. Abecasis, et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, sep 2015.

[AGM$^+$90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, oct 1990.

[AH10] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11(10):R106, 2010.

[AW10] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[Ben12] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[Bio] Biological pathways. https://www.genome.gov/27530687/. Accessed June 12, 2017.

[BLP$^+$07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[BM13] Stephen A. Bustin and Jamie Murphy. RNA biomarkers in colorectal cancer. *Methods*, 59(1):116–125, jan 2013.

[BPM04] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1):20–29, 2004.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[CBHK02] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[CFG$^+$09] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The sanger FASTQ file format for sequences with quality scores, and the solexa/illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, dec 2009.

[CH16] Geoffrey M. Cooper and Robert E. Hausman. *The Cell: A Molecular Approach*. Boston University, 7 edition, 2016.

[Cla08] S. Clancy. RNA functions. *Nature Education*, 1(1):102, 2008.

[CLS$^+$11] Geng Chen, Ruiyuan Li, Leming Shi, Junyi Qi, Pengzhan Hu, Jian Luo, Mingyao Liu, and Tieliu Shi. Revealing the missing expressed genes beyond the human reference genome by RNA-seq. *BMC Genomics*, 12(1), dec 2011.

[CMT$^+$16] Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Michał Wojciech Szcześniak, Daniel J. Gaffney, Laura L. Elo, Xuegong Zhang, and Ali Mortazavi. A survey of best practices for RNA-seq data analysis. *Genome Biology*, 17(1), jan 2016.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[DAA$^+$11] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, and R. Durbin and. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, jun 2011.

[DCM] C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*. IEEE.

[DCM$^+$12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DGH16] PADIDEH DANAEE, REZA GHAEINI, and DAVID A. HENDRIX. A DEEP LEARNING APPROACH FOR CANCER DETECTION AND RELEVANT GENE IDENTIFICATION. In *Biocomputing 2017*. WORLD SCIENTIFIC, nov 2016.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[DNA] What is DNA? https://ghr.nlm.nih.gov/primer/basics/dna. Accessed January 13, 2017.

[Duf05] M. J. Duffy. Predictive markers in breast and other cancers: A review. *Clinical Chemistry*, 51(3):494–503, mar 2005.

[Dže03] Sašo Džeroski. Multi-relational data mining: an introduction. *ACM SIGKDD Explorations Newsletter*, 5(1):1–16, 2003.

[EB$^+$11] D. Earl, K. Bradnam, et al. Assemblathon 1: A competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224–2241, sep 2011.

[EMK$^+$11] Henrik Edgren, Astrid Murumagi, Sara Kangaspeska, Daniel Nicorici, Vesa Hongisto, Kristine Kleivi, Inga H Rye, Sandra Nyberg, Maija Wolf, Anne-Lise Borresen-Dale, and Olli Kallioniemi. Identification of fusion genes in breast cancer by paired-end RNA-sequencing. *Genome Biology*, 12(1):R6, 2011.

[FAS] FASTA format. http://zhanglab.ccmb.med.umich.edu/FASTA/. Accessed January 13, 2017.

[FCC11] Nuno A Fonseca, Vitor Santos Costa, and Rui Camacho. Conceptual clustering of multi-relational data. In *International Conference on Inductive Logic Programming*, pages 145–159. Springer, 2011.

[FPMB14] Nuno A. Fonseca, Robert Petryszak, John Marioni, and Alvis Brazma. irap - an integrated rna-seq analysis pipeline. *bioRxiv*, 2014.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GEN] How big is the human genome? https://medium.com/precision-medicine/how-big-is-the-human-genome-e90caa3409b0. Accessed January 17, 2017.

[GO07] The gene ontology project in 2008. *Nucleic Acids Research*, 36(Database):D440–D444, dec 2007.

[Hay09] Erika Check Hayden. Genome sequencing: the third generation. *Nature*, 457(7231):768–769, feb 2009.

[Hay14] Erika Check Hayden. Technology: The $1, 000 genome. *Nature*, 507(7492):294–295, mar 2014.

[HBGL08] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE, 2008.

[HBL$^+$07] Fu Jie Huang, Y-Lan Boureau, Yann LeCun, et al. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[HC16] James M. Heather and Benjamin Chain. The sequence of sequencers: The history of sequencing dna, genomics. *Genomics*, 107(1):1–8, jan 2016.

[HG09] Haibo He and E.A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, sep 2009.

[Hin02] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, aug 2002.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2011.

[HSK$^+$12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[HSL08a] Da Wei Huang, Brad T. Sherman, and Richard A. Lempicki. Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists. *Nucleic Acids Research*, 37(1):1–13, nov 2008.

[HSL08b] Da Wei Huang, Brad T Sherman, and Richard A Lempicki. Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources. *Nature Protocols*, 4(1):44–57, dec 2008.

[JTL$^+$15] Jaroslaw Jendrzejewski, Andrew Thomas, Sandya Liyanarachchi, Andrew Eiterman, Jerneja Tomsic, Huiling He, Hanna S. Radomska, Wei Li, Rebecca Nagy, Krzysztof Sworczak, and Albert de la Chapelle. PTCSC3 is involved in papillary thyroid carcinoma development by ModulatingS100a4gene expression. *The Journal of Clinical Endocrinology & Metabolism*, 100(10):E1370–E1377, oct 2015.

[K$^+$95] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[Kan12] Mehmed Kantardzic. *Data Mining: Concepts, Models, Methods, and Algorithms*. Wiley-IEEE Press, 2012.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kis15] Takahiro Kishikawa. Circulating RNAs as new biomarkers for detecting pancreatic cancer. *World Journal of Gastroenterology*, 21(28):8527, 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[LBOM98a] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. In *Lecture Notes in Computer Science*, pages 9–50. Springer Berlin Heidelberg, 1998.

[LBOM98b] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[Le13] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[LFG+13] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, nov 2013.

[LHW+09] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin and. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, jun 2009.

[LNA17] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[LS12] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature Methods*, 9(4):357–359, mar 2012.

[Mar08] Elaine R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24(3):133–141, mar 2008.

[Met09] Michael L. Metzker. Sequencing technologies — the next generation. *Nature Reviews Genetics*, 11(1):31–46, dec 2009.

[MWM+08] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by RNA-seq. *Nature Methods*, 5(7):621–628, may 2008.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Doklady an SSSR*, volume 269, pages 543–547, 1983.

[NHH00] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, sep 2000.

[O+93] World Health Organization et al. Biomarkers and risk assessment: concepts and principles. 1993.

[OJD04] Julian D Olden, Michael K Joy, and Russell G Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(3):389–397, 2004.

[Pfa13] Michael W. Pfaffl. Transcriptional biomarkers. *Methods*, 59(1):1–2, jan 2013.

[PSL+08] Qun Pan, Ofer Shai, Leo J Lee, Brendan J Frey, and Benjamin J Blencowe. Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nature Genetics*, 40(12):1413–1415, nov 2008.

[PSM15] Kathryn L. Pellegrini, Martin G. Sanda, and Carlos S. Moreno. RNA biomarkers to facilitate the identification of aggressive prostate cancer. *Molecular Aspects of Medicine*, 45:37–46, nov 2015.

[PTS+10] D. Pflueger, S. Terry, A. Sboner, L. Habegger, R. Esgueva, P.-C. Lin, M. A. Svensson, N. Kitabayashi, B. J. Moss, T. Y. MacDonald, X. Cao, T. Barrette, A. K. Tewari, M. S. Chee, A. M. Chinnaiyan, D. S. Rickman, F. Demichelis, M. B. Gerstein, and M. A. Rubin. Discovery of non-ETS gene fusions in human prostate cancer using next-generation RNA sequencing. *Genome Research*, 21(1):56–67, oct 2010.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Qia99] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

[RF04] K. E. Rhodes and J. W. Fawcett. Chondroitin sulphate proteoglycans: preventing plasticity or protecting the CNS? *Journal of Anatomy*, 204(1):33–48, jan 2004.

[RMS09] M. D. Robinson, D. J. McCarthy, and G. K. Smyth. edgeR: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, nov 2009.

[RS00] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[RS08] A. Ralston and K. Shaw. Gene expression regulates cell differentiation. *Nature Education*, 1(1):127, 2008.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[SH07] Ilya Sutskever and Geoffrey E Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *AISTATS*, volume 2, pages 548–555, 2007.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, jul 2009.

[SLS$^+$10] Eric E. Schadt, Michael D. Linderman, Jon Sorenson, Lawrence Lee, and Garry P. Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, sep 2010.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[SSM98] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, jul 1998.

[STM$^+$05] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, and J. P. Mesirov. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, sep 2005.

[TJWJ11] Natalie A. Twine, Karolina Janitz, Marc R. Wilkins, and Michal Janitz. Whole transcriptome sequencing reveals gene expression and splicing differences in brain regions affected by alzheimer's disease. *PLoS ONE*, 6(1):e16266, jan 2011.

[TPS09] C. Trapnell, L. Pachter, and S. L. Salzberg. TopHat: discovering splice junctions with RNA-seq. *Bioinformatics*, 25(9):1105–1111, mar 2009.

[TRG+12] Cole Trapnell, Adam Roberts, Loyal Goff, Geo Pertea, Daehwan Kim, David R Kelley, Harold Pimentel, Steven L Salzberg, John L Rinn, and Lior Pachter. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and cufflinks. *Nature Protocols*, 7(3):562–578, mar 2012.

[TUCG15] Jie Tan, Matthew Ung, Chao Cheng, and Casey S Greene. Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, volume 20, page 132. NIH Public Access, 2015.

[USC] BED format. http://genome.ucsc.edu/FAQ/FAQformat.html. Accessed January 29, 2017.

[Uti05] Robert D. Utiger. The multiplicity of thyroid nodules and carcinomas. *New England Journal of Medicine*, 352(23):2376–2378, jun 2005.

[vdWCV11] Stefan van der Walt, S Chris Colbert, and Gaël Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, mar 2011.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[WGS09] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, jan 2009.

[WL09] Brian T. Wilhelm and Josette-Renée Landry. RNA-seq—quantitative measurement of expression through massively parallel RNA-sequencing. *Methods*, 48(3):249–257, jul 2009.

[WM06] Yanusz Wegrowski and François-Xavier Maquart. Chondroitin sulfate proteoglycans in tumor progression. In *Chondroitin Sulfate: Structure, Role and Pharmacological Activity*, pages 297–321. Elsevier, 2006.

[ZYH+16] Kejun Zhang, Meiqin Yu, Fengyun Hao, Anbing Dong, and Dong Chen. Knockdown of s100a4 blocks growth and metastasis of anaplastic thyroid cancer cells in vitro and in vivo. *Cancer Biomarkers*, 17(3):281–291, Sep 2016.

REFERENCES

Appendix A

DL4J model configurations

A.1 Configuration file (POM)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.vitorteixeira</groupId>
  <artifactId>GenomicDL</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>GenomicDL</name>
  <url>http://maven.apache.org</url>

  <properties>
        <jcommander.version>1.27</jcommander.version>
        <nd4j.backend>nd4j-native-platform</nd4j.backend>
        <nd4j.cuda.backend>nd4j-cuda-8.0</nd4j.cuda.backend>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <shadedClassifier>bin</shadedClassifier>
        <java.version>1.8</java.version>
        <nd4j.version>0.8.0</nd4j.version>
        <dl4j.version>0.8.0</dl4j.version>
        <guava.version>19.0</guava.version>
        <arbiter.version>0.8.0</arbiter.version>
        <datavec.version>0.8.0</datavec.version>
        <logback.version>1.1.7</logback.version>
        <jfreechart.version>1.0.13</jfreechart.version>
        <jcommon.version>1.0.23</jcommon.version>
        <maven-shade-plugin.version>2.4.3</maven-shade-plugin.version>
        <exec-maven-plugin.version>1.4.0</exec-maven-plugin.version>
        <maven.minimum.version>3.3.1</maven.minimum.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.nd4j</groupId>
            <artifactId>${nd4j.backend}</artifactId>
            <version>${nd4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-core</artifactId>
            <version>${dl4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-nlp</artifactId>
            <version>${dl4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>deeplearning4j-ui_2.10</artifactId>
            <version>${dl4j.version}</version>
        </dependency>
        <dependency>
            <groupId>org.deeplearning4j</groupId>
            <artifactId>arbiter-deeplearning4j</artifactId>
            <version>${arbiter.version}</version>
        </dependency>
        <dependency>
            <groupId>com.beust</groupId>
            <artifactId>jcommander</artifactId>
            <version>${jcommander.version}</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>${guava.version}</version>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
            <version>5.1.0.Final</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>${exec-maven-plugin.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>exec</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <executable>java</executable>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>${maven-shade-plugin.version}</version>
                <configuration>
                    <shadedArtifactAttached>true</shadedArtifactAttached>
                    <shadedClassifierName>${shadedClassifier}</shadedClassifierName>
                    <createDependencyReducedPom>true</createDependencyReducedPom>
                    <filters>
                        <filter>
                            <artifact>*:*</artifact>
                            <excludes>
                                <exclude>org/datanucleus/**</exclude>
                                <exclude>META-INF/*.SF</exclude>
                                <exclude>META-INF/*.DSA</exclude>
                                <exclude>META-INF/*.RSA</exclude>
                            </excludes>
                        </filter>
                    </filters>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
                                    <resource>reference.conf</resource>
                                </transformer>
                                <transformer implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
                                <transformer implementation="org.apache.maven.
                                    plugins.shade.resource.
                                    ManifestResourceTransformer">
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

A.2 Available options for network configuration

```

 Options:
    -UI                  Use the UI
                         Default: false
    -activation          Activation function: 1 - Sigmoid | 2 - TANH | 3 -
                         LeakyReLU
                         Default: 2
    -backprop            Backpropagate errors
                         Default: true
    -batchSize           Mini-batch size
                         Default: 50
    -crossV              Cross validate (iterate through files)
                         Default: false
    -dropout             Dropout
                         Default: 0.9
    -earlyStopping       Whether to use early stopping training or not
                         Default: false
    -expNum              Number of the experience
                         Default: 2
    -featureRed          0 - Raw | 1 - PCA | 2 - KPCA+RBF
                         Default: 1
    -fileNum             Number of kFold file
```

```
                      Default: 0
    -h1               Number of hidden nodes layer 1
                      Default: 15000
    -h2               Number of hidden nodes layer 2
                      Default: 10000
    -h3               Number of hidden nodes layer 3
                      Default: 2000
    -h4               Number of hidden nodes layer 4
                      Default: 50
    -labelindex       Label index for classification
                      Default: 57490
    -learningRate     Learning rate
                      Default: 0.1
    -normalize        Normalize input
                      Default: true
    -numEpochs        Number of epochs for training
                      Default: 500
    -oversampleType   0 - None | 1 - SMOTE+TOMEK | 2 - SMOTE+ENN | 3 - ADASYN
                      Default: 1
    -pretrain         Pretrain network
                      Default: true
    -reg              Regularization
                      Default: true
    -saveWeights      Save weights
                      Default: true
    -updater          Updater: 1 - ADAM | 2 - NESTEROVS | 3 - ADAGRAD
                      Default: 1
```

Listing A.1: Available options for neural network configuration

A.3 Shallow Artificial Neural network

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
      .seed(seed)
      .iterations(iterations)
      .optimizationAlgo(optimizationAlgorithm)
      .updater(updater)
      .activation(activationFunction)
      .learningRate(learningRate)
      .weightInit(weightInit)
      .regularization(true).dropOut(dropout)
      .list()
      .layer(0, new OutputLayer.Builder(lossFunction)
            .activation(Activation.SOFTMAX).nIn(numAttributes).nOut(numClasses).
                build())
      .pretrain(false).backprop(true)
      .build();
```

Listing A.2: MultiLayerConfiguration for the shallow artificial neural network architecture

A.4 Denoising Autoencoder

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
    .seed(seed)
    .iterations(iterations)
    .optimizationAlgo(optimizationAlgorithm)
    .updater(updater)
    .learningRate(learningRate)
    .activation(activationFunction)
    .regularization(true).dropOut(dropout)
    .list()
    .layer(0, new AutoEncoder.Builder().nIn(numAttributes).nOut(numNodesHL)
        .weightInit(weightInit)
        .lossFunction(lossFunction)
        .corruptionLevel(corruptionLevel)
        .build())
    .layer(1, new OutputLayer.Builder(lossFunction)
        .activation(Activation.SOFTMAX).nIn(numNodesHL1).nOut(numClasses).build())
    .pretrain(true).backprop(true)
    .build();
```

Listing A.3: MultiLayerConfiguration for the denoising autoencoder architecture

A.5 Stacked Denoising Autoencoder

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
      .seed(seed)
      .iterations(iterations)
      .optimizationAlgo(optimizationAlgorithm)
      .updater(updater)
      .learningRate(learningRate)
      .activation(activationFunction)
      .regularization(true).dropOut(dropout)
      .list()
      .layer(0, new AutoEncoder.Builder().nIn(numAttributes).nOut(numNodesHL1)
            .weightInit(weightInit)
            .lossFunction(lossFunction)
            .corruptionLevel(corruptionLevel)
            .build())
      .layer(1, new AutoEncoder.Builder().nIn(numNodesHL1).nOut(numNodesHL2)
            .weightInit(weightInit)
            .lossFunction(lossFunction)
            .corruptionLevel(corruptionLevel)
            .build())
      .layer(2, new AutoEncoder.Builder().nIn(numNodesHL2).nOut(numNodesHL3)
            .weightInit(weightInit)
            .lossFunction(lossFunction)
            .corruptionLevel(corruptionLevel)
            .build())
      .layer(3, new OutputLayer.Builder(lossFunction)
            .activation(Activation.SOFTMAX).nIn(numNodesHL3).nOut(numClasses).build())
      .pretrain(true).backprop(true)
      .build();
```

Listing A.4: MultiLayerConfiguration for the stacked denoising autoencoder architecture