KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering

Algorithms for analyzing biological sequences

Eduardo de Paula Costa

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

July 2013

Algorithms for analyzing biological sequences

Eduardo DE PAULA COSTA

Supervisory Committee:
Prof. dr. ir. J. Vandewalle, chair
Prof. dr. ir. H. Blockeel, supervisor
Prof. dr. ir. J. Ramon, supervisor
Prof. dr. ir. M. Bruynooghe
Prof. dr. ir. L. De Raedt
Prof. dr. K. Marchal
Prof. dr. ir. P. Geurts
(Université de Liège, Belgium)
Dr. E. Danchin
(Institut National de la Recherche Agronomique, France)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

July 2013

D/2013/7515/84
ISBN 978-94-6018-698-1

Abstract

In 1995, scientists achieved the first complete genetic map of a free living organism, the bacterium *Haemophilus influenzae*. Since then, a huge amount of genomic and proteomic data has been generated due to the developments in biological and computer sciences. This availability of data has led to a new challenge, namely, the analysis and interpretation of this data. These involve many tasks such as: the identification of genes, regulatory regions, and repetitive elements; phylogenetic analysis; identification of proteins and peptides expressed in an organism, cell or tissue; and protein function prediction.

In this thesis, we investigate computational methods for analyzing nucleotide and amino acid sequences. We do it in the context of three biological problems: phylogenetic tree reconstruction, protein subfamily identification, and peptide identification using mass spectrometry data.

In phylogenetic tree reconstruction, one is interested in inferring the most likely tree topology that explains the evolutionary history of genes and organisms. We propose a method that, given a set of nucleotide or amino acid sequences, builds a phylogenetic tree in a top-down way. Our method is based on a conceptual clustering method that extends the well-known decision tree learning approach. We start from a single cluster containing all sequences and repeatedly divide it into subclusters until all sequences form a different cluster. We assume that a split can be described by referring to particular polymorphic positions of the multiple sequence alignment and propose a heuristic to choose the best split at each iteration. This allows us to identify important mutations that might have given rise to different evolutionary lineages. The trees generated by our method are therefore more informative than the ones generated by standard methods. Moreover, we show that our method is comparable to standard ones in terms of accuracy of the produced phylogenetic tree.

In protein subfamily identification, given a set of sequences belonging to a protein family, the goal is to identify subgroups of functionally closely

related proteins. This can be done by using evolutionary information. We propose a method that first uses the phylogenetic tree reconstruction method described in the previous paragraph, and then cuts the tree to extract clusters (subfamilies) of sequences. We show that the proposed method yields good results with advantages over those produced by a state-of-the-art method. More specifically, it identifies subfamily-specific positions that might be important for the function(s) associated to the subfamily, and it allows easy classification of new sequences into one of the identified subfamilies.

Finally, we consider the problem of inferring the peptide identification of mass spectrometry data. In mass spectrometry, an unknown peptide undergoes fragmentation, and its fragment masses are registered in a fragmentation spectrum. Then, computational methods infer the peptide sequence from its spectrum. We propose a method that does this by searching for the peptide identification in the six-frame translation of the genome. Differently from other methods that allow a similar search, it does not use a filtering procedure to limit the computation of the scoring function to a small subset of sequences that are likely to obtain a high score. Instead, it performs an exhaustive scan of the genome translation. We show that our strategy can identify more peptides than a non-exhaustive search.

As an additional contribution of this thesis, we consider a problem purely computational, namely, the problem of estimating the certainty of a prediction given by a decision tree. We propose a method that uses a transductive procedure to tackle this task. We show that our method improves the certainty estimates given by standard decision trees, and is especially suitable for ranking estimation. The ideas we use in our method can be easily generalized to other machine learning techniques, as well as combined with existing methods for prediction certainty estimation.

Beknopte samenvatting

In 1995 slaagden wetenschappers er voor het eerst in om het volledige genoom van een levend organisme, de bacterie *Haemophilus influenzae*, in kaart te brengen. Sindsdien werd een enorme hoeveelheid aan gegevens uit het genoom en het proteoom gegenereerd dankzij de ontwikkelingen in de biologie en computerwetenschappen. Deze beschikbaarheid aan gegevens heeft tot een nieuwe uitdaging geleid, namelijk de analyse en interpretatie van deze gegevens. Dit proces bestaat uit meerdere taken zoals de identificatie van genen, regulerende gebieden en repetitieve elementen, fylogenetische analyse, de identificatie van proteïnen en peptides die in een organisme, cel of weefsel voorkomen, en de voorspelling van genfuncties.

In deze thesis onderzoeken we computationele methoden voor de analyse van nucleotide- en aminozuursequenties. We doen dit in de context van drie biologische problemen: de reconstructie van fylogenetische bomen, de identificatie van proteïnedeelamilies en de identificatie van peptides op basis van massaspectrometriegegevens.

Bij de reconstructie van fylogenetische bomen is men geïnteresseerd in het vinden van de meest plausibele boomtopologie die de evolutionaire geschiedenis van genen en organismen verklaart. We stellen een methode voor die, gegeven een verzameling van nucleotide- of aminozuursequenties, een fylogenetische boom bouwt op een *top-down* manier. Onze methode is gebaseerd op conceptuele clustering die op zich een uitbreiding is op het leren van beslissingsbomen. We beginnen met een cluster die alle sequenties bevat en splitsen deze cluster recursief in deelclusters tot alle sequenties een verschillende cluster vormen. We veronderstellen dat een splitsing beschreven kan worden door de specifieke polymorfe posities van de sequentiealiniëring en stellen een heuristiek voor om de beste splitsing te kiezen tijdens iedere iteratie. Dit zorgt ervoor dat we belangrijke mutaties kunnen identificeren die potentieel geleid hebben naar verschillende evolutionaire takken. De bomen die geleerd werden met onze methode zijn dus informatiever dan de bomen geleerd door standaardmethodes.

Bovendien tonen we aan dat onze methode fylogenetische bomen leert die vergelijkbaar accuraat zijn met bestaande methodes.

Bij de identificatie van deelfamilies van proteïnen is de invoer een verzameling van sequenties die tot een proteïnefamilie behoren en het doel de identificatie van deelgroepen van functioneel gerelateerde proteïnen. Dit is mogelijk door gebruik te maken van evolutionaire informatie. We stellen een methode voor die eerst fylogenetische bomen reconstrueert (aan de hand van de methode beschreven in de vorige paragraaf) en vervolgens de boom knipt om clusters (deelfamilies) van sequenties te bekomen. We tonen aan dat de methode goede resultaten behaalt en ook andere voordelen heeft t.o.v. van een state-of-the-art methode. Meer specifiek identificeert onze methode specifieke posities die belangrijk kunnen zijn voor de functies geassocieerd aan de deelfamilie, en laat ze classificatie toe van nieuwe sequenties m.b.t. één van de geïdentificeerde deelfamilies.

Tenslotte beschouwen we de identificatie van peptides uit massaspectrometriegegevens. In massaspectrometrie ondergaat een ongekende peptide fragmentatie, waarbij de massa's van de verschillende fragmenten weergegeven worden in een fragmentatiespectrum. Vervolgens infereren computationele methodes de peptidesequentie uit dit spectrum. We stellen een methode voor die zoekt naar de peptide-identificatie in de vertaling van het genoom voor de zes leesramen. In tegenstelling tot andere methodes die een gelijkaardige zoekruimte doorzoeken, gebruikt onze methode geen filterprocedure, die als doel heeft om de berekening van de scorefunctie te beperken tot een klein aantal sequenties die een hoge kans hebben op een hoge score. In plaats daarvan doorzoekt ze de vertaling van het genoom exhaustief. We tonen aan dat deze strategie meer peptides kan identificeren dan een niet-exhaustief zoekproces.

Een bijkomende bijdrage van deze thesis is een puur computationeel probleem, namelijk de schatting van de zekerheid van een predictie gegeven door een beslissingsboom. We stellen een methode voor die een transductieve procedure gebruikt om dit probleem aan te pakken. We tonen aan dat onze methode de zekerheidsschattingen van standaard beslissingsbomen verbetert, en in het bijzonder bruikbaar is voor het schatten van rankings. De ideeën die we voor deze methode gebruiken kunnen gemakkelijk veralgemeend worden naar andere leertechnieken; tevens kunnen ze gecombineerd worden met bestaande methodes voor de schatting van de zekerheid van voorspellingen.

Acknowledgments

I am about to finish my Ph.D. journey. A journey that started five years ago when I moved to Belgium, leaving behind my home country (Brazil), my family, friends, and the "world" I knew so far. A journey that has been full of challenges, adventures and excitement, and deep in learning. A journey that is composed of an inseparable mixture of personal and academic experiences. A journey that has changed me in so many different ways that I am sure that my decision to pursue a Ph.D. at the KU Leuven was one of the best decisions I have made in my life.

When I look back at everything that has happened since my first day in Belgium, my heart becomes full of gratitude for all those people that helped me somehow along the way. And I would like to take this opportunity to thank some of them.

First of all, I would like to thank my supervisor Hendrik Blockeel. I feel privileged to have been given the opportunity to work with him. I have learned a lot with him and I am very grateful for every compliment, critical remark and suggestion I received from him during these years. I would also like to thank Jan Ramon and Maurice Bruynooghe, who were involved in the supervision of my Ph.D. as well and who were always ready to help me out.

Apart from Hendrik, Jan and Maurice, I also want to thank the other members of my Ph.D. jury for reading and evaluating my thesis, and for their suggestions to improve the text: Luc De Raedt, Kathleen Marchal, the external members Pierre Geurts and Etienne Danchin, and the chairman Joos Vandewalle.

As I was involved in different research tasks during my Ph.D., I had the opportunity to work with different people and I am very grateful for all their help and great contribution to this thesis. First of all, I would like to give a huge thank you to Celine Vens, who is by far the person with whom I spent most of my working hours. She was involved in most of the research tasks I have worked on, and I am very thankful for her daily guidance during all these

years. I would like to thank Kurt De Grave for all his help and meticulousness in the work involving mass spectrometry data analysis. I also want to thank Gerben Menschaert and Walter Luyten for their time in providing biological data and answering my doubts about mass spectrometry. I also want to thank Sicco Verwer for his ideas and discussions in the work we did together.

I would like to give a special thanks to Leander and Tias, two colleagues who have also become good friends. They were always ready to encourage me during the difficult times, both for Ph.D. and personal matters. They were always there to enjoy and celebrate the good moments with me as well. They also helped me in revising parts of this thesis and in providing feedback about my work. And, of course, I cannot forget our weekly running sessions during lunch break, which were always very welcome.

I want to thank Karin, Denise and Inge, who helped me out several times with administrative procedures involving my Ph.D. I gratefully acknowledge the project "Learning from data originating from evolution", funded by the Research Foundation - Flanders (FWO-Vlaanderen), which provided me with financial support during my Ph.D. studies.

I would like to thank all my "kotgenootjes" (residence mates), who played a fundamental role in my adaptation to Belgian life. They have seen me during relaxed and stressful periods, and were especially supportive in the last phase of my thesis writing. A special thanks to three of them, who have become very good friends: Mieke, Thomas and Sophie.

I would like to thank the incredible people that I have met in the different organizations/communities which I joined at some point during my stay in Belgium: Ichtus, ICEL (International Church of Evangelicals in Leuven), the Brazilian group in Leuven, HRC (Hagelandse Running Club), and Omkadering. A special thanks to my good friends Alejandra, Hanna, Sarah, Suzana and Gabriel, who were always very interested in knowing about the progress of my Ph.D. studies and have even been volunteers for some of my try-out presentations. Their friendship and companionship have also made my days more joyful and meaningful.

A special thanks to my good friend Philip and all his family, who welcomed me in their lives as part of the family. I cannot describe in words how important it was for me to have a family here. I am also grateful to four other families that also welcomed me in their homes on different occasions: Mieke's family, Thomas' family, Sarah's family and Hanna's family. This meant a lot to me.

I am very grateful to my friends in Brazil who were very supportive during this period and helped me in many different ways. A special thanks to Cinthia, César, Ray, Carol, Ivan, Camila, Magda, Gabi, Mariana, Cerri, Olga and Fischer.

A special thanks to my friend Liam, from England, who was always ready for a Skype chat during my breaks while I was writing my thesis. That made the writing phase less lonely, especially in the late evenings. He has also helped me with some English corrections.

Finally, I would like to thank my family in Brazil. Even being so far away, they were very present in every step of my Ph.D. studies and my adaptation/life in Belgium. They were always there for words of encouragement and have always supported and believed in me. And I would like to thank God for sustaining and guiding me in so many different ways that I cannot even grasp or understand.

Eduardo de Paula Costa
Heverlee, June 2013

Contents

List of Figures

List of Tables

Abbreviations

AUC	Area under the ROC curve
ECC	Encoding cost
Da	Dalton
DNA	Deoxyribonucleic acid
Ens	Ensemble
HMM	Hidden Markov model
HTU	Hypothetical taxonomic unit
IG	Information gain
MALDI	Matrix-assisted laser desorption/ionization
MaxAvgDist	Maximization of the average inter-cluster distance
MaxMinDist	Maximization of the minimum inter-cluster distance
MS	Mass spectrometry
MSA	Multiple sequence alignment
MS/MS	Tandem mass spectrometry
MIWS	Multi-ions weighted SEQUEST

NJ	Neighbor Joining
OTU	Operational taxonomic units
PIUS	Peptide Identification by Unbiased Search
PST	Peptide sequence tag
PTDC	Phylogeny by Top-Down Clustering
PTM	Post-translational modification
RA	Relative abundance of ions
RNA	Ribonucleic acid
SHMM	Subfamily hidden Markov model
TBC	Tree-based classification
TOF	Time-of-flight
TPCE	Transductive prediction certainty estimation
UPGMA	Unweighted pair group method with arithmetic mean
VI	Variation of information

Chapter 1

Introduction

The goal of this thesis is to develop machine learning methods for analyzing biological sequences. In this chapter, we give an overview of our work, describing its context, motivations and contributions, and present the structure of the thesis.

1.1 Context

This thesis is situated in the field of machine learning and in the application domain of bioinformatics. We briefly describe these two research areas in this section.

1.1.1 Machine learning

Broadly speaking, learning can be defined as the ability of an organism to adapt its behavior as a result of interaction with the environment. This change of behavior, which can be stimulated by different kinds of interactions, aims at improving the performance in solving one or more tasks. In a pride of lions, for example, the young members of the pride improve their hunting skills by observing the behavior of the elder and more experienced members, and by their own success and failure experiences.

Similarly, in the context of computer science, we can develop programs that are able to automatically improve their own performance on a certain task.

They do it by analyzing recorded data for that task. This data often consists of observations of how instances of the task were performed in the past [13]. The experience given by these observations enables the program to "learn" how to improve its performance. This leads to the following definition of learning behavior:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [94].

This artificial learning behavior is the subject of machine learning, which is one of the main research branches within artificial intelligence. Next, we give an example of a learning task.

Example 1. *Consider the task of predicting if a certain drug will cause side effects in patients under treatment for a particular disease. The performance of a machine learning program tackling this task can be measured by the percentage of patients for which the correct prediction is made. The experience used by the program can be the history information of former patients that have undertaken the same treatment, and how they reacted to the drug. If the program improves its prediction for the new patients by using this experience, we say that the program has the ability to learn.*

The above example illustrates a *predictive learning* task. In this kind of learning, the goal is to make predictions for new cases based on what has been previously observed. What we want to predict is called the "target variable", and the observations are described in terms of "descriptive variables". In Example 1, the target variable indicates whether a patient showed side effects for the drug or not, and the descriptive variables could be the information contained in the patients' profiles, for example. To be able to make predictions, a predictive machine learning builds a model or hypothesis that maps the descriptive variables (input) to the target variable (output). Examples of predictive learning techniques include learners of decision trees [17, 102], artificial neural networks [143], and support vector machines [28].

Machine learning can also be used for descriptive purposes; we call this *descriptive learning*. In this learning setting, the goal is to build a model that captures properties (ideally, properties that are not trivially visible) of what has been observed [13]. Examples of descriptive learning tasks are clustering [66], frequent itemset discovery [2], and association rule mining [1]. K-means [84] and Expectation Maximization (EM) [31] are examples of descriptive learning techniques. Some originally predictive learning techniques have also been applied

to descriptive tasks: top-down induced clustering trees [14] and self-organizing maps (SOM) [74] are descriptive counter-parts of decision trees and artificial neural networks, respectively. An example of a descriptive learning problem follows.

Example 2. *Consider the clustering task where, starting from a set of biological sequences, the goal is to group sequences into clusters of similar sequences. To determine which sequences should be grouped together, the clustering algorithms use the values for the descriptive variables. In this case, these variables could be features of the sequences such as its length, and molecular weight.*

In this thesis we investigate both kinds of learning.

1.1.2 Bioinformatics

The origin of bioinformatics is usually associated to the time of the large-scale genome projects and the development of the first large biological databases. However, the roots of this interdisciplinary research area actually date back to the early 1960s. By then, computers emerged as important tools in molecular biology. This was possible due to three main factors [57]: (1) the development of techniques to sequence proteins; (2) the increasing interest of scientists in the information carried by these sequences; and (3) the availability of computers to academic researchers. The term bioinformatics was used for the first time in the early 1970s to define "the study of informatic processes in biotic systems" [59]. By then, computational biologists had already developed a diverse set of techniques for analyzing molecular structure, function and evolution [57].

Since those early years, bioinformatics has grown into a broad field, with a large research community, and now plays added roles in biological and medical research [95].

In its essence, bioinformatics can be defined as an interdisciplinary research area that studies, develops and applies computational, statistical and mathematical methods to solve biological problems. Among the large range of problems tackled by bioinformatics methods, we can mention: genome sequencing, gene finding, protein and peptide identification, sequence alignment, phylogenetic tree reconstruction, protein function prediction, biological database design and management, just to name a few. Next, we give a quick overview of each one of these tasks.

Genome sequencing consists of determining the whole hereditary information of an organism, which is encoded in DNA (deoxyribonucleic acid); in the case of some viruses, the hereditary information is encoded in RNA (ribonucleic acid).

The complete DNA sequence of an organism contains all necessary information for its development and functioning. More specifically, this information is used to produce proteins, which are responsible for almost all cellular activities necessary for the functioning of an organism. The regions of the genome that are used to produce proteins are called genes.[1] Thus, gene finding is the task of determining those regions.

In protein identification, one is interested in identifying which proteins are present in a certain organism, tissue or cell type. One technology that is commonly used for this is mass spectrometry [54]. This technology is also used for the identification of peptides, which are structurally very similar to proteins, but shorter.

Sequence alignment involves arranging sequences in a way that it identifies regions of similarity. These similar regions may be due to evolutionary reasons, for example. Hence, such alignments are usually used in phylogenetic tree reconstruction [112]. In this task, the goal is to reconstruct the evolutionary history of organisms, species, or genes. Knowing how sequences are evolutionarily related is important for protein function prediction, for example. Proteins that are evolutionarily closer are more likely to be responsible for the same functions in the biological system of an organism.[2]

Finally, all the data resulting from the aforementioned tasks needs to stored in a structured way, so that data can be retrieved or updated, for example. For this reason, biological database design and management [9, 8] is also an important task in bioinformatics.

1.2 Motivations and contributions

In 1995, scientists achieved the first complete genetic map of a free living organism, the bacterium *Haemophilus influenza*. Since then, many genomes have been fully sequenced. These genomic developments were followed by advances in proteomics, which can be defined as the large scale study of the proteins produced by an organism.

The progress in both genomics and proteomics has led to an enormous increase in the amount of publicly available data in biology [100, 105]. The availability of biological data has, in turn, led to new challenges that involve how to make

[1] There are also genes that do not code for proteins. For example, some genes that code for subunits of the ribosome are made of RNA and are not translated into proteins. Some genes also code for small RNAs that are involved in regulation of other genes.

[2] Phylogenetic analysis also has other applications such as the investigation of the origin/history of epidemics and infectious disease contamination.

sense out of all this data. It is in this context that data mining and machine learning have been playing an important role as a source of new methods and techniques to analyze biological data.

The overall goal of this thesis is the development of computational methods for analyzing biological sequences (in particular, DNA and amino acids sequences). We approach this goal in two different ways. First, we explore the evolutionary information encoded in biological sequences to tackle two related tasks: phylogenetic tree reconstruction and protein subfamily identification. Second, we explore the information encoded in the genome to perform peptide identification using mass spectrometry data.

The main contributions of this thesis can be summarized as follows:

The **first contribution** is the investigation of the use of divisive conceptual clustering for phylogenetic tree reconstruction. In this context, we propose a novel method based on a decision tree learner that builds the phylogenetic tree in a top-down way. We show that, even though little investigated and used, top-down clustering can produce results comparable to those of well-known phylogenetic methods, which use a bottom-up approach.

The **second contribution** is a new method for protein subfamily identification. In this problem, given a set of sequences belonging to a protein family, the goal is to identify subgroups of functionally closely related proteins. Our proposed method does it by analyzing the evolutionary relationship among the protein sequences. It first uses our aforementioned method for phylogenetic tree reconstruction. Then, it cuts the resulting tree to extract clusters (subfamilies) of sequences. We show that the proposed method yields good results with advantages over those produced by a state-of-the-art method.

The **third contribution** is a new method for peptide identification. The proposed method infers the peptide identification from tandem mass spectrograms by searching against the six-frame translation of the genome. Differently from other methods that allow a similar search, it does not limit the analysis to a small subset of candidate sequences. Instead, it performs an exhaustive scan of the translation of the six reading frames of the complete genome. We show that our strategy can identify more peptides than a non-exhaustive search.

As an **additional contribution** of this thesis, we introduce a new procedure to calculate prediction certainty for decision trees. This research topic was motivated by the work in phylogenetic tree reconstruction and protein subfamily identification, which uses decision tree learning. We show that our method improves the certainty estimates given by standard decision trees, and is especially suitable for ranking estimation.

1.3 Structure of the thesis

The remainder of this text is structured as follows. **Chapter 2** presents the background and basic concepts needed for the other chapters. The next four chapters present the methods we propose for the tasks investigated in this thesis. For each one of them, we indicate the key paper on which the chapter is based.

- **Chapter 3** presents our conceptual clustering method for phylogenetic tree reconstruction.
  - Celine Vens, Eduardo P. Costa, and Hendrik Blockeel, *Top-down induction of phylogenetic trees.* European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics. Lecture Notes in Computer Science, volume 6023, pages 62-73, 2010.
- **Chapter 4** presents our method for protein subfamily identification.
  - Eduardo P. Costa, Celine Vens, and Hendrik Blockeel, *Top-down clustering for protein subfamily identification.* Evolutionary Bioinformatics, volume 9, pages 185-202, 2013.
- **Chapter 5** presents our method for peptide identification using mass spectrometry data.
  - Eduardo P. Costa, Gerben Menschaert, Walter Luyten, Kurt De Grave, and Jan Ramon, *PIUS: Peptide Identification by Unbiased Search*, Bioinformatics - application notes, doi: 10.1093/bioinformatics/btt298, 2013.
- **Chapter 6** presents our alternative method for prediction certainty estimation in decision trees.
  - Eduardo P. Costa, Sicco Verwer, and Hendrik Blockeel, *Estimating prediction certainty in decision trees.* Submitted to the 12th International Symposium on Intelligent Data Analysis. Submitted in May 2013.

Finally, **Chapter 7** summarizes our work, and presents the main conclusions and final considerations.

Chapter 2

Background

This chapter introduces basic concepts that will be used in this thesis, both from the biological and computational points of view. As such, we organize the chapter in two parts. In the first part, we introduce concepts from molecular biology, which together give a clearer context to the bioinformatics problems we consider. In the second part, we introduce some machine learning concepts related to the methods we present in the remaining of the thesis.

2.1 Biological sequences

As the goal of this thesis is the computational analysis of biological sequences, we focus on the biological aspects related to nucleotide and amino acid sequences.

2.1.1 A cell and its genetic information

All living organisms are constituted by one or more *cells*. A cell is the smallest structural unit of an organism that is capable of independent functioning. For this reason, cells are often mentioned as the "building blocks" of life. Cells are involved in all vital functions of an organism, such as absorption of nutrients and water, reproduction and growth. Cells also store the instructions necessary for the development and functioning of the organism, which are referred to as the *genetic information* of the organism. This information is passed from one cell generation to the next when cells make copies of themselves. Moreover, the genetic information of an organism is also transferred to its offspring.

The genetic information is found in a cell in the form of *DNA* (*deoxyribonucleic acid*).[1] Most DNA molecules are formed by two strands of *nucleotides* that are held together in a double helix (Figure 2.1). Each nucleotide is formed by a chemical base (purine or pyrimidine base) attached to a sugar molecule and a phosphate molecule; the two purine bases in the DNA are guanine (G) and adenine (A), and the two pyrimidine bases are thymine (T) and cytosine (C). In the double helix structure, DNA bases pair up with each other (A with T, and C with G).

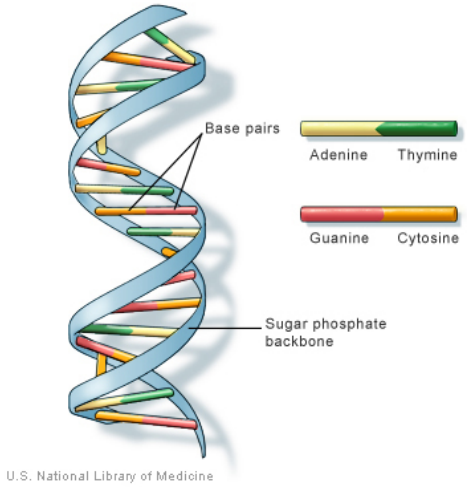


Figure 2.1: DNA structure [126].

DNA strands have a directionality, which is determined by how the the sugar molecules of the nucleotides are joined to one another. The ends of a DNA strand are called the *5' (five prime)* and *3' (three prime)* ends. Conventionally, the 5' and 3' ends are called the *upstream* and *downstream* ends, respectively. The strands in the DNA double helix have opposite directions, as illustrated in Figure 2.2. The direction of a strand is important for the chemical reactions involving DNA. For example, DNA replication is carried out in the 5' to 3' direction.

In the cell, the DNA molecule is packaged into structures called *chromosomes*. Human cells, for example, have either (a) 23 chromosomes, in the case of *germ cells*, which are the cells used for reproduction of an organism, or (b) 23 pairs

[1]Some viruses contain *RNA* (*ribonucleic acid*) instead of DNA. However, viruses do not have an organized cell structure. As a result, they need to enter a cell of a living organism to be able to replicate. For this reason, viruses are not considered living organisms.

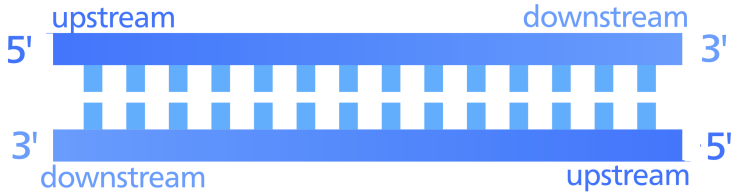


Figure 2.2: Illustration of the directionality of a double-stranded piece of DNA. Adapted from [139].

of chromosomes (giving a total of 46 chromosomes), in the case of the *somatic cells*, which are all the other cells of an organism. The number of nucleotides in the human chromosomes range from 46 to 247 million base pairs of nucleotides, and the whole human genome is around 3 billion base pairs long. The DNA information encoded in all chromosomes of an organism corresponds to its *genome*.

2.1.2 From DNA to proteins

The genome encodes the information necessary to produce *proteins*, which are macromolecules consisting of one of more chains of units called *amino acids*. There are 20 amino acids that can be encoded by the genome of all organisms. They are called *standard amino acids*. However, not all of them can be synthesized by all organisms. The human body, for example, can only synthesize 11 of the 20 standard amino acids. When an amino acid cannot be synthesized by an organism, it is called an *essential amino acid* of that organism, and it needs to be obtained from other organisms via feeding for example. Leucine and lysine are examples of essential amino acids in humans.

The common structure of amino acids (Figure 2.3) consists of a central carbon atom to which an amino group (-NH2), a carboxyl group (-COOH), a hydrogen atom, and a side chain (or R-group) are attached. The side chain is specific to each amino acid.

Distinct proteins have not only distinct amino acid sequences, but also distinct three-dimensional shapes, which determine their activity. Proteins perform many important functions within living organisms. For example, keratin, collagen, and elastin are structural proteins that provide support to tissues; and enzymes are proteins responsible to speed up chemical reactions. The study of all proteins expressed at a certain time in a certain organism, tissue, or cell type is the subject of *proteomics*; this entire set of proteins is called *proteome*.

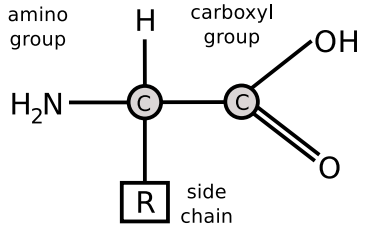


Figure 2.3: General structure of an amino acid.

Proteins are produced by specific fragments of the genome called *genes*; a gene usually has the instructions to produce a certain protein, but it can also regulate the operation of other genes. The instructions to produce the protein sequence are encoded in the nucleotide sequence of the gene. Each triplet of nucleotides, called a *codon*, codifies for one amino acid. Figure 2.4 depicts an example of how DNA is encoded into an amino acid chain.[2] There are two main steps in this encoding process. First, the information in the DNA strand is copied into RNA; this is known as *transcription*. Second, the information stored in RNA is used to produce the amino acid chain; this is known as *translation*. It is between transcription and translation that introns are removed from the RNA strand. In the example shown in Figure 2.4 we do not consider introns.

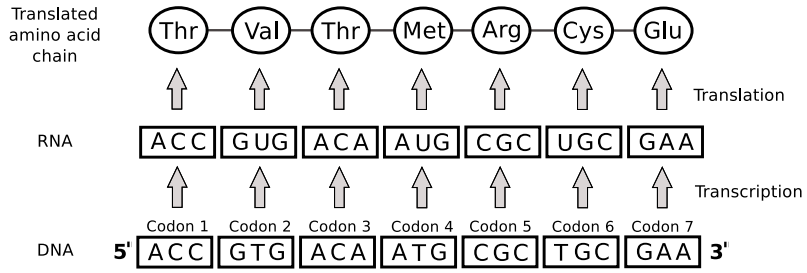


Figure 2.4: Example of a amino acid chain encoded from a DNA fragment. The notation 5' and 3' indicate the direction in which the fragment is translated. For the sake of simplicity, we omit the complementary strand of the DNA fragment being translated.

During this process, not the complete nucleotide sequence of the gene will be translated into amino acids. Some parts of the gene are responsible to regulate the translation procedure (e.g., to indicate the nucleotide position

[2] As a codon is formed by three nucleotides, and there are four different kinds of nucleotides in the DNA, there are 64 distinct codons. Some of them codify for the same amino acid; they are called *synonymous codons*. Note that, in the example depicted in Figure 2.4, two different codons codify for the amino acid Threonine (Thr).

where the translation will start). Other parts called *introns* are located inside the fragment that will be translated into amino acids, but they are removed from the translation process and do not end up producing amino acids. The fragments of the gene that do produce amino acids are called *exons*. The sum of the exons of a gene is called *coding region* or *coding sequence*.

As amino acids are coded by triplets of nucleotides, the position at which the translation begins determines which triplets/codons will be used. In fact, the same DNA sequence can produce three different sequences of nucleotide triples. In this case we talk about different *reading frames*. If we consider the two strands of DNA, which are oriented in opposite directions, there are six possible ways in which the same DNA fragment can be translated: three for each strand.

After translation, chemical modifications can occur in the amino acids of a protein; they are called *post-translational modifications (PTMs)*. PTMs create modified versions of the standard amino acids, increasing the range of proteins that can be produced by the genes. These modifications thus contribute for function diversity in the proteome. They can occur *in vivo* or *in vitro* (e.g., during sample preparation in proteomics studies).

2.1.3 Peptides

When a chain of amino acids has a short length (conventionally, fewer than 50 amino acids), it is called a *peptide*. Peptides are thus structurally similar to proteins but shorter. They are known to have a key role in many physiological processes, such as blood pressure regulation, feeding behavior, water balance, analgesia, and glucose metabolism. Endogenous peptides (i.e., those originated from within an organism) are typically generated from a precursor protein through cleavage by various types of processing enzymes [36, 47]. The study of all endogenous peptides expressed at a certain time in a certain organism, tissue, or cell type is the subject of *peptidomics*.

2.1.4 Evolutionary mechanisms and phylogenetic analysis

New cells are produced by a process called *cell division*. In this process, a *parent cell* divides itself into two (or more) *daughter cells*. Each resulting cell will carry the genetic information of its parent cell, or half of this information, in the case of germ cells.[3] To that aim, DNA (or RNA, in the case of some

[3]The cell division that produces germ cells is called *meiosis*. The division that produces somatic cells is called *mitosis*.

viruses) in the parent cell first needs to be duplicated to then be transmitted to the resulting cells.

In this duplication process, errors can occur. To prevent these errors to be transmitted to daughter cells, the cell has repair mechanisms that identify and correct damage to the DNA molecules. However, errors can escape these repair mechanisms, giving rise to *point mutations* in the genome. A point mutation can correspond to the replacement of one nucleotide by another one, or the insertion or deletion of a nucleotide. Another form of mutation is that originated by the recombination of different strands of DNA. These mutations can have major effects on the associated function of a gene. Mutations can also be caused by *transposable elements*. These elements are fragments of DNA that have the ability of changing their location in the chromosome or generating copies of themselves that will be inserted in other locations of the chromosome. In the latter case, there is an enlargement of the genome.

When mutations occur in a germ cell, they can be transmitted to the descendants of the organism. The accumulation of such mutations can cause the division of a single species into two or more distinct ones. This evolutionary process is called *speciation* and is usually a result of geographical separation of individuals of the same species. Mutations in the somatic cells are only transmitted to the daughter cells of the mutated ones.

The investigation of the evolutionary history of genes and organisms is called *phylogenetic analysis*. In this context, evolutionary relationships are illustrated by a tree-shaped structure called a *phylogenetic tree* [112]. The topology of a phylogenetic tree shows which genes (or organisms) are more evolutionarily closely related. Moreover, branch lengths can be used to add more information to the tree. More specifically, branch lengths in phylogenetic trees indicate either evolutionary time or the number/proportion of changes (mutations) that have occurred in the branch. In the phylogenetic trees considered in this thesis, the branch lengths are calculated based on mutations.

A phylogenetic tree can be rooted (Figure 2.5.a) or unrooted (Figure 2.5.b). In a rooted tree, the direction of evolution goes from the root down to the leaf nodes, while in the unrooted tree the direction of evolution is not known.

Phylogenetic methods usually start from a *multiple sequence alignment* (MSA) to build phylogenetic trees, as we discuss in more detail in Chapter 3. In an MSA, each sequence corresponds to one organism (alternatively, to one gene or protein), and similar sequence fragments (conserved regions) are aligned under one another. In this process, gaps can be inserted when positions of one or more sequences cannot be aligned well with the other sequences. Figure 2.6 shows an example of an MSA.

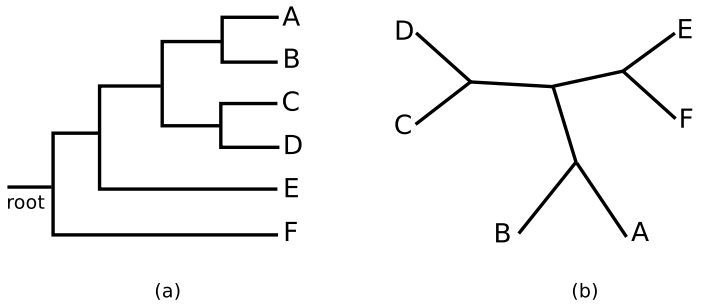


Figure 2.5: Example of (a) a rooted and (b) an unrooted phylogenetic tree.

A:	A	T	T	C	G	A	T	C	G	A	A	C	C	T	C	-	-	-	A	A	T	A	T	G
B:	A	T	T	C	G	A	T	C	G	A	A	C	C	T	A	-	-	-	A	C	-	T	T	G
C:	C	T	T	C	G	A	T	C	G	A	C	C	C	T	A	T	A	C	-	A	G	T	T	G
D:	C	T	T	C	G	A	T	C	G	A	A	C	C	T	A	-	A	C	A	C	A	T	T	G
E:	C	T	T	C	G	A	T	C	G	A	A	C	C	T	A	T	A	-	A	T	A	A	T	G
F:	A	T	T	C	G	A	T	C	G	A	A	C	C	-	-	-	-	-	A	T	A	A	T	G

Figure 2.6: Example of a multiple sequence alignment.

2.1.5 Genome annotation

Phylogenetic analysis is one of the many strategies used by biologists for *genome annotation*. The annotation of an organism's genome aims at identifying key features of the genome [121]. It comprises the identification of genes, regulatory regions, and repetitive elements such as transposable elements; identification of proteins and peptides expressed in the organism; protein function prediction; etc.

In protein function prediction, for example, biologists commonly start by dividing the proteins being studied in smaller groups of related proteins (protein families/subfamilies). This classification is intended to group together sequences which are potentially functionally related. One approach to form such groups is using phylogenetic information, as we discuss in Chapter 4.

Mass spectrometry (MS) is another technique commonly used in genome annotation. This technique measures the mass of molecules (and molecule fragments) as a means of identifying which molecules are expressed in a certain cell, tissue or organism. MS is commonly used for protein/peptide identification, as we discuss in Chapter 5.

By identifying key features of the genome of an organism, genome annotation helps biologists in the process of bridging the gap between the sequence information and the biology of the organism [121].

2.2 Machine learning

In Chapter 1 we defined artificial learning as the ability of a computer program to use experience about a certain task to improve its own performance on that task, given a performance measure. In this section, we explain the concepts related to this definition in more detail. In particular, we focus on the concepts that will be used in this thesis.

2.2.1 Learning from experience

In a learning task, the *experience* related to the execution of a certain task is the extra information that the computer has with respect to that task. This experience can be observations of how the task was executed in the past and what the outcome of each execution was. This experience will prove to be useful if it gives the program enough information to improve its performance in solving the task.

To make a parallel with a daily situation where experience is also used, consider the case where a patient goes to a doctor in search for a treatment for a certain disease. In order to prescribe the right treatment, the doctor uses his experience (or the experience of his peers) about how previous patients with a similar medical history reacted to that treatment, for example.

Note that this example is very similar to Example 1 (see Chapter 1), in terms of the experience used to solve the problem. In that example, we described the task where a computer program needs to predict if a certain drug will cause side effects in patients under treatment for a certain disease. However, the way this information is represented in both cases might be considerably different. While a doctor does not necessarily need this information to be organized in a structured way, a computer does. The most common way to represent the input knowledge used by a computer is to use the *attribute-value* format. This representation can be seen as a table where every row corresponds to one observation (or one instance) and every column corresponds to some kind of description of that observation.

Example 3. *A multiple sequence alignment can be seen as an example of attribute-value data. Each sequence of the alignment corresponds to one instance,*

and each column corresponds to a descriptive feature of that sequence, namely, which nucleotide or amino acid is present at that specific sequence position.

2.2.2 Types of learning

Regarding the goal of the learning process, we can have two main kinds of learning: *predictive learning* and *descriptive learning*. In the former, we are interested in making predictions for new instances based on previously observed ones. In the latter, on the other hand, we are interested in building a model that captures properties of what has been observed.[4]

Next, we describe how decision trees can be used for predictive learning. Then, we discuss top-down induced clustering trees, which are decision trees learned in the context of descriptive learning. We finish the section with a note on the decision tree learner Clus, which can be used to learn predictive and clustering trees. We focus on decision trees because this is the machine learning technique that will be considered throughout this thesis.

Decision trees for predictive learning

A *decision tree* is a tree-shaped predictive model that maps instances from an input domain X to some output domain Y. In this model, each internal node contains a Boolean test (e.g., $<$, $>$, $==$, $\in$). These tests compare a constant value with an attribute value[5] defined for elements from X, or a set of values in the case of the operator $\in$. Each leaf node, on the other hand, is associated with a value $y \in Y$. The mapping of an instance $x \in X$ to a value y is obtained by sorting it down the tree from the root to some leaf node, consistently with the tests in the internal nodes (e.g., 'yes' go left, 'no' go right). In the *classification* context, decision trees are used to map an instance x to a nominal value y, usually called a class; we denote the set of classes $C = \{c_1, ..., c_k\}$. Figure 6.1 shows an example of a classification decision tree. When x is mapped to a numerical value y instead, we talk about *regression*.

Most decision tree learners use a top-down approach to grow the tree. This is known as *top-down induction of decision trees* [103]; see pseudocode in Figure 2.8. In this approach, the initial tree contains only one leaf node (i.e., the root of the tree) containing all data. Then the data is iteratively partitioned by selecting a leaf node and replacing it by a new internal node with two new leaves as children (which is called *splitting*). Which node to replace and what test to

[4] For a more detailed explanation of predictive and descriptive learning, see Section 1.1.1.
[5] Decision trees can handle both continuous and discrete valued attributes.

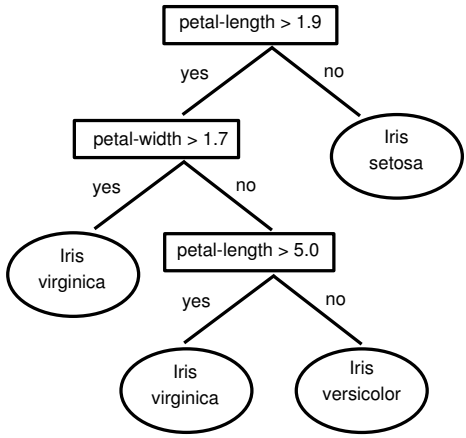


Figure 2.7: Example of a decision tree. This tree was induced for the Iris dataset [7].

introduce is decided using criteria from information theory such as *information gain*. This procedure continues until a stop criterion is reached (e.g., when the tree has reached a pre-defined maximum depth or when the tree is completely grown). Once the tree model has been constructed, *pruning* may be applied to eliminate some sections of the tree in order to reduce the risk of overfitting.

```
GrowTree

    // Input: Data to be partitioned D
    // Output: Decision tree built on D

    if stopping criterion has not been reached then
        For each possible test t do
            Partition the data according to t
            Calculate the heuristic for the resulting partition
        Choose the test with the best heuristic value
        Partition the data according to this test
        Run GrowTree on each resulting subset
```

Figure 2.8: Pseudocode for the top-down induction of a decision tree. Note that GrowTree is called recursively.

Decision tree learning uses a *symbolic machine learning paradigm*. The main

characteristic of this kind of learning is that the learned model is represented in a symbolic language which can be easily understood and interpreted in natural language. The decision tree in Figure 2.7, for example, expresses classification rules for the dataset Iris in a way that can be easily understood by a non-machine learning expert.

Information gain

As we come back to the definition of information gain in Chapters 3 and 6, we briefly discuss its definition next. This explanation is based on the one provided by Blockeel [13].

In information theory, there is the notion of "missing information" or uncertainty about a variable. This notion is quantified by a measure called entropy. A high entropy for a variable means that we have very little certainty (or knowledge) about which value that variable might assume. Given a set of instances S, the entropy of a variable V is defined as

$$H_V(S) = -\sum_i p(v_i|S) \log_2 p(v_i|S),$$

where i ranges over every value v_i that V can assume, and $p(v_i|S)$ is the probability of a random instance from S has value v_i. In the classification context, we calculate the entropy for the target variable C, by ranging over all possible classes c_i:

$$H_C(S) = -\sum_i p(c_i|S) \log_2 p(c_i|S).$$

If we split S into subsets S_l, we can obtain the entropy of the resulting partition $\boldsymbol{S}$ by calculating the weighted average of the entropies of each subset:

$$H_C(\boldsymbol{S}) = -\sum_l \frac{|S_l|}{|S|} \sum_i p(c_i|S_l) \log_2 p(c_i|S_l),$$

where $|S|$ is the cardinality of S and $p(c_i|S_l)$ is the probability that a random element from S_l has class c_i.

Finally, we define the information gain resulting from a split as the difference between the entropies before and after the split:

$$IG(S,\boldsymbol{S}) = H_C(S) - H_C(\boldsymbol{S}).$$

When inducing a decision tree, we can choose the best test at each iteration by selecting the one which gives the largest information gain.

Top-down induced clustering trees

Clustering is a descriptive learning task that aims at finding subgroups (called *clusters*) of similar instances within a group of instances. Blockeel et al. [14] discusses how a simple extension of decision tree learning leads to a *divisive conceptual clustering* algorithm. More specifically, they point out that each node of a tree can be viewed as a cluster and that the tree as whole defines a hierarchy of clusters. Using this viewpoint, they show that the top-down induction approach of decision trees discussed in the last section can also be used for clustering. To that aim, the splitting procedure takes into account the distance between the resulting clusters (or alternatively, the intra-cluster similarity) given by the descriptive attributes. For example, the criterion to split a node can be the maximization of the inter-cluster distance between the two child nodes. As for classification trees, we can also use information gain as the splitting criterion in this top-down clustering procedure. We can do it by calculating the entropy for every descriptive attribute and averaging the resulting entropy values. We discuss other heuristics for inducing clustering trees in Chapters 3 and 4.

The *clustering tree* (hierarchy of clusters) resulting from this procedure is a *conceptual clustering* because the tree explicates not only the hierarchical structure of the data but also the clusters, which are described by the tests in the internal nodes of the tree. This conceptual property is one of the main features that differ *top-down induced clustering trees* from *agglomerative clusterings* (i.e., clusterings originating from iteratively merging instances in a bottom-up way).

Clus - unifying predictive learning and clustering

Clus [30] is an open source decision tree learner that can be used for both predictive learning and clustering. Clus is based on the ideas discussed in the last section, which unifies these two learning tasks in the decision tree framework.

With the right choices of parameters, Clus works similarly to decision tree learners such as C4.5 [17] or CART [102]. We use Clus in this way to build classification trees in Chapter 6. The use of Clus for building conceptual clustering trees is exploited in Chapter 3 and 4.

Clus can also be used for other learning settings such as multi-task learning, multi-label classification and semi-supervised learning. Furthermore, it can also be used for rule learning. These settings are, however, outside the scope of the work presented in this thesis.

2.2.3 Evaluating the learning performance

Several criteria may be used to evaluate the performance of a learned model. Different measures evaluate different characteristics of the model. Furthermore, different kinds of learning tasks often require different evaluation measures which take into account the specificities of the task into consideration. For example, to evaluate a predictive model, we need different measures than to evaluate a descriptive model.

We briefly discuss evaluation in the context of classification and clustering, which are the learning tasks considered in this thesis.

Evaluation of classification models

In the context of classification, evaluation measures are often defined from a so-called *confusion matrix*, which contains the numbers of examples correctly and incorrectly classified for each class. Table 2.1 shows the confusion matrix for a *binary classification problem* (i.e., classification problem with only two classes - positive and negative). The FP, FN, TP and TN concepts may be described as:

- *False positives* (FP): examples incorrectly predicted as belonging to the positive class.
- *False negatives* (FN): examples incorrectly predicted as belonging to the negative class.
- *True positives* (TP): examples correctly predicted as belonging to the positive class.
- *True negatives* (TN): examples correctly predicted as belonging to the negative class.

Table 2.1: Confusion matrix for a binary classification problem.

	Predicted class	
Actual class	Positive	Negative
Positive	|TP|	|FN|
Negative	|FP|	|TN|

Accuracy (ACC) is the most straightforward measure to evaluate the predictive performance of a classifier. It calculates the percentage of correct predictions. Equation 2.1 shows how ACC is computed.

$$ACC = \frac{|TN| + |TP|}{|TN| + |TP| + |FN| + |FP|} \tag{2.1}$$

Other measures that are often used to evaluate classifiers are recall, precision, and specificity, whose calculations are shown in equations 2.2, 2.3, and 2.4, respectively. *Recall* (also known as sensitivity or true positive rate) is the proportion of positive examples which were correctly predicted as positive; *specificity* is the proportion of negative examples correctly predicted as negative; and *precision* is the proportion of positive predictions which were correctly predicted.

$$Recall = \frac{|TP|}{|TP| + |FN|} \tag{2.2}$$

$$Specificity = \frac{|TN|}{|TN| + |FP|} \tag{2.3}$$

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{2.4}$$

The previous measures require a fixed discrimination threshold to determine the predicted class. For example, in a binary problem this fixed threshold is 0.5. Thus, a decision tree will classify an instance as positive if more than 50% of the instances in the leaf responsible for the prediction are positive. However, in some situations it is of interest to evaluate a classifier regardless of its threshold. It may be desirable, for example, to change the threshold in response to changing class or cost distributions [43]. In this context, it is common to use ROC (Receiver Operating Characteristics) analysis to evaluate classifiers.

A *ROC curve* is a plot where the discrimination threshold is varied. The x-axis of this curve corresponds to the specificity and the y-axis corresponds to the recall. ROC analysis is suitable, in particular, to evaluate the ranking ability of a classifier, as we discuss in Chapter 6. A quantitative way to compare classifiers based on their ROC curves consists of comparing the area under their ROC curves (AUC). This area estimates the probability that a randomly selected positive instance is ranked before a randomly selected negative instance. A perfect rank has an AUC equal to 1.

Clustering evaluation

Clustering evaluation can be performed in two main scenarios: (1) when a *reference clustering* (i.e., the true clustering or a gold standard solution) is given; (2) when no reference clustering is given. In the former case, the quality of the predicted clustering is calculated by comparing it to the reference one. In the latter case, the quality of the clustering is usually calculated based on the gain in information provided by the predicted clustering.

Rand index [104] is an example of a clustering evaluation measure for the case where a reference clustering is given. This measure considers that every pair of instances in a clustering defines a clustering decision: the pair is either clustered together or not. If a pair of instances is clustered together (or separately) in both the clustering being evaluated and in the reference clustering, the clustering decision is considered to be correct. Conversely, if a pair of instances is clustered together in one clustering but not in the other one, the clustering decision is considered to be incorrect. The rand index of a clustering is obtained by dividing the number of correct clustering decisions by the total number of decisions made (i.e., the total number of pairs of instances). Note that this is the same idea used in the accuracy calculation. In Chapter 4, we introduce other measures that can be used to compare clusterings. We use them to evaluate protein subfamily clusterings.

Category utility (CU) [53] is a clustering evaluation measure for the case where no reference clustering is given. It computes the improvement of the predictability of attributes given the clustering, in comparison with the situation in which no clustering is defined. For example, clustering adult persons according to their continent of origin gives us more homogeneous groups in terms of the physical characteristics height, hair color, and eye color, than in the situation where everyone was part of one big group. In this case we can say that we improve the overall predictability of the attributes height, hair color, and eye color, given the clustering. While, if we would cluster the same group of people according to their age, we would have little or no improvement in the predictability of those same attributes.

The definition of category utility is given by Equation 2.5, where $Pred(A|C)$ (Equation 2.6) measures the predictability of the descriptive attributes A giving the clustering C, $Pred(A)$ (Equation 2.7) measures the predictability of A when no clustering is defined, and k is the number of clusters. Note that the division by the number of clusters is important to have a trade-off between improvement of the predictability of attributes and the number of clusters.

$$CU(C) = \frac{Pred(A|C) - Pred(A)}{k} \tag{2.5}$$

$$Pred(A|C) = \sum_{l=1}^{k} Pr(C_l) \sum_i \sum_j Pr(A_i = a_{ij}|C_l)^2 \quad (2.6)$$

$$Pred(A) = \sum_i \sum_j Pr(A_i = a_{ij})^2 \quad (2.7)$$

In Equation 2.6, $Pr(C_l)$ is the probability of an arbitrary instance to belong to cluster C_l, i ranges over the instance attributes, j ranges over the possible values for each attribute A_i, $Pr(A_i = a_{ij}|C_l)$ is the probability that attribute A_i has value a_{ij} - given that the instance belongs to cluster C_l. In Equation 2.7, $Pr(A_i = a_{ij})$ is the probability that A_i has value a_{ij} when no clustering is defined.

2.3 Summary

In this chapter, we explained basic concepts from biology and machine learning that will be used in the rest of this thesis. More specifically, we introduced the main concepts related to nucleotide and amino acid sequences, defined the role of these sequences in an organism, and gave few examples of biological tasks that involve sequence analysis. Additionally, we explained artificial learning and the main concepts related to its definition. We also introduced decision trees, which is the machine learning technique we focus on.

Concepts that are specific to only one of the chapters of the thesis are introduced in the chapter itself.

Chapter 3

Top-down induction of phylogenetic trees

This is the first of two chapters where we use a divisive conceptual clustering approach to partition a set of biological sequences. In this chapter[1], we do it in the context of phylogenetic tree reconstruction. In the next chapter, we consider it in the context of protein subfamily identification.

3.1 Introduction

In phylogenetic tree reconstruction, one is interested in deriving from a set of aligned DNA or amino acid sequences the most likely phylogenetic tree. Many methods have been proposed for this. One popular approach consists of computing a dissimilarity measure between each pair of sequences, and then using the resulting matrix to infer the tree. Methods that use this approach are called distance based methods, and are represented by the well-known Neighbor Joining (NJ) [111, 122] algorithm.

NJ is essentially a so-called agglomerative hierarchical clustering algorithm: starting from one cluster per sequence, it iteratively merges clusters of sequences until a single cluster is obtained. While agglomerative clustering algorithms are among the most popular ones for clustering, many other clustering algorithms exist. Among these algorithms, one can distinguish extensional and conceptual

[1] Based on the paper "Top-down induction of phylogenetic trees" [Vens, Costa and Blockeel 2010].

clustering algorithms. In extensional clustering, a cluster is described by enumerating its elements, whereas in conceptual clustering, a cluster is described by listing properties of the cluster's elements, using a particular description language. In the context of phylogenetic analysis, a natural conceptual language would be one that refers to polymorphic locations (i.e., positions that have more than one amino acid residue); for instance, one cluster might be described as "all sequences having a C at position 72 and A at position 31".

If we assume that it must be possible to describe clusters using a particular language, the space of all possible clusterings is greatly reduced. This enables us to use a divisive clustering approach which starts from a single cluster containing all sequences and repeatedly divides it into subclusters until all sequences form a different cluster. Normally, given a set of N sequences, there are $\frac{2^N}{2} - 1$ distinct ways to split it into two non-empty subsets. But if we assume that the split can be described by referring to a particular polymorphic location, the number of splits is linear in the length of the sequences, and constant in the size of the set, making such a divisive method computationally feasible, and potentially faster than agglomerative methods. A similar observation was made by Arslan and Bizargity [6], who were the first to propose a top-down clustering method for phylogenetic tree reconstruction. Differences between their algorithm and our work are described in the related work section.

Blockeel et al. [14] discuss how a simple extension of decision tree learning leads to a (general-purpose) divisive conceptual clustering algorithm (see Section 2.2.2). In this chapter we address the question to what extent this approach lends itself to phylogenetic tree reconstruction. If it works well, that is, if it yields phylogenetic trees with an accuracy comparable to that of existing methods, such an approach would have a number of important advantages over existing methods. First, as just argued, it may be faster than methods based on agglomerative clustering. Second, as each division into subclusters is defined by polymorphic locations, the resulting tree immediately gives an evolutionary trace. Third, by using an adequate stopping criterion or pruning procedure, the divisive method can be used not only to reconstruct complete phylogenetic trees, but also to identify subfamilies of genes or proteins; we investigate this in Chapter 4.

In order to study the top-down conceptual clustering approach in the context of phylogenetic tree reconstruction, we propose a method that is strongly based on the decision tree learner Clus [30]; the only change made to it is the heuristic used for splitting nodes.

The remainder of the chapter is organized as follows. Section 3.2 describes the phylogenetic tree reconstruction problem and discusses related work. Section 3.3 presents the proposed method. We thoroughly evaluate this new method

with respect to accuracy and efficiency in Section 3.4; the results show that our method builds trees with comparable accuracy to those produced by existing methods, with a tendency to perform better for highly symmetric trees, and somewhat worse for highly asymmetric trees. We conclude in Section 3.5.

3.2 Background and related work

We start this section by describing the phylogenetic tree reconstruction problem. Then, we present the classical approaches for this problem and briefly describe three well-known phylogenetic methods. Next, we discuss a top-down clustering method which is related to the one we propose. We finish the section with a short note on comparing phylogenetic trees.

3.2.1 Phylogenetic tree reconstruction

In phylogenetic tree reconstruction, one is interested in inferring the evolutionary relationships among genes and organisms by constructing a tree-shaped structure called a phylogenetic tree (see Figure 3.1). In this structure, the leaf nodes (called operational taxonomic units, or OTUs) correspond to the objects of the study of the phylogenetic analysis (e.g., the species for which we want to build the phylogenetic tree). The internal nodes are hypothetical progenitors of the leaf nodes, and are called hypothetical taxonomic units (HTUs). In the trees shown in Figure 3.1, A is an OTU and X is an HTU, for example. The root of the tree defines the direction of evolution. When the root place is not defined (Figure 3.1.a), the direction of evolution is unknown. Branch lengths can also be used to indicate the estimated evolutionary distance between nodes/units of the tree (OTUs and/or HTUs) : the larger the branch length connecting to units of the tree is the more evolutionarily distant these units are.

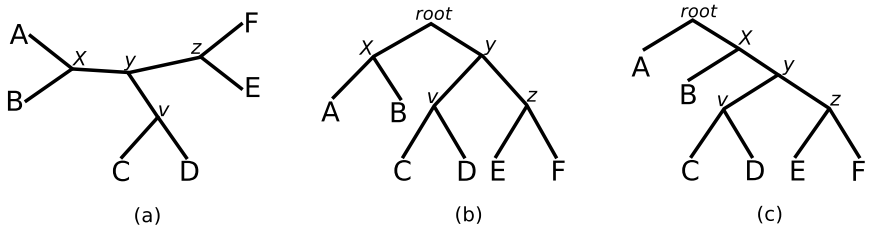


Figure 3.1: Three phylogenetic trees with the same topology, but different rooting definition: (a) undefined root; (b) root between HTUs X and Y; and (c) root between OTU A and HTU X.

Phylogenetic trees can be constructed based on morphological characteristics or molecular data (sequence data). We focus on phylogenetic tree reconstruction using sequence data.

3.2.2 Classical approaches

Phylogenetic methods can be grouped into two approaches according to the format of the data they use: character-state and distance based methods.

Character-state methods use any set of discrete characters, such as morphological characters, physiological properties, or sequence data [129]. In the case of sequence data, each column (position) of the multiple sequence alignment (MSA) correspond to a "character" and the nucleotides (or amino acids) at that position are "states". The algorithm looks for the tree that best represents the input data in a search space that can contain all theoretically possible trees or a subset of them (in the case of a heuristic search). For every tree being analyzed, the algorithm reconstructs the character-states of the internal nodes; the characters are usually considered independently. The idea is to select the topology for which the state of the internal nodes are as compatible as possible with those of the leaf nodes. In this section, we see one example of such a method: maximum parsimony [42].

Distance based methods, on the other hand, are specific to sequence data. They start by converting the aligned sequences into a matrix of distances between each pair of sequences. The idea is to compute a matrix that approximates the true genetic distance between the sequences. This is an approximation because every state of the alignment can be a result of multiple events, such as subsequent mutations at the same site. An accurate approximation is crucial for a good phylogenetic inference [127]. Once the pairwise distances have been calculated, different methods can be used to infer the tree. We see two examples: unweighted pair group method with arithmetic mean (UPGMA) [118] and neighbor joining (NJ) [111].

As distance based methods do not use the sequences directly, they are much more efficient than character-state methods. On the other hand, they lose evolutionary information by converting the sequences into a matrix. For example, as they do not retain the character-state of each position, they cannot reconstruct the state of internal nodes during the tree construction.

Maximum Parsimony

Maximum parsimony [42] searches for the tree (or alternatively, a set of trees) that requires the least number of character-state changes to explain the input sequence data. This idea is based on the principle of minimum evolution [21], which prefers simple evolutionary explanations to more complicated ones.

This analysis can be divided into two subproblems: (1) the small parsimony problem and (2) the large parsimony problem. The first corresponds to the question "given a tree topology T, what is the minimum number of character-state changes for T?". This question is answered by looking for an optimal way to label the internal nodes, which is usually performed by the so-called Fitch algorithm [42]. The second subproblem corresponds to searching for an optimal tree. This can be done by an exhaustive search (i.e., checking all possible trees) or by using approximate methods.

Unweighted pair group method with arithmetic mean

Unweighted pair group method with arithmetic mean (UPGMA) [118] is a simple agglomerative clustering method. At the beginning of the procedure, every sequence corresponds to an independent cluster. Then, it iteratively merges the two closest clusters until only one cluster is left. To calculate the distance between two clusters, the algorithm averages all pairwise distances between sequences from the two clusters. The tree returned by UPGMA is rooted and is determined by the order in which the clusters were merged.

The main drawback of this method is that it is assumes a clock-like behavior of mutations (i.e., that the mutation rate is constant over all branches of the tree). This makes UPGMA extremely sensitive to unequal substitution rates in different lineages [119]. For an example of this behavior, consider the two trees shown in Figure 3.2; the numbers indicate the branch lengths. The left tree assumes a clock-like behavior. As a result, all OTUs are equally distant from the root. This behavior is not assumed by the right tree. Consider OTUs A and B, for example. The branch that connects A to its parent node is longer than the one which connects B to the same node.

Neighbor joining

The neighbor joining (NJ) algorithm [111] is one of the most widely used methods for phylogenetic tree reconstruction. It is a heuristic estimation of the minimum evolution tree, which is the tree with minimal sum of branch lengths.

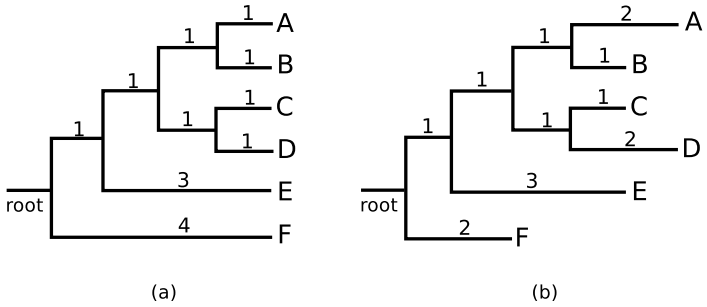


Figure 3.2: Example of (a) a tree that assumes a clock-like behavior of mutations and (b) a tree that allows different mutation rates across its branches.

Similarly to UPGMA, it uses an agglomerative clustering approach. However, it does not assume a clock-like behavior.

The algorithm starts with a fully unresolved star tree (Figure 3.3.a). Then, it computes the total branch length of the tree for every possibility of joining two OTUs (see Figure 3.3.b, for an example). The pair that yields the smallest total branch length is then merged to make a combined OTU. This operation results in a new star tree (Figure 3.3.c), which has one OTU less than the previous tree. After the distance matrix has been updated with the pairwise distances between the new OTU and the remaining ones, the whole process starts again. The algorithm stops when there are only three OTUs left. It then returns the fully resolved tree, which is by definition an unrooted tree.

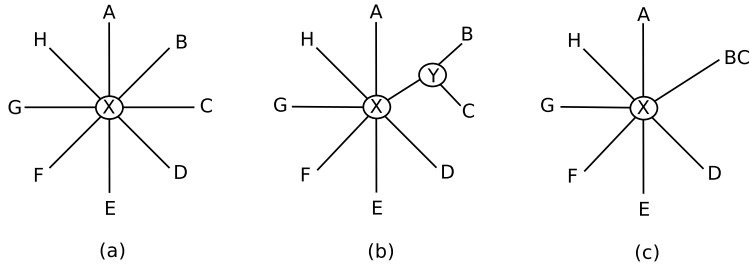


Figure 3.3: Illustration of the NJ merging procedure: (a) fully unresolved star tree; (b) tree resulting from merging OTUs B and C; (c) star tree considered by the NJ algorithm in the next iteration.

As our proposed method uses a heuristic function which is a generalization of the one used by NJ, we explain the latter next. This explanation is based on the one presented by Saitou and Nei [111].

When deciding which pair of OTUs to merge, the NJ algorithm needs to estimate the total branch length of trees with a structure similar to the one shown in Figure 3.3.b (i.e., a star tree with a branch that separates two sequences - the ones being merged - from the other ones). Consider, for example, the tree T in Figure 3.3.b. To compute its total branch length ($TBL(T)$), we can divide T in three parts and make the calculations separately: (1) the branch between X and Y; (2) the subtree containing the two sequences being merged (B and C); and (3) the subtree containing the remaining OTUs (we call it T').

The calculation of the branch length between X and Y (L_{XY}) is given by Equation 3.1, where |T| is the number of OTUs in T, and $D_{o_i o_j}$ is the distance between OTUs o_i and o_j. The first term within brackets sums all distances that include L_{XY}, and the other two terms exclude the irrelevant branch lengths that are added by the first term. We divide by $2(|T|-2)$ because L_{XY} was added that many times in the sum.

$$L_{XY} = \frac{1}{2(|T|-2)}[\sum_{o_i \in T'}(D_{o_i B} + D_{o_i C}) - (|T|-2)(D_{BC}) - 2\sum_{o_i \in T'} L_{o_i X}] \tag{3.1}$$

The total branch length of the subtree containing B and C corresponds to D_{BC}. Finally, we use Equation 3.2 to calculate the total branch length of T'. Since each branch length is counted $|T'|-1$ times when all distances are added, we divide the sum by that factor.

$$TBL(T') = \frac{1}{|T'|-1}\sum_{\substack{o_i, o_j \in T' \\ i<j}} D_{o_i o_j} \tag{3.2}$$

Adding L_{XY}, D_{BC}, and $TBL(T')$ together, and performing the right substitutions, we obtain the formula given by Equation 3.3.

$$TBL(T) = \frac{1}{2(|T|-2)}\sum_{o_i \in T'}(D_{o_i B} + D_{o_i C}) + \frac{1}{2}D_{BC} + \frac{1}{|T|-2}\sum_{\substack{o_i, o_j \in T' \\ i<j}} D_{o_i o_j} \tag{3.3}$$

The algorithm we have just described has a complexity $O(N^5)$ [49]. However, Studier and Keppler [122] have presented an alternative version, which is

$O(N^3)$. This makes NJ one of the most efficient algorithms for phylogenetic tree reconstruction [112].

3.2.3 PTDC - Phylogeny by Top-Down Clustering

Another efficient method, and among all methods the one that most closely resembles ours, is the PTDC (Phylogeny by Top-Down Clustering) algorithm [6]. This algorithm shares with our proposal the idea of recursively partitioning clusters. The main differences are: (1) our approach makes the link to the decision tree learning framework, which allows us to exploit the highly developed state-of-the-art in that area; (2) PTDC's heuristic maximizes the average of all pairwise distances between sequences of different clusters, making it similar to the UPGMA algorithm, from which it inherits the sensitivity to unequal substitution rates in different lineages, whereas our method uses a heuristic that approximates the NJ criterion; (3) while PTDC splits clusters based on equality of subsequences, our approach creates splits based on a single, most informative, position; this gives it an efficiency advantage.

3.2.4 Comparing phylogenetic trees

As different methods can infer different phylogenetic trees, it is important to have an objective way to compute differences between trees [108]. For this reason, many metrics to compare phylogenetic trees have been proposed in the literature [108, 35, 136]. These measures are also particularly important for the case where we want to compare the results of a method to a gold standard solution (e.g., for synthetic datasets, where the true phylogenetic tree is known). Next, we briefly describe the quartet distance [35], which we use in the empirical evaluation presented in this chapter.

A group of four OTUs is the smallest group for which there is more than one possible unrooted tree topology (see Figure 3.4). Based on this observation, Estabrook et al. [35] proposed the quartet distance, which is the number of quartets (subtrees induced by four leaves) that have a different topology in the two trees being compared.

To obtain the quartets of a tree, we consider every set of four OTUs, and for each one of them we eliminate all branches that are not part of a path between two of the four OTUs. Examples of the quartets that can be obtained from the tree in Figure 3.1.a are: [AB|CD], [AB|DE], and [BC|EF].

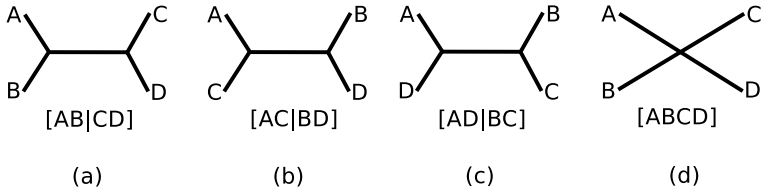


Figure 3.4: Four possible unrooted trees that can be defined for four OTUs. Note that, within binary trees, only the topologies (a)-(c) can be obtained. Topology (d) is obtained when multifurcation is allowed.

3.3 Proposed method

The proposed method is built on top of the decision tree learner Clus [30], which we introduced in Section 2.2.2. We call the new method Clus-φ.

Following the Clus-based approach (see pseudocode in Figure 3.5), Clus-φ starts from a single cluster containing all sequences (which are given as input in the form of an MSA), and recursively splits clusters up to the level of single sequences. Each split is determined by a test of the form "$p = a$", or more generally "$p \in S$", with p a location, a a character-state (amino acid or nucleotide), and S a set of character-states. For example, the test "p6 = R" checks for the occurrence of amino acid R at position 6, and splits the set of sequences in two subsets, one containing all sequences with an R at position 6 and one containing all other sequences (see Figure 3.6). Similarly, the test "p15 $\in \{P, G\}$" creates one subset containing all sequences with a P or G at position 15, and one containing all other sequences.[2]

To choose which split to use at each iteration of the tree growing procedure, the algorithm tries all possible tests. More specifically, it tentatively splits the current cluster according to each test, evaluates this split (according to a certain heuristic), and remembers the best one. It finally splits the set according to the best test encountered.

The only question that remains, then, is how to determine which is the "best split" to choose at each iteration (i.e, the heuristic function). Intuitively, one could argue that an "old" mutation, i.e., one that occurred a long time ago in evolutionary history, is more likely to have led to two different branches that by

[2]Gaps are treated as an extra nucleotide/amino acid. Tests that check for sets of nucleotides are treated differently than those that check for sets of amino acids. For the former, as there are only four nucleotides, all possible combinations of nucleotides are tested. For the latter, we use a greedy strategy for the sake of computation efficiency: first all amino acids are considered individually, then iteratively the best one is expanded with all possible amino acids, until no improvement is obtained.

GrowTree

// **Input:** Data to be partitioned D
// **Output:** Phylogenetic tree built with sequences in D

For each possible test t of form $p_i \in S$ **do**
    Partition the data according to t
    Calculate the heuristic for the resulting partition
Choose the test with the best heuristic value
Partition the data according to this test
Run GrowTree on each non-singleton subset

Figure 3.5: Pseudocode for the Clus-based approach. This pseudocode is an instantiation of the one we showed in Figure 2.8 (Section 2.2.2). Note that GrowTree is called recursively. In the end, the first call of GrowTree will return the completely resolved tree.

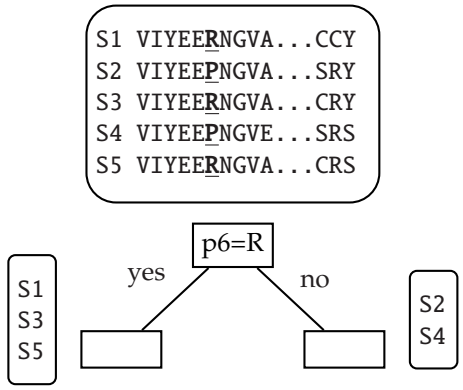


Figure 3.6: Illustration of how our method uses polymorphic mutations to induce a clustering tree in a top-down way.

now have grown far apart, than a more recent mutation. As we are building a rooted tree, where we hope to find older mutations nearer to the root, we should prefer tests that create subsets that are far apart. A natural heuristic for selecting tests is then simply: compute the average distance between elements from different subsets induced by the test; choose the test that maximizes that average distance. This is the heuristic that Arslan et al. use [6]. Alternatively, one could use an extension of the information gain heuristic that is commonly used for classification trees. This extension consists of looking for the split that maximizes the average information gain with respect to the descriptive attributes

(i.e., the sequence positions) at each iteration of tree induction procedure.

These standard heuristics, however, do not take into account the particular setting of phylogenetic tree reconstruction. Using the average distance heuristic, one essentially gets the top-down counterpart of the UPGMA algorithm and inherits its main drawback (sensitivity to unequal substitution rates in different lineages). The heuristic used by NJ, on the other hand, uses a more advanced calculation for deciding which two clusters to merge, and this yields better results. This raises the question whether a top-down counterpart of NJ's heuristic can be developed. Such a heuristic would have to estimate the total branch length of the tree that will finally be constructed. We introduce this fuction next. See Appendix A for a more detailed explanation.

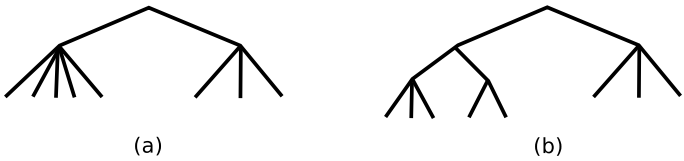


Figure 3.7: Split topologies considered during the top-down tree construction: (a) Clus-φ (root); (b) Clus-φ (non-root).

Using the same reasoning used in the NJ's calculations (see Section 3.2.2), we can define an equivalent heuristic function for splitting the root node:

$$TBL(T\{T_L, T_R\}) = \frac{1}{|T_L||T_R|} \sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} + \frac{1}{|T_L|} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + \frac{1}{|T_R|} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}, \quad (3.4)$$

where T_L and T_R denote the set of OTUs (sequences) in the left and right subtrees of the split, respectively, in tree T. This formula computes the total branch length of a "double star"-shaped topology (see Figure 3.7.a), which is an upper bound for the actual total branch length of the final tree. It is a generalization of the NJ heuristic (Equation 3.2), which has $|T_L| = 2$ and $|T_R| = |T| - 2$, with $|T|$ the number of OTUs in the tree for which we want to calculate the total branch length.[3]

[3]Note that the first term of Equation 3.4 corresponds to the calculation of the heuristic that maximizes the average inter-cluster distance. It might look counter-intuitive that the Clus-φ heuristic, which is a minimization function, uses the same calculation performed by a maximization function in its formula. However, there is a trade-off between the first term of the heuristic function and the other two terms, as we explain in Section A.3 (Appendix A).

The previous equation can be rewritten into a more efficient formulation,

$$TBL(T\{T_L, T_R\}) = \frac{1}{|T_L||T_R|}\Big(\sum_{\substack{o_i,o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} + (|T_R|-1)\sum_{\substack{o_i,o_j \in T_L \\ i<j}} D_{o_i o_j} + (|T_L|-1)\sum_{\substack{o_i,o_j \in T_R \\ i<j}} D_{o_i o_j}\Big),$$

where the first summation is constant for the node to be split. Therefore, this summation only needs to be calculated once for each node.

For splitting the other internal nodes, a more complex heuristic function is needed, since the particular split influences the length of other branches in the tree, and hence, the total branch length (Figure 3.7.b). Again, using a similar reasoning as applied by Saitou and Nei [111], it can be verified that the function to minimize is given by

$$TBL(T\{T_L, T_R, T_O\}) = \frac{1}{2|T_L||T_R|}\Big(\sum_{\substack{o_i,o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} + (2|T_R|-1)\sum_{\substack{o_i,o_j \in T_L \\ i<j}} D_{o_i o_j} + (2|T_L|-1)\sum_{\substack{o_i,o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{|T_R|}{|T_O|}\sum_{\substack{o_i \in T_O \\ o_j \in T_L}} D_{o_i o_j} + \frac{|T_L|}{|T_O|}\sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j}\Big) + \frac{1}{|T_O|}\sum_{\substack{o_i,o_j \in T_O \\ i<j}} D_{o_i o_j}.$$

with T_O denoting the set of other OTUs in the tree (i.e., not in $T_R \cup T_L$). The first summation is constant for the node being split and only needs to be calculated once. The last summation is also constant. As it is multiplied by $\frac{1}{|T_O|}$, which is also constant, this summation can be dropped from the calculation when choosing the best split.

When this heuristic is used, a modification is needed in the pseudocode we showed in Figure 3.5. More specifically, an extra input parameter is needed. As, at each iteration of the tree building, the heuristic needs the information

about all the sequences to estimate the total branch length of the current tree, the recursive calls of GrowTree need to be given two input parameters: (1) the original data (i.e., data containing all sequences); (2) the data to be partitioned D for that call of the procedure (i.e., data containing the sequences in the node to be split).

The computational complexity of a decision tree learning method is $O(aN \log N)$ with a the number of tests and N the number of elements in the original dataset, under the assumption that a reasonably symmetric tree is built (the depth of which is logarithmic in the number of leaves) and the evaluation of a single test takes linear time in the size of the dataset (this is the case of the information gain heuristic, for example). This scales much better in N than agglomerative methods, which have complexity $O(N^3)$ [122]. As such, such a divisive method may be much more efficient when analyzing large sets of sequences. Our proposed heuristic is, however, quadratic, rather than linear, in the number of sequences, which increases the overall complexity of the method to $O(aN^2 \log N)$. If we use Clus with the heuristic that maximizes the average inter-cluster distance, we obtain a computation complexity of $O(aN^2)$.

3.4 Empirical evaluation

We evaluate our alternative method for phylogenetic tree construction on a number of datasets. In all experiments, we use the Jukes-Cantor correction formula [70] to compute the genetic distance between two DNA sequences, and the Jones-Taylor-Thornton matrix [69] for amino acid sequences. We measure differences between trees using the quartet distance (see Section 3.2.4). To that aim, we use the tool QDist [86].

3.4.1 Real datasets

In a first set of experiments, we check how much Clus-φ trees differ from the ones returned by NJ. As a reference point, we include the difference between parsimony methods and NJ.[4] To construct the NJ and parsimony trees we used the programs neighbor and dnapars/protpars from the Phylip software package [37], respectively.

Table 3.1 reports the quartet distance for 11 datasets used in [112]. While the trees generated by Clus-φ are very similar to those generated by NJ for some datasets (for datasets Chimp and hivALN, Clus-φ and NJ generated identical

[4]When parsimony analysis returns multiple trees, we report the average difference.

Table 3.1: Results for real datasets in terms of quartet distance. The quartet distances are shown in the last two columns of the table. The remaining columns show information about the datasets.

Datasets	Seq type	Nb seq	Alignment length (bp)	NJ vs. Clus-φ	NJ vs. Pars
Chimp	DNA	5	896	0	2
cynmix	DNA	32	3080	11680	16543
Primates	DNA	21	1500	184	632
SIV	DNA	41	6042	21350	10608
hivALN	DNA	14	2352	0	156
InvertebrateEF	DNA	16	1050	200	152
mtDNA	DNA	17	1998	218	26
VertebrateMtCOI	DNA	8	1509	178	31
g3pdh	Protein	14	313	180	65.33
gpd	Protein	12	234	52	18
gdpAA	Protein	12	422	57	97.50

phylogenetic trees), there are larger differences for other datasets. However, this variation is comparable to the variation between the parsimony method and NJ.

The question is then whether, in those cases where NJ and Clus-φ differ, any of them is more likely to be correct than the other one. To check this, we tested Clus-φ on a number of synthetic datasets, where the correct tree is known. These experiments are discussed in the next section.

3.4.2 Synthetic datasets

The synthetic datasets were generated by simulating an evolutionary process, using the coding sequence simulation program EvolveAGene3 [58], with a randomly generated DNA sequence as input. By default, this software produces symmetric trees, i.e., binary trees that have all leaves at the same depth. However, also random tree topologies can be produced. We set all parameters of EvolveAGene3 to their default values, except for the average branch length, which has no default value. We arbitrarily set it to 0.05 in all experiments, meaning that each branch has an average mutation rate of 5%.

First, we evaluate our proposed heuristic. Second, we compare the results of Clus-φ with those of NJ. Finally, we investigate the influence of the number of sequences on quartet distance and computational cost for both NJ and Clus-φ.

Analysis of the heuristic

To evaluate our proposed heuristic, we compare trees constructed using our heuristic with trees constructed using other heuristics for decision trees. The first heuristic is based on the average information gain with respect to the sequence positions; we call this implementation Clus-IG. The second heuristic is similar[5] to the one used in the PTDC algorithm [6]. It recursively splits the data by looking, in each step, for the two subclusters with the largest average inter-cluster distance. We call this implementation Clus-MaxAvgDist.

In order to perform this experiment, we generated 400 synthetic datasets: 200 based on symmetric topologies and 200 based on random topologies, with an input sequence length of 250 (200 datasets) and 900 (200 datasets),[6] each dataset containing 128 sequences. Table 3.2 presents a summary of the results, in terms of the number of wins/losses of Clus-φ in the comparison with Clus-IG and Clus-MaxAvgDist, according to the quartet distance from the inferred trees to the true tree. As can be seen from the results, Clus-φ presents a higher number of wins than Clus-IG, and is better than Clus-MaxAvgDist for all datasets. It shows that, in the context of phylogenetic tree reconstruction, the new heuristic yields better results than standard heuristics for decision trees.

Table 3.2: Comparison of different heuristics for phylogeny tree reconstruction. We show the number of wins/losses of Clus-φ's heuristic compared to information gain (Clus-IG) and inter-cluster distance (Clus-MaxAvgDist).

	Clus-φ vs Clus-IG	Clus-φ vs Clus-MaxAvgDist
Symmetric	113/78 (9 ties)	200/0
Random	113/87	200/0

Comparison between Clus-φ and NJ in terms of similarity to the true tree topology

We now compare the similarity with the true tree for NJ and Clus-φ. We generated 100 synthetic datasets based on symmetric topologies and 100 datasets based on random topologies, containing 128 sequences and using an input sequence of length 900. A summary of the results, in terms of the number of datasets for which each method presents the best performance, is shown in

[5] The only difference between our implementation and the one discussed in [6] is that we create splits based on a single position instead of based on subsequences.

[6] As insertions and deletions are allowed in EvolveAGene3, the final sequence length is usually larger than the input sequence length, e.g. for an input sequence of length 900, the length of the final sequences is around 1000.

Table 3.3; the numbers in boldface represent the largest number of wins for each line. On average, NJ performs slightly better than Clus-φ; however the table shows a large difference in the results for symmetric and random tree topologies. For symmetric trees, the Clus-φ tree is in general closer to the true tree than the tree produced by NJ. For the random tree topologies, on the other hand, Clus-φ is closer to the true tree in only 5 cases. We observed that Clus-φ tends to build more symmetric topologies, which explains its inferior performance for random trees.

Interestingly, we noted that in a few cases, the trees produced by Clus-φ are identical to the true topologies; this occurred for 5 out of 100 cases for the datasets generated from symmetric topologies. For NJ, that is never the case. The probability of observing at most 0 correct predictions for NJ and at least 5 for Clus-φ, under the null hypothesis that both have the same probability p of making a correct prediction, depends on p but is always less than 0.01; this implies that the hypothesis that Clus-φ does not have a higher probability of predicting the correct tree than NJ for symmetric topologies is rejected at the 1% significance level.

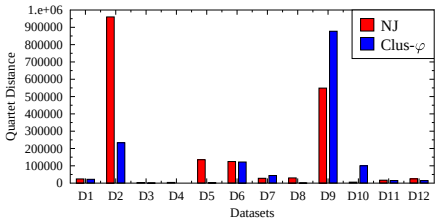
Table 3.3: Number of wins of NJ and Clus-φ for synthetic datasets.

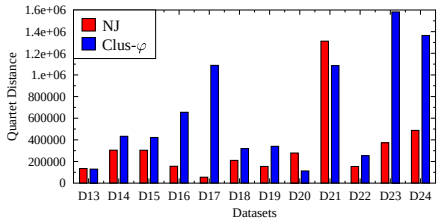
	NJ	Clus-φ
Symmetric	32	**68**
Random	**95**	5

In Figure 3.8 we show the quartet distance for 12 datasets of each kind, in order to analyze the results in more detail. We can see that for random trees the differences between the two methods are larger. The graphs also show the strong variation of tree quality with the dataset: the dataset has a larger influence on the overall performance than the choice of method.

From the results shown in Table 3.3 and Figure 3.8 we can conclude that Clus-φ tends to perform better for symmetric tree topologies, while NJ tends to perform better for random topologies.[7] In a sense, these settings are at both ends of a spectrum. The question is then how the results differ for datasets based on trees that are neither perfectly symmetric nor completely random, which is what we expect to occur in nature.

[7]Experiments considering the parsimony method (using the program protpars from the Phylip software package [37]) revealed similar results: Clus-φ also tends to perform better than the parsimony method for symmetric tree topologies, and worse for random topologies. Compared to NJ, the parsimony method presents better results for random topologies and worse results for symmetric topologies.





D1	D2	D3	D4	D5	D6
0.91	0.24	0.16	0.00	0.01	0.98
D7	**D8**	**D9**	**D10**	**D11**	**D12**
1.57	0.02	1.60	17.63	0.87	0.59

D13	D14	D15	D16	D17	D18
0.96	1.42	1.39	4.20	19.88	1.52
D19	**D20**	**D21**	**D22**	**D23**	**D24**
2.20	0.40	0.83	1.64	4.25	2.80

Figure 3.8: Results for symmetric (left) and random (right) trees, in terms of quartet distance. Graphs: absolute values; Tables: quartet distance of Clus-φ relative to that of NJ.

Table 3.4: Analysis of the effect of the symmetry of the tree on the performance of NJ and Clus-φ, according to the quartet distance.

Dataset	NJ	Clus-φ
Symmetric	32	**68**
5 steps	**56**	44
10 steps	**56**	44
15 steps	**53**	47
20 steps	**68**	32
25 steps	**65**	35
30 steps	**63**	37

Dataset	NJ	Clus-φ
35 steps	**65**	35
40 steps	**66**	34
45 steps	**77**	23
50 steps	**76**	24
55 steps	**77**	23
60 steps	**81**	19
Random	**95**	5

To investigate this question, we generated datasets based on a series of tree topologies, starting from a perfectly symmetric tree, and gradually adding more random tree structure. More precisely, we considered a tree operation that takes two subtrees, one consisting of a single leaf, and another one consisting of one internal node and two leaves, and switches them. Table 3.4 reports the results for an experiment that counts the number of wins for NJ and Clus-φ on 100 datasets with an increasing number of operations. Each dataset again has 128 sequences generated from a DNA input sequence of 900 positions.

As can be seen from the results, the performance of Clus-φ, compared to NJ, decreases with the increase of the number of modifications in the symmetric tree. However, it is important to notice that this decrease of the performance of Clus-φ occurs gradually, which means that it presents good results not only

for completely symmetric trees, but also for almost symmetric trees: up till 40 steps, Clus-φ finds a better approximation of the true tree in one out of three datasets. As an illustration, Figure 3.9 shows a topology with 40 steps from symmetry.

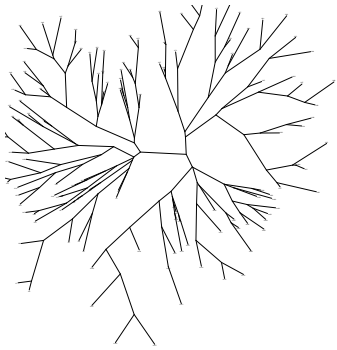


Figure 3.9: Tree topology with 40 steps from symmetry.

Analysis of the influence of the number of sequences on quartet distance and computational cost

In this section, we analyze the effect of the number of sequences on the performance of the algorithms, both in terms of tree reconstruction, and computational cost. For this analysis, we use datasets based on symmetric tree topologies.

To analyze how NJ and Clus-φ scale in the number of sequences, we generated a number of synthetic datasets containing sequences of the same length,[8] but with an increasing number of sequences.

Figure 3.10 shows run times for datasets with 300 nucleotides. Each point in the curve shows the average run times over 20 datasets of the specified size. As we can see, between 1024 and 2048 sequences, the run times of Clus-φ become smaller than those of NJ. For 8000 sequences, Clus-φ is about 5 times faster.

Figure 3.11 shows the run times for datasets with 900 nucleotides. In this case, NJ is more efficient than Clus-φ. However, the relative run time difference

[8] To ensure an equal dataset length, we disabled insertions and deletions here.

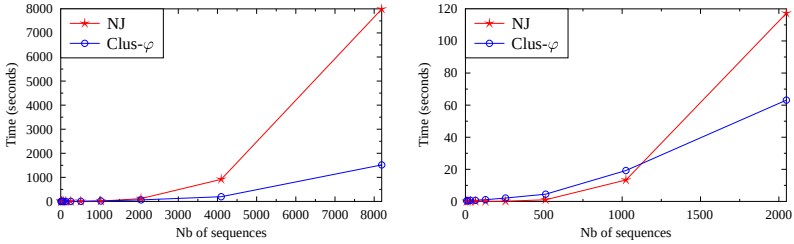


Figure 3.10: Running times for Clus-φ and NJ; the right graph zooms in on the interval 0-2048 of the left graph. Sequence length: 300.

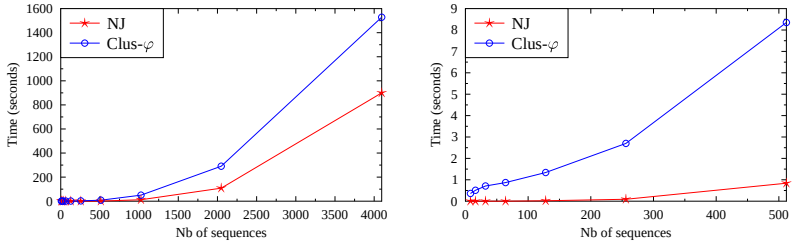


Figure 3.11: Running times for Clus-φ and NJ; the right graph zooms in on the interval 0-512 of the left graph. Sequence length: 900.

between both methods decreases, and we expect Clus-φ to be faster than NJ from a certain dataset size on.[9]

We also computed the quartet distance of the trees produced by NJ and Clus-φ to the true tree for the datasets with 900 nucleotides (see Table 3.5). It can be noted that Clus-φ obtains better results than NJ for datasets with few sequences. However, between 128 and 256 sequences, the results change, making NJ the best method for a large number of sequences. The reason for this is related to the relatively small sequence length: in order to generate 256 sequences or more with only 900 nucleotides, each branch having on average 45 mutations, a lot of nucleotide positions are likely to have many mutations. This negatively influences Clus-φ, because it cannot find the right splits anymore.

[9]It is infeasible to generate datasets with more than 4000 sequences of length 900 with EvolveAGene3.

Table 3.5: Analysis of the influence of the number of sequences on the performance of NJ and Clus-φ, according to the quartet distance. We show the number of wins of NJ and Clus-φ for synthetic datasets with sequences of length 900 (Figure 3.11).

nb Sequences	NJ	Clus-φ	Ties
8	0	**1**	19
16	2	**7**	11
32	2	**9**	9
64	5	**13**	2
128	6	**14**	0
256	**11**	9	0
512	**13**	7	0
1024	**13**	7	0
2048	**18**	2	0
4096	**15**	5	0

3.4.3 Comparison to PTDC

As a last experiment, we compare Clus-φ to PTDC [6]. The authors have used a single dataset to evaluate their method and report the resulting tree in their article. The dataset was prepared by Rennert et al. [107] and corresponds to an alignment of amino acid sequences from the Runt domain of RUNX genes from a set of species. Rennert et al. [107] who also report a maximum likelihood tree for the alignment, which we use as reference tree. For this experiment, both PTDC and Clus-φ calculate pairwise protein distances using the PAM1 substitution matrix [29]. Given that the reference tree in [107] is not binary (it contains a node with six branches), it is impossible[10] to calculate the quartet distance. Therefore, for this experiment, we compare the trees using the symmetric difference measure. The symmetric difference between two trees is the number of binary node partitions that are found in one tree and not in the other. The symmetric difference to the reference tree is 11 for PTDC, and 10 for Clus-φ. We conclude that our method finds a slightly better tree.

3.5 Conclusions

In this chapter, we proposed Clus-φ, a novel method for reconstruction of phylogenetic trees. The method differs strongly from classical phylogenetic

[10]The tool QDist [86], which we used to calculate the quartet distance between trees, only supports binary trees.

methods (such as maximum parsimony and NJ), both from an algorithmic point of view and from the point of view of the information it uses.

Clus-φ is based on a conceptual clustering method that extends the well-known decision tree learning approach. Starting from a single cluster, it repeatedly splits the sequences into subclusters until all sequences form a different cluster. To define the best split, our method uses a criterion that is close to NJ's optimization criterion, namely, constructing a phylogenetic tree with minimal total branch length.

Our method assumes that a split can be described by referring to particular polymorphic locations, which makes such a divisive method computationally feasible, and at the same time provides an evolutionary trace for the resulting tree topology.

We have shown that: (a) the heuristic used by Clus-φ gives better results than when standard heuristics for decision tree learning are used; (b) Clus-φ is close to the performance of NJ with respect to quality of the produced trees; (c) Clus-φ has a larger tendency to produce the correct tree than NJ for the cases where this tree is symmetric.

We propose Clus-φ not as a substitute for NJ or other standard methods for phylogeny reconstruction, but as a method to be used complementarily to these methods. As argued by Vandamme [129], there are no uniquely correct methods for inferring phylogenies. The fact that Clus-φ behaves differently than NJ shows that their results can be used to complement one another.

Chapter 4

Using top-down induced clustering trees for protein subfamily identification

In this chapter[1], we consider the divisive clustering approach we presented in the previous chapter in the context of protein subfamily identification.

4.1 Introduction

In protein subfamily identification, given a set of sequences that belong to one protein family, the goal is to identify subsets of functionally closely related sequences (called subfamilies). This is in essence a clustering task. Most current methods for subfamily identification use a bottom-up clustering method to construct a cluster hierarchy, then cut the hierarchy at the most appropriate locations to obtain a single partitioning. Such approaches rely on the assumption that functionally similar proteins have sequences with a high overall similarity, but do not exploit the fact that these sequences are likely to be highly conserved at particular positions. This raises the question to what extent clustering procedures can be improved by making them exploit this property.

[1] Based on the paper "Top-down clustering for protein subfamily identification" [Costa, Vens and Blockeel 2013].

We propose and evaluate a method that does exactly this. The proposed method uses the divisive clustering procedure described in Chapter 3 to build the cluster hierarchy, and then applies a post-pruning procedure to extract clusters from the hierarchy. This approach differs from bottom-up clustering methods in that it forms clusters whose elements do not only have high overall similarity, but also have particular properties in common. In the case of subfamily identification, these properties can be the amino acids found at particular positions.

Apart from possibly yielding higher quality clusterings, this approach has the advantage that it automatically identifies functionally important positions, and that new sequences can be classified into subfamilies by just checking those positions.

The remainder of this chapter is organized as follows. Section 4.2 describes the protein subfamily identification problem and discusses existing methods in this context. Section 4.3 presents the proposed method. Before experimentally comparing the new method with a state-of-the-art method, we review evaluation measures for subfamily identification in Section 4.4, leading to the introduction of two new measures. Section 4.5 presents an empirical evaluation of our method over 11 datasets. Our results show that: splits based on polymorphic positions are highly discriminative between protein subfamilies; using such splits to guide a clustering procedure improves protein subfamily identification; the identified positions yield accurate classification of new sequences; the resulting clustering tree identifies functionally important sites. We conclude in Section 4.6.

4.2 Background and related work

Whereas an entire protein family may be associated with many functions, on the subfamily level usually one or a small set of functions are retained. Therefore, being able to identify subgroups of functionally closely related sequences is an important aspect of protein function prediction.

Protein subfamilies can be identified by phylogenomic analysis [33], i.e., by using phylogenetic information. In particular, a hierarchical or phylogenetic tree is constructed over the entire family, starting from a multiple sequence alignment (MSA); then, subfamilies are extracted from the tree. Phylogenomic analysis avoids some of the problems associated with more traditional, homology based, approaches towards protein function prediction [20], such as database annotation error propagation [52].

Bayesian evolutionary tree estimation (Bête) [115] was the first algorithm to automatically decompose a protein family into subfamilies. The method was

later extended to include a module for classification of novel sequences into subfamilies, and is better known under the name SCI-PHY [20]. We discuss SCI-PHY in detail below, since it will be used in the experimental comparison in Section 4.5.

Secator [137] is another phylogenomic method that automatically predicts subfamilies. It first constructs a phylogenetic tree using the BIONJ algorithm [48]. The tree is then cut based on a sequence dissimilarity measure. The idea behind this post-pruning procedure is that tree nodes should be iteratively merged as long as the sequence dissimilarity within the resulting cluster is below an automatically calculated threshold.

Albayrak et al. [3] proposed a method that, in contrast to the aforementioned methods, does not start from an MSA. Instead, it uses a relative complexity measure (RCM) to construct a pairwise distance matrix, and constructs a NJ tree with it. The authors do not extract clusters from the tree, but rather evaluate the tree topology. The results were comparable to those given by NJ trees using MSA.

GeMMA [79] was designed to deal with large and diverse protein families, where an accurate global MSA becomes infeasible. It applies a bottom-up clustering procedure, and after each merging step recomputes an MSA for the sequences in the newly generated cluster. In order to decide which clusters to merge, a comparison score between alignments is computed. The scores are also thresholded to obtain a stopping criterion. The performance of GeMMA was shown to be comparable to SCI-PHY [79].

Whereas all mentioned methods build the hierarchical tree using bottom-up clustering, the method we propose performs top-down clustering. Top-down clustering approaches have rarely been used in biological applications. Varsharvsky et al. [130] point out three reasons for this: there is much more software available for bottom-up clustering; bottom-up clustering has an intuitive appeal; and it tends to generate topologies with high reliability at the more specific levels. On the other hand, Varsharvsky et al. [130] show that top-down clustering can be successfully applied to biological data.

Our idea of using conserved positions to define subfamilies is similar to the work of Bickel et al. [11]. Their main goal is to identify functionally related sites in an MSA. To that aim, they first enumerate all possible protein subfamilies that are defined by having at least two positions with subfamily-specific residues. Afterwards, they prune the list of potential functional sites and corresponding protein subfamilies by assessing the degree of association between these sites for each subfamily.

The main difference between the method proposed by Bickel et al. [11] and ours is

that our method constructs a tree (i.e., finds a complete and non-overlapping set of subfamilies), whereas Bickel et al. [11] discover each subfamily independently, possibly resulting in an incomplete or overlapping set of subfamilies.

An alternative strategy to the use of phylogenetic tree reconstruction for protein subfamily identification consists of clustering sequences based on pairwise sequence comparison to find clusters of homologs.[2] More specifically, most of the methods based on this approach look for orthologs. The idea behind this approach is that orthologous genes/proteins are likely to have the same/similar function(s). INPARANOID [106] is one method that is representative of this approach. This method first compares pairs of sequences originating from two different species, and then identify clusters of orthologs and in-paralogs. However, the input data expected by INPARANOID is different from the input data expected by our method. More specifically, (1) INPARANOID expects input sequences from only two species, while our method does not have constraints on the origin of the input sequences; (2) INPARANOID uses species information in its strategy (i.e., it needs to know from which species each sequence comes), while our method uses only sequence information; (3) INPARANOID expects to have the complete set of genes or proteins of the species being analyzed, while our method expects an alignment of similar[3] sequences. Another method that follows a similar approach as INPARANOID is OrthoMCL [23]. One of the main differences between INPARANOID and OrthoMCL is that the latter can consider multiple species.

As argued by the authors of SCI-PHY [20], the species information required by methods that find orthologs restricts their application to data where this information is available. In environmental sequence analysis [131], for example, this information might not be available.

4.2.1 SCI-PHY

To identify protein subfamilies, SCI-PHY [20] first builds a hierarchical tree bottom-up. It then extracts clusters from the tree, which are output as the predicted subfamilies.

The tree construction process starts with each sequence being a separate cluster. Then, for each cluster a profile is defined, which gives the expected amino acid

[2]Homologs are genes are that have a common ancestor in evolution. Homologs are divided in orthologs and paralogs. Orthologs are genes found in different species that originated from the same gene in the last common ancestor of those species [106]. Paralogs are genes resulted of gene duplication. When the gene duplication occurs before a given speciation event, paralogs are called out-paralogs; otherwise, they are called in-paralogs.

[3]The similarity of the sequences should allow an alignment of good quality.

probabilities for each position, based on the observed amino acid distribution and a Dirichlet mixture density [10]. Next, using relative entropy [76] to estimate the distance between the profiles, the two closest profiles are merged, and a profile for the new cluster is created. This merging procedure is repeated until all sequences are part of the same cluster. Finally, the resulting tree topology is given as input to a post-pruning procedure.

In this procedure, the best way to prune the tree is searched using a measure called encoding cost (see Section 4.3), which can be interpreted as the cost to encode a clustering given the number and homogeneity of the clusters. SCI-PHY returns the stage in the clustering procedure with minimal encoding cost.

Once the protein subfamilies have been predicted, SCI-PHY can classify new protein sequences into one of these subfamilies, by subfamily hidden Markov model (SHMM) construction [19]: a SHMM is built for each subfamily and the best match with the query sequence is predicted.

We use SCI-PHY in our experimental comparison (Section 4.5), because it has been extensively evaluated: it was compared to several methods, and was shown to give comparable or superior results [20] [79]. To our knowledge, no other method has been shown to give better results.

4.3 Proposed method

Sequences within a protein subfamily are not only similar to each other, they are also characterized by a small set of conserved amino acids at particular locations, which distinguishes them from sequences in other subfamilies. The method we propose exploits this property. It creates clusters in which sequences are not only globally similar, but, additionally, identical in particular locations. These locations are discovered by the clustering process as it goes.

Using the same top-down approach described in Chapter 3 (see pseudocode in Figure 3.5, Section 3.3), the method starts with a set of sequences, which is given as an MSA,[4] and tries to split it into subsets such that (1) sequences within a subset are similar, and (2) the split is defined by a test of the form $p = a$, or more generally $p \in S$, with p a location, a an amino acid, and S a set

[4]Our method assumes that the size and diversity of the sequences allow an alignment of good quality. If this does not hold (due to high subfamily diversity, for example), it might be the case that the conserved positions within subfamilies are not well aligned; as a result, our method might not be able to find the appropriate splits to identify the subfamilies. A possible solution is to reconstruct the alignment after each split in a similar way as the GeMMA algorithm [79], mentioned in Section 4.2. However, this complicates classification of new sequences and functional site analysis.

of amino acids. After dividing a set into two subsets, the same principle can be used to further subdivide the subsets, up to the level of singletons (subsets with only one sequence). This yields a hierarchical tree. For the purpose of subfamily identification, the tree is cut at particular locations and the resulting clusters are assumed to form subfamilies.

We propose the following pruning procedure (see pseudocode in Figure 4.1) to cut the tree. In a single pruning step, a pair of sibling leaves is pruned, turning their parent into a leaf. Which pair is pruned is determined by a pruning heuristic. This step is continued until the whole tree is reduced to a single node. Each tree encountered in the process defines a clustering, the leaves of the tree being the clusters. Among all the trees thus found, the one with the highest-quality clustering is returned as the final result.

```
PruneTree

    // Input: T: tree to be pruned
    // Output: T_B : tree with the highest quality encountered during pruning

    T_B := T
    While T has more than one node do
        For each pair of sibling leaves (l_i, l_j) do
            Tentatively prune the leaves
            Calculate the quality of the resulting tree
        Let T' be the highest-quality tree among all tentatively pruned versions of T
        If T' has a better quality than T_B then
            T_B := T'
        T := T'
```

Figure 4.1: Pseudocode for the proposed post-pruning procedure.

The resulting clusters, which correspond to the predicted protein subfamilies, are then output along with the underlying tree, which explicates how the clusters were split and which tests were used. We show an example of such a tree in Figure 4.2. Note that the internal nodes typically contain multiple tests. This indicates that there are equivalent tests for that stage of the clustering process; tests are equivalent when they yield the same outcome for all the sequences.

Apart from identifying subfamilies, the tree has two additional advantages. First, it allows for easy classification of new sequences into a subfamily. Starting at the root node, a new sequence is moved down the tree according to the outcome of the tests, until it is classified into one of the predicted subfamilies.

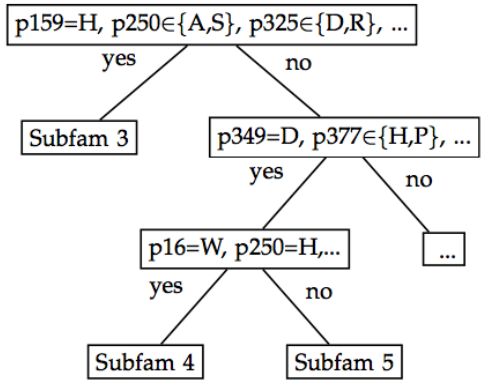


Figure 4.2: Example of a tree output by our method.

When not all tests in a node agree (which is impossible for the sequences used to build the tree, but may happen for other sequences), the majority decides.

Second, the identified tests result in a candidate list of functionally important sites, i.e., positions that are likely to play a role in the subfamily-specific functions. Protein functional site prediction is an important step in the functional analysis of new proteins (e.g. [11][24][16]). As biological validation is costly, providing a first selection of potential sites is an important advantage of our method.

Important parameters of the method are the heuristics used during tree growing (to select the best test to split the data at each step) and pruning (to evaluate the quality of a tree). We now discuss these in detail. We will end this section with a note on the computational complexity of the method.

Test selection heuristics

In the experimental section, we explore three test selection heuristics. Two of them are standard for hierarchical tree learners: maximization of the average inter-cluster distance, and maximization of the minimum inter-cluster distance. We call the versions of Clus that use these heuristics Clus-MaxAvgDist and Clus-MaxMinDist, respectively.

These heuristics, however, do not take into account the particular requirements of the phylogenomic context, which is based on phylogenetic analysis. Therefore,

we also consider the NJ-based heuristic we introduced in Section 3.3. Clus with this heuristic is named Clus-φ.

The proposed selection heuristics make use of distances between pairs of amino acid sequences. Our method computes distances based on the Jones-Taylor-Thornton matrix [69], which is a model of amino acid substitution widely used for phylogenetic inference. Alternatively, we allow the user to give a pairwise distance matrix as input.

Extracting clusters from the hierarchical tree

The quality measure used during pruning is encoding cost [116][20], which can be interpreted as the cost to encode a clustering given the homogeneity of the clusters and the number of clusters. Ideally, one wants to achieve two goals: to have as few clusters as possible, and to have maximally homogeneous clusters. There is a trade-off between these two goals, as having fewer clusters implies larger clusters, which are less likely to be homogeneous. The encoding cost (Equation 4.1) combines these two goals.

$$EncodingCost = N \log k - \sum_{l=1}^{k} \sum_{i} \log \Pr(n_{li}|\alpha) \tag{4.1}$$

The first component of the equation is the cost associated to the number of subfamilies; the second component is the cost to encode each subtree alignment for a certain clustering. More specifically, N is the number of sequences, k is the number of clusters, and $Pr(n_{li}|\alpha)$ is the probability of n_{li} - which is the count vector of observed amino acids for subfamily l at column i, under a Dirichlet mixture density α. Dirichlet mixture densities [10] contain prior information about amino acids and, when combined with observed amino acid frequencies, provide estimates of expected amino acid probabilities [116].

Computational complexity

The computational complexity of the proposed method is $O(aN^2 \log N)$, with a the alignment length and N the number of sequences. We obtain this complexity by adding the complexity of the tree building and post-pruning procedures, as described next.

The complexity to construct the tree is $O(aN^2 \log N)$, if we consider that the NJ-based heuristic is used (see Section 3.3 for detailed discussion). In order to extract subfamilies from the tree, every pruning step and every merging candidate requires calculating Equation 4.1. In two subsequent calculations,

most of the clusters remain the same, and therefore most of the $\sum_i \log \Pr(n_{li}|\alpha)$ terms do not change. We can avoid recomputing these values by calculating them only once for every node (cluster), and storing them. As a (complete) tree with N sequences contains $2N - 1$ nodes, and the computation of Equation 4.1 has a complexity $O(aN)$, the resulting complexity of the cluster extraction is $O(aN^2)$, leading to an overall complexity of $O(aN^2 \log N)$ for the complete method.

4.4 Evaluation measures

In Section 4.5 we evaluate the subfamilies output by Clus and SCI-PHY on a number of datasets, for which the true subfamilies (reference clustering) are known. We evaluate both the tree topology from which clusters are extracted, and the clusters themselves. The reason for evaluating also the tree topology is threefold. First, as the authors of SCI-PHY also point out, the definition of the "right" clusters is somewhat arbitrary, since subfamilies can be defined on several levels of granularity. By evaluating the tree topology, which defines clusterings on multiple levels, we can analyze how the reference clustering is represented in the tree. Second, we can evaluate the results regardless of the quality of the pruning procedure. Third, the tree is often interesting in itself, as it can help biologists to interpret the predicted clustering, and obtain insights in how the clusters are related.

4.4.1 Tree topology evaluation

We evaluate tree topologies using two measures we propose - edited tree size and number of subfamily changes, and an existing one - tree-based classification error [78].

Edited tree size

The edited tree size indicates how compact the smallest possible pure clustering, derived from the tree, is. It is calculated by repeatedly merging sibling leaves that belong to the same subfamily until such merging is no longer possible.

Consider the two trees shown in Figure 4.3, for example. They have five sequences, three of which belong to subfamily 1 ($S1$), and two of which belong to subfamily 2 ($S2$). The edited tree for tree a would merge the $S2$ sequences, resulting in an edited tree size of 4. The edited tree for tree b would merge the

two $S1$ sequences connected by branches r_7 and r_8, also resulting in an edited tree size of 4.

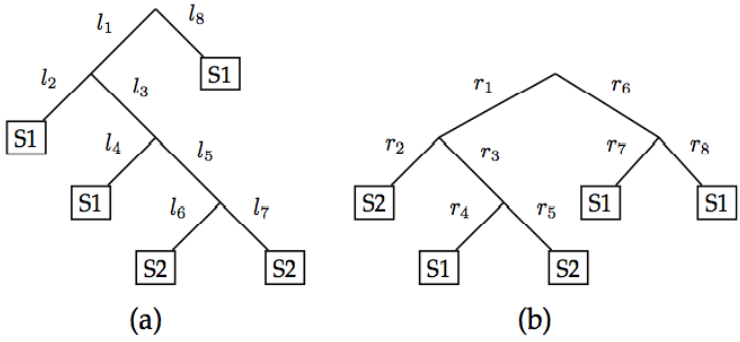


Figure 4.3: Two trees with the same edited tree size, a smaller TBC error for tree a, and a smaller number of subfamily changes for tree b. Branches are labeled to make the explanation easier.

Tree-based classification error

Similarly to the edited tree size, the tree-based classification (TBC) error [78] evaluates to which extent the tree places sequences from the same subfamily in the same subtree. While the former considers clusters that are pure and as large as possible, the TBC error considers clusters that minimize the number of "classification errors" in the derived clustering, as follows.

A subtree is said to be "good" for a subfamily F if more than half of its sequences belong to F, and more than half of F's sequences belong to the subtree. Given a set of disjoint good subtrees, a sequence is considered correctly classified if it occurs in a good subtree for its subfamily, and incorrectly classified otherwise. Moreover, a sequence that does not occur in any good subtree is also considered a misclassified sequence. The TBC error is defined as the smallest number of incorrectly classified sequences in any set of disjoint good subtrees.

Tree a in Figure 4.3 defines two good subtrees: a cut in branch l_5 yields a good subtree for $S2$, and the complete tree is a good subtree for $S1$. As these subtrees are not disjoint, each one of them defines a set of disjoint subtrees containing a single subtree. The set defined by the subtree obtained by the cut of l_5 results in three classification errors: the three sequences $S1$ are not classified in any good subtree. The set defined by the complete tree results in two classification

errors: the two sequences $S2$ are classified in a good subtree for $S1$. As this set gives the smallest number of incorrectly classified sequences, the TBC error for this tree is 2. Tree b defines three good subtrees. A cut in r_1 (or r_6) yields two disjoint good subtrees: a good subtree for $S2$ at the left, and a good subtree for $S1$ at the right. This set of subtrees results in one classification error: one sequence $S1$ is classified in the good subtree for $S2$ (subtree at the left). The complete tree is again a good subtree for $S1$, with two classification errors: the two sequences $S2$ are classified in a good subtree for $S1$. Hence, the TBC error for this tree is 1. Lazareva-Ulitsky et al. [78] provide a algorithm to compute the TBC error.

The subtrees defined by the TBC error are more permissive than the ones defined by the edited tree size in the sense that clusters are not required to be pure; on the other hand, TBC error is stricter in the case where sequences from the same subfamily are spread over two or more subtrees.

Both measures are dependent on the place of the tree root. If the root of tree a would be in branch l_5, for example, the edited tree size would be 2, instead of 4; and the TBC error would be 0, instead of 2. Although evaluating the rooted tree is important, because the root influences the possible ways in which the tree can be cut, we also propose a measure that is independent of the place of the root.

Number of subfamily changes

If we associate a subfamily (or alternatively, a molecular function) to each internal node of the tree, then we say that a subfamily change occurs for each branch connecting two nodes with different associated subfamilies. For instance, if we associate subfamily 1 to the root of tree a in Figure 4.3, there is one change to subfamily 2 in branch l_5. The right tree, however, requires two subfamily changes (branches r_2 and r_5).

Counting the minimal number of subfamily changes corresponds to counting mutations in parsimony analysis [123], where one prefers the phylogenetic tree that requires the least evolutionary change to explain some observed data. Therefore, we can directly apply the Fitch parsimony algorithm [42] to count the number of subfamily changes.

Note that, in contrast to the previous two measures, this measure does not penalize a tree for having a ladder-like shape. That is why tree a has a smaller number of subfamily changes, while having the same edited tree size and a

higher TBC error as tree b.[5] However, it important to note that the shape of the tree does influence the possible ways to cut it. Therefore, we use the three measures, as they provide complementary information to one another.

4.4.2 Clustering evaluation

We evaluate clusters using three measures earlier used for SCI-PHY [20] (purity, edit distance, and variation of information distance) and two additional ones: the percentage of sequences in pure clusters, and category utility. The first four measures compare the predicted clustering to the reference clustering, while the latter evaluates the quality of the predicted clustering itself, regardless of a given reference clustering.

We have introduced category utility in Section 2.2.3. The other four measures are described next.

Purity

Purity is defined as the fraction of clusters in a given clustering that contain instances of only one reference cluster. It assesses the ability of the method to cluster instances of different kinds in different clusters. However, as it does not penalize if instances of one kind are spread over many pure clusters, perfect purity can be achieved when every instance corresponds to a single cluster. Therefore, singletons are not included in the calculation.

Percentage of examples in pure clusters

To complement the information given by purity, we also report the percentage of examples in pure clusters (denoted further as PctExPureClusters). Again, singleton clusters are discarded.

Edit distance

The edit distance between two clusterings is the number of merge and/or split operations required to transform one clustering into the other one. For example,

[5] For an example with larger trees, consider Figures B.1 and B.2, in Appendix B. The tree in Figure B.1 has an edited tree size of 12, a TBC error of 11, and 9 subfamily changes; while the tree in Figure B.2 has an edited tree size of 28, a TBC error of 64, and 11 subfamily changes. The larger difference between the trees in their edited tree size and TBC error is due to the ladder-like shape of the tree in Figure B.2.

if instances of three kinds - A, B, and C - are clustered in only one cluster, we need two split operations to separate the three groups of instances. The higher the edit distance is, the more different the clusterings are.

The formal definition of edit distance is given by Equation 4.2 [20], where $Edit(C^1, C^2)$ is the edit distance to transform clustering C^1 into clustering C^2 (or the other way around), k' is the number of clusters in C^1, k'' is the number of clusters in C^2, and $r(C^1_m, C^2_n)$ is equal to 1 if clusters C^1_m and C^2_n have items in common, and equal to 0 otherwise.

$$Edit(C^1, C^2) = 2 * (\sum_{m=1}^{k'} \sum_{n=1}^{k''} r(C^1_m, C^2_n)) - k' - k'' \tag{4.2}$$

Edit distance penalizes more strongly clusterings for which clusters are too small. For this reason, this measure can be used to counter-balance purity.

Variation of information distance distance

The variation of information distance (VI distance) measures the amount of information that is not shared between two clusterings. The formula to calculate the VI distance is given by Equation 4.3 [20], where $H(C^1)$ (Equation 4.4) is the entropy of clustering C^1, and $I(C^1, C^2)$ (Equation 4.5) is the mutual information between clusterings C^1 and C^2. In Equation 4.4, $|C_l|$ is the number of instances in cluster C_l, $|C|$ is the total number of instances in the clustering, and k is the number of clusters in C.

$$VI(C^1, C^2) = H(C^1) + H(C^2) - 2 * I(C^1, C^2) \tag{4.3}$$

$$H(C) = \sum_{l=1}^{k} \frac{|C_l|}{|C|} \log \frac{|C_l|}{|C|} \tag{4.4}$$

$$I(C^1, C^2) = \sum_{m=1}^{k'} \sum_{n=1}^{k''} \frac{|C^1_m \cap C^2_n|}{|C|} \log \frac{|C^1_m \cap C^2_n|}{|C|} \tag{4.5}$$

In Equation 4.5, $|C^1_m \cap C^2_n|$ is the number of overlapping instances between clusters C^1_m and C^2_n, k' and k'' are the number of clusters in C^1 and C^2, respectively.

4.5 Empirical evaluation

We empirically evaluate, first, the soundness of the assumptions underlying our method, and second, the method's capacity to respectively propose a meaningful tree topology, identify subfamilies, classify new sequences, and identify functional sites.

4.5.1 Datasets

We use two groups of datasets. The first group consists of the five EXPERT datasets used by Brown et al. [20] to evaluate SCI-PHY. The second group consists of six datasets extracted from NucleaRDB [61], which contains protein data for Nuclear Hormone Receptor (NHR) families. These eleven datasets were chosen because for each of them a reliable subfamily identification is provided for every sequence, which gives us a gold standard to evaluate the results.

Each dataset consists of the MSA for one protein family. The EXPERT datasets contain sequences from the families Enolase, Crotonase, Secretin, Aminergic (Amine), and NHR. The NucleaRDB datasets contain sequences from the families thyroid hormone like (Thyroid), estrogen like (Estrogen), nerve growth factor IB-like (Nerve), HNF4-like (HNF4), fushi tarazu-F1 like (Fushi), and DAX like (DAX). To construct the NucleaRDB datasets, we used MSAs for each family as provided by NucleaRDB, with replicate sequences removed.

For Amine, NHR, Thyroid, Estrogen, and HNF4, subfamilies are provided at more than one level of granularity. Thus, two sequences can be associated to the same subfamily x in one dataset, but to different subfamilies $x.1$ and $x.2$ in the other dataset.

Some of the datasets are very unbalanced, complicating the subfamily identification task. For instance, Enolase contains a subfamily consisting of 60% of the sequences. The number of sequences in the subfamilies Crotonase and NHR1 ranges from 1 to 212 (=58%) and 139 (=34%), respectively.

In Table 4.1 we report, for each dataset, the number of subfamilies, the number of sequences, the MSA length, the average pairwise distance between all sequences within the family, and the overall average distance within the subfamilies[6]. We calculated the sequence distances based on the Jones-Taylor-Thornton model.

[6]We first calculate the average pairwise distance for each subfamily, and then we report the average value over all subfamilies.

Table 4.1: Protein subfamily datasets,

Datasets	Nb subfam	Nb seq	Align length	Avg dist (family)	Avg dist (subfam)
Enolase	8	472	431	2.229	1.041
Crotonase	10	365	264	1.842	0.728
Secretin	15	153	263	1.885	0.485
Amine 1	7	358	344	1.467	1.075
Amine 2	31	358	344	1.467	0.442
NHR 1	8	412	183	2.124	0.945
NHR 2	27	412	183	2.124	0.547
NHR 3	77	409	183	2.116	0.263
Thyroid 1	8	799	239	1.771	0.708
Thyroid 2	24	799	239	1.771	0.375
Estrogen 1	3	482	226	1.041	0.498
Estrogen 2	10	482	226	1.041	0.301
HNF4 1	5	448	229	1.276	0.619
HNF4 2	22	448	229	1.276	0.404
Nerve	5	76	219	0.429	0.260
Fushi	4	117	227	0.756	0.369
DAX	2	40	133	0.867	0.397

4.5.2 Testing the usability of polymorphic positions for clustering protein subfamilies

In this experiment, we verify our assumption that splits based on polymorphic positions can indeed discriminate protein subfamilies. To that aim, we add the subfamily information to the data and build a classification tree using Clus (i.e., we performed supervised learning), without pruning, i.e., up to the point where all leaves are pure. Table 4.2 shows the number of leaves in the resulting tree for each dataset.

The results show that subfamilies can be perfectly separated from one another using compact trees containing slightly more leaves than the number of subfamilies in the datasets. For five of the datasets - Enolase, Secretin, Nerve, Fushi and DAX - the classification tree has the same number of leaves as the number of subfamilies. From this we conclude that polymorphic positions are indeed highly discriminant for protein subfamily identification.

The fact that a good clustering tree exists does not imply it will be found by our learner. The above trees are built with the subfamily information, but in a

Table 4.2: Number of leaves in the classification trees (CTs).

Datasets	Nb subfam	Nb leaves	Datasets	Nb subfam	Nb leaves
Enolase	8	8	Thyroid 1	8	13
Crotonase	10	11	Thyroid 2	24	38
Secretin	15	15	Estrogen 1	3	4
Amine 1	7	14	Estrogen 2	10	15
Amine 2	31	34	HNF4 1	5	7
NHR 1	8	11	HNF4 2	22	36
NHR 2	27	30	Nerve	5	5
NHR 3	77	79	Fushi	4	4
			DAX	2	2

real situation, this will not be the case. In the next sections we evaluate our unsupervised learning method.

4.5.3 Evaluating the tree topology

We first compare the three variants of our method (Clus-φ, Clus-MaxAvgDist and Clus-MaxMinDist) on the EXPERT datasets. The results are shown in Tables 4.3, 4.4 and 4.5. We indicate the best results per row in boldface. As the results show a better performance for Clus-φ, the version adapted to phylogenetic data, we focus on this version for the remainder of the chapter.

Table 4.3: Edited tree size - choosing the test selection criterion.

Datasets	Clus-φ	Clus-MaxAvgDist	Clus-MaxMinDist
Enolase	**12**	51	25
Crotonase	33	111	**25**
Secretin	**19**	32	21
Amine 1	**30**	54	33
Amine 2	49	75	**48**
NHR 1	**22**	49	36
NHR 2	43	68	**41**
NHR 3	**90**	139	107

We now evaluate the tree topologies produced by Clus-φ in comparison to SCI-PHY, for all datasets. For the sake of completeness, we include the phylogenetic tree produced by the neighbor joining (NJ) algorithm [111] in the comparison. To report the edited tree size and TBC error for NJ, we have to define a tree

Table 4.4: TBC error - choosing the test selection criterion.

Datasets	Clus-φ	Clus-MaxAvgDist	Clus-MaxMinDist
Enolase	**11**	90	64
Crotonase	41	99	**27**
Secretin	**8**	30	11
Amine 1	178	**161**	217
Amine 2	**41**	80	56
NHR 1	**36**	105	257
NHR 2	**24**	80	38
NHR 3	**42**	108	70

Table 4.5: Number of subfamily changes - choosing the test selection criterion.

Datasets	Clus-φ	Clus-MaxAvgDist	Clus-MaxMinDist
Enolase	**9**	14	**9**
Crotonase	11	37	**9**
Secretin	15	15	**14**
Amine 1	**16**	26	19
Amine 2	**37**	49	38
NHR 1	**10**	24	16
NHR 2	**29**	48	32
NHR 3	**78**	97	83

root - since NJ trees are unrooted. The most direct way to do that is to root the tree according to the order in which the sequences were merged by the NJ algorithm. As NJ merges three subtrees in its last step, we root the tree according to each of these subtrees and report the average results. Tables 4.6, 4.7, and 4.8 show the edited tree size, the TBC error, and the number of protein subfamily changes, respectively. For illustration, the (edited) tree topologies for dataset Enolase for the three algorithms are shown in Figures B.1, B.2, and B.3, in Appendix B.

Throughout the chapter, we report the results of pairwise comparisons as wins/ties/losses. An asterisk (*) indicates that a two-tailed sign test rejects the hypothesis that the methods being compared present equally good results, according to the evaluation measure into consideration, at significance at 5%.

In terms of edited tree size (Table 4.6), Clus-φ obtains 13/2/2* wins/ties/losses in comparison to SCI-PHY, and 12/0/5 wins/ties/losses in comparison to NJ. The edited tree size for Clus-φ, SCI-PHY, and NJ is, on average, 2.6, 3.3, and

Table 4.6: Edited tree size - evaluating the Clus-φ topologies.

Datasets	Clus-φ	SCI-PHY	NJ	Datasets	Clus-φ	SCI-PHY	NJ
Enolase	**12**	28	37.7	Thyroid 1	**34**	28	34.7
Crotonase	33	70	**26**	Thyroid 2	**74**	86	103.7
Secretin	19	22	**18.7**	Estrogen 1	**10**	13	19.7
Amine 1	30	36	**29.7**	Estrogen 2	**36**	44	52.7
Amine 2	**49**	52	54.7	HNF4 1	**14**	21	29.7
NHR 1	22	30	**21.7**	HNF4 2	**83**	111	136.7
NHR 2	43	38	**38.7**	Nerve	**10**	**10**	23.7
NHR 3	**90**	105	104.7	Fushi	**10**	11	16.7
				DAX	**3**	**3**	8.7

Table 4.7: TBC error - evaluating the Clus-φ topologies.

Datasets	Clus-φ	SCI-PHY	NJ	Datasets	Clus-φ	SCI-PHY	NJ
Enolase	**11**	64	189	Thyroid 1	**17**	117	55
Crotonase	**41**	50	137.3	Thyroid 2	**54**	130	116
Secretin	**8**	13	12.3	Estrogen 1	**2**	12	234
Amine 1	178	242	**164**	Estrogen 2	**27**	44	161
Amine 2	**41**	96	59	HNF4 1	**4**	67	152
NHR 1	**36**	269	133	HNF4 2	**89**	202	161
NHR 2	**24**	34	85	Nerve	**11**	**11**	28
NHR 3	**42**	58	79	Fushi	**24**	29	53
				DAX	**3**	4	7

3.9 larger than the number of subfamilies.

Table 4.7 shows that Clus-φ obtains a smaller TBC error than NJ for all cases but one case (Amine 1, for which NJ has a smaller TBC error)*, and a smaller TBC error than SCI-PHY for all but one case (Nerve, for which there is a tie)*. On average, the Clus-φ tree has a TBC error 48.5% smaller than the SCI-PHY tree, and 60.7% smaller than the NJ tree.

Regarding the number of protein subfamily changes (Table 4.8), Clus-φ obtains 13/4/0* wins/ties/losses in comparison to SCI-PHY, and 4/2/9 wins/ties/losses in comparison to NJ. On average, the numbers of subfamily changes for Clus-φ, SCI-PHY, and NJ are, respectively, 1.6, 2.0, and 1.5 times larger than the minimum possible number of changes.

Summarizing, Clus-φ outperforms both other systems in terms of edited tree size and TBC error, but outperforms only SCI-PHY, not NJ, in terms of subfamily

Table 4.8: Number of subfamily changes - evaluating the Clus-φ topologies.

Datasets	Clus-φ	SCI-PHY	NJ	Datasets	Clus-φ	SCI-PHY	NJ
Enolase	9	11	**8**	Thyroid 1	16	17	**9**
Crotonase	11	16	**9**	Thyroid 2	**39**	58	42
Secretin	15	15	**14**	Estrogen 1	**4**	6	5
Amine 1	16	22	**12**	Estrogen 2	**19**	27	20
Amine 2	37	42	**35**	HNF4 1	**6**	11	**6**
NHR 1	10	12	**7**	HNF4 2	**50**	67	51
NHR 2	29	29	**26**	Nerve	**5**	5	**5**
NHR 3	78	82	**77**	Fushi	**5**	6	6
				DAX	**1**	**1**	**1**

changes. As the number of subfamily changes does not depend on the position of the root, this can be taken as confirmation that NJ is good at creating evolutionary trees (unsurprisingly), but does not aim at creating rooted trees from which subfamilies can easily be extracted. Visual inspection of the trees (see Figures B.1, B.2, and B.3 in Appendix B) further reveals that NJ and SCI-PHY tend to produce trees with a more ladder-like shape, while Clus-φ produces more symmetrical trees. Ladder-like trees usually result in clusterings with over-splitting of subfamilies and/or large impure clusters.

These results together show that Clus-φ has the potential to yield good subfamilies, provided that an adequate pruning criterion is used.

4.5.4 Evaluating the cluster predictions

In this section, we evaluate the cluster predictions given by Clus-φ using the post-pruning method based on encoding cost discussed in Section 4.3; we call this variant of our method Clus-φ-ECC. For illustration, we show the Clus-φ-ECC and SCI-PHY trees for Enolase in Figures B.4 and B.5 (Appendix B), respectively. For this evaluation, we use the measures described in Section 4.4.2. The results for the EXPERT and NucleaRDB datasets are shown in Tables 4.9 and 4.10, respectively. Note that in the rows corresponding to purity we also display, in parentheses, the number of pure non-singletons followed by the total number of non-singletons. This gives some additional purity information. Further, as the number of clusters is also given, we can obtain the number of singletons present in the clustering.

The results show that, in general, both Clus-φ-ECC and SCI-PHY present a very good purity. In comparison to SCI-PHY, Clus-φ-ECC obtains 4/1/3

Table 4.9: Evaluation of the clustering predictions for the EXPERT datasets.

Datasets	Eval. Measure	Clus-φ-ECC	SCI-PHY
Enolase	Purity	0.97 (30/31)	**1.00 (26/26)**
	PctExPureC	**0.951**	0.890
	VI Distance	1.676	**1.374**
	Edit Distance	**42**	70
	# Clusters	48	78
Crotonase	Purity	**0.97 (31/32)**	0.94 (15/16)
	PctExPureC	**0.729**	0.521
	VI Distance	1.580	**1.048**
	Edit Distance	**29**	32
	# Clusters	37	38
Secretin	Purity	**0.89 (17/19)**	0.88 (14/16)
	PctExPureC	**0.758**	0.673
	VI Distance	**0.467**	0.565
	Edit Distance	**12**	15
	# Clusters	21	22
Amine 1	Purity	0.96 (45/47)	**0.97 (36/37)**
	PctExPureC	**0.966**	0.950
	VI Distance	1.870	**1.548**
	Edit Distance	46	**38**
	# Clusters	49	43
Amine 2	Purity	**0.87 (41/47)**	0.86 (32/37)
	PctExPureC	**0.852**	0.701
	VI Distance	**0.831**	0.898
	Edit Distance	38	**36**
	# Clusters	49	43
NHR 1	Purity	**1.00 (40/40)**	**1.00 (29/29)**
	PctExPureC	**0.981**	0.959
	VI Distance	1.984	**1.620**
	Edit Distance	40	**38**
	# Clusters	48	46
NHR 2	Purity	0.95 (38/40)	**0.97 (28/29)**
	PctExPureC	**0.954**	0.932
	VI Distance	0.708	**0.357**
	Edit Distance	25	**21**
	# Clusters	48	46
NHR 3	Purity	**0.62 (25/39)**	0.38 (11/29)
	PctExPureC	**0.518**	0.152
	VI Distance	**0.610**	0.949
	Edit Distance	**44**	54
	# Clusters	45	43

Table 4.10: Evaluation of the clustering predictions for the NucleaRDB datasets.

Datasets	Eval. Measure	Clus-φ-ECC	SCI-PHY
Thyroid 1	Purity	0.80 (28/35)	**1.00 (31/31)**
	PctExPureC	0.937	**0.974**
	VI Distance	1.443	**1.225**
	Edit Distance	**42**	44
	# Clusters	36	52
Thyroid 2	Purity	0.63 (22/35)	**0.77 (24/31)**
	PctExPureC	**0.655**	0.645
	VI Distance	0.691	**0.647**
	Edit Distance	**47**	**47**
	# Clusters	36	52
Estrogen 1	Purity	**1.00 (19/19)**	**1.00 (15/15)**
	PctExPureC	**0.994**	0.967
	VI Distance	1.624	**1.232**
	Edit Distance	**19**	28
	# Clusters	22	31
Estrogen 2	Purity	**0.79 (15/19)**	0.73 (11/15)
	PctExPureC	**0.666**	0.521
	VI Distance	0.835	**0.552**
	Edit Distance	**24**	33
	# Clusters	22	41
HNF4 1	Purity	0.93 (27/29)	**1.00 (19/19)**
	PctExPureC	**0.971**	0.951
	VI Distance	1.471	**1.049**
	Edit Distance	**32**	36
	# Clusters	33	41
HNF4 2	Purity	**0.59 (17/29)**	0.47 (9/19)
	PctExPureC	**0.422**	0.156
	VI Distance	**1.086**	1.249
	Edit Distance	**53**	55
	# Clusters	33	41
Nerve	Purity	0.25 (1/4)	**0.60 (3/5)**
	PctExPureC	0.224	**0.263**
	VI Distance	**0.541**	0.600
	Edit Distance	**5**	7
	# Clusters	4	8
Fushi	Purity	**0.875 (7/8)**	0.667 (4/6)
	PctExPureC	**0.949**	0.367
	VI Distance	**0.583**	0.774
	Edit Distance	**6**	**6**
	# Clusters	8	6
DAX	Purity	**1.00 (4/4)**	**1.00 (4/4)**
	PctExPureC	**1.000**	**1.000**
	VI Distance	**0.608**	0.633
	Edit Distance	**2**	**2**
	# Clusters	4	4

wins/ties/losses for the EXPERT datasets; and 3/2/4 for the NucleaRDB datasets. Even though these results are quite comparable, an argument in favor of Clus-φ-ECC is that it generally achieves a higher percentage of examples in pure clusters; for only two cases - Thyroid 1 and Nerve - the SCI-PHY clustering has a larger value for this measure. The only dataset for which Clus-φ-ECC presents a very poor purity is Nerve. However, the same dataset seems to present a difficult task for SCI-PHY as well, which is witnessed by the small number of examples in pure clusters for the SCI-PHY results. Additionally, SCI-PHY presents a poor purity for datasets NHR 3 and Fushi, which contrasts with the considerably better results obtained by Clus-φ-ECC for the same datasets.

Regarding the edit distance, Clus-φ-ECC obtains 4/0/4 win/ties/losses for the EXPERT datasets, and 7/2/0* wins/ties/losses for the NucleaRDB datasets. The superior performance of Clus-φ-ECC in terms of edit distance for most the NucleaRDB datasets reflects, in part, the large number of singletons in the SCI-PHY clustering for those datasets. As stated by the authors of SCI-PHY [20], the edit distance penalizes over-division of subfamilies proportionally more than joining a few subfamilies into large clusters. For the VI distance, the results are mixed: Clus-φ-ECC obtains 3/0/5 wins/ties/losses, in comparison to SCI-PHY, for the EXPERT datasets; and 4/0/5 for the NucleaRDB datasets.

These results show that Clus-φ-ECC predicts clusters of at least equal quality as SCI-PHY, but with fewer singleton clusters and more instances in pure clusters.

The measures in Tables 4.9 and 4.10 depend on the reference clustering, the choice of which is somewhat arbitrary. Table 4.11 reports the category utility of the clusterings, which is independent of this. Again, the results favor Clus-φ-ECC (9 wins, versus 3 for SCI-PHY).

4.5.5 Evaluating the classification performance

In this section, we evaluate the ability of the tree generated by Clus-φ-ECC to classify new sequences into one of the predicted protein subfamilies. For this evaluation, we divided each dataset into ten subsets, keeping the original subfamily distribution for all subsets. Then, we performed a cross-validation procedure where, at each iteration, (1) nine subsets were used to identify protein subfamilies, and (2) the resulting tree was used to classify the sequences from the remaining subset; each subset was used exactly once to test the classification performance of the tree. To evaluate the correctness of each prediction, we verify if the actual subfamily of the sequence corresponds to the majority subfamily in the leaf node responsible for the prediction. We then report the accuracy of

Table 4.11: Category utility results. As NHR 3 has less sequences than NHR 1 and NHR 2, its clustering differs from the ones obtained for the latter. For this reason, NHR 3 is displayed on a separate row.

Datasets	Clus-φ-ECC	SCI-PHY
Enolase	**3.597**	2.229
Crotonase	**2.225**	1.797
Secretin	**5.640**	5.177
Amine 1 / 2	3.138	**3.198**
NHR 1 / 2	**2.029**	1.919
NHR 3	**2.154**	2.040
Thyroid 1 / 2	**3.117**	2.150
Estrogen 1 / 2	**3.838**	2.506
HNF4 1 / 2	**2.704**	2.131
Nerve	**6.789**	4.363
Fushi	7.687	**8.874**
DAX	8.345	**8.465**

the predictions, i.e., the fraction of the sequences whose subfamily membership was correctly predicted.

We compare these results with those using profile hidden Markov model (HMM) construction [32] on the clustering given by SCI-PHY;[7] we denote it SCI-PHY+HMM. We built the profile HMMs using the tool HMMER[8] version 3.0; and, for each query sequence, we predicted the subfamily for which the profile HMM best matched the sequence. The results are obtained using the cross-validation procedure described in the previous paragraph, with the same subsets.

The first two columns of Table 4.12 show the results for these two classification strategies, in terms of average accuracy over the ten subsets. We can observe that both strategies generally produce a high accuracy. When we compare their results, SCI-PHY+HMM obtains a larger number of wins: 11/0/6 wins/ties/losses. On the other hand, we can see that their accuracy values are comparable for most datasets; for two datasets (HNF4 2 and Fushi) we observe a large accuracy difference, both times in favor of the Clus-φ-ECC tree. A two-sided Wilcoxon signed-rank test does not indicate any difference (p-value of 0.88).

[7] These results are not necessarily equivalent to those which would be produced by the classification module of SCI-PHY, since the latter uses profile SHMMs [19] instead of profile HMMs. The classification module of SCI-PHY was not used because it is no longer supported.

[8] http://hmmer.org

With these results we can conclude that the tests identified by our method provide an accurate way to classify new sequences, with the advantage that no extra computation is required.

Note that the previous results are obtained from different clusterings. To evaluate the performance independent of the clustering, we compared the predictions of Clus-φ-ECC tree with those of profile HMMs built on the Clus-φ-ECC clustering; we denote it Clus-φ-ECC+HMM. The results are shown in the last column of Table 4.12.

The results show that, for 15 out of 17 cases, using the Clus-φ-ECC tree produces slightly less accurate results than Clus-φ-ECC+HMM. These results are not entirely unexpected, since profile HMMs use much more information than the Clus-φ-ECC tree in the classification process; the former uses information about every position of the MSA to perform the classification, while the latter uses only a set of tests on certain positions. The fact that the tree's classification performance approaches that of profile HMMs shows that the tests identified by the tree capture most of the information needed for classification, but not all of it.

Finally, we compare the classification results of Clus-φ-ECC+HMM with those of SCI-PHY+HMM. The former obtains 9/3/4 wins/ties/losses in comparison with the latter. A two-sided Wilcoxon signed-rank test gives a p-value of 0.02, indicating that Clus-φ-ECC+HMM performs statistically significantly better than SCI-PHY+HMM.

4.5.6 Analyzing the identified positions

Our method identifies at which positions in the alignment the predicted clusters differ. To gain more insight in these identified positions, we took one dataset - Enolase - and examined its tree in detail. In particular, for all known annotations of a certain kind, we check whether they occur in the tree. Part of the Enolase tree was depicted in Figure 4.2 of Section 4.3 to illustrate the output of our method. For ease of notation, we abbreviate the Enolase subfamilies, as shown in Table 4.13.

We queried all Enolase sequences in Uniprot [8] and retrieved all sequence positions with *active site* annotations. Mapping those to the sequence alignment resulted in a list of six positions. For four of them, the annotations are restricted to one or two subfamilies. We discuss whether and where they occur in the Clus-φ-ECC tree. For the other two positions, the annotations are found in sequences of various subfamilies, making the discussion more difficult.

Table 4.12: Accuracy of the protein classification results given by Clus-φ-ECC tree, profile HMMs built on the Clus-φ-ECC clustering, and profile HMMs profiles built on the SCI-PHY clustering.

Datasets	Clus-φ-ECC (tree)	SCI-PHY + HMM	Clus-MinLth-ECC + HMM
Enolase	0.987	**0.994**	0.985
Crotonase	0.975	**0.995**	0.989
Secretin	**0.948**	0.882	0.948
Amine 1	0.969	**0.975**	0.989
Amine 2	**0.894**	0.846	0.908
NHR 1	0.973	**0.998**	0.998
NHR 2	0.932	**0.985**	0.976
NHR 3	0.628	**0.633**	0.660
Thyroid 1	0.965	**0.996**	0.991
Thyroid 2	**0.842**	0.812	0.860
Estrogen 1	0.994	**0.996**	0.996
Estrogen 2	**0.917**	0.890	0.936
HNF4 1	0.980	**0.998**	0.998
HNF4 2	**0.672**	0.511	0.652
Nerve	0.766	**0.829**	0.791
Fushi	**0.974**	0.846	0.983
DAX	0.975	**1.000**	1.000

Table 4.13: Enolase subfamily definitions.

Subfamily 1	chloromuconate cycloisomerase
Subfamily 2	dipeptide epimerase
Subfamily 3	enolase
Subfamily 4	galactonate dehydratase
Subfamily 5	glucarate dehydratase
Subfamily 6	methylaspartate ammonia-lyase
Subfamily 7	muconate cycloisomerase
Subfamily 8	o-succinylbenzoate synthase

Position 159 (H) is annotated as an active site for one sequence, which belongs to subfamily 3. It is one of the seven equivalent tests that occur in the root node of our tree; the root splits subfamily 3 (consisting of 283 of the 472 sequences) from the rest of the family. Looking at the Uniprot annotations, we observed that position 159 actually is annotated as a *binding site* in 176 sequences of subfamily

3 (and does not have any annotations in sequences from other subfamilies).

Position 211 (E) is annotated as an active site for 176 sequences of subfamily 3 (the same sequences where 159 (H) is a binding site). Surprisingly, it does not occur in our tree, although this mutation is present for each sequence in subfamily 3. It turns out that this mutation is also present in one sequence of subfamily 8, which explains why it is not present in the root node that separates subfamily 3 from the rest.

Position 250 (H) is an active site for six members of subfamily 4. Interestingly, in the tree, it occurs in the node that splits off subfamily 4, indicating that it could be an active site for all 22 members of this subfamily. It also occurs in a node that splits off one sequence of subfamily 8.

Position 377 (H), finally, is annotated as an active site for 6 members of subfamily 4, and 3 members of subfamily 5. Among the 5 tests in the node that splits off all members of subfamilies 4 and 5, our tree contains a test "p377 $\in$ {H,P}".

Of course, one should also take into account the number of (equivalent) tests at the tree nodes. The more tests are present, the more likely it is that a particular annotation will be among them. Figure 4.4 shows the positions in the alignment that appear at the first four levels of the tree; each line corresponds to one node, and each node is numbered as indicated in the tree in Figure B.4 (Appendix B). The number of tests is generally small, ranging from 1 to 12, except for two nodes, which contain much more tests. The node corresponding to line 12 in the figure splits off one sequence from a set of 23 sequences, and hence lists all positions where this single sequence differs from the set. Line 7 corresponds to the node that separates all sequences of subfamily 4 from all sequences of subfamily 5. There are 98 positions where these two subfamilies differ (i.e., the intersection of the sets of amino acid residues occurring in both subfamilies is empty). Further inspection revealed that 18 of these 98 positions refer to insertions or deletions in the alignment, and 39 positions refer to proper mutations with a conserved amino acid residue in at least one of the two subfamilies (11 of them have a conserved residue in each of the subfamilies). As a side note, disregarding these two discussed lines of Figure 4.4, we see that several lines contain groups of very close positions, which could indicate functional regions.

To summarize, we have observed that many of the active site annotations in Uniprot for the Enolase dataset are present in prominent positions in our tree. Moreover, the tree makes suggestions for new annotations on two levels: it identifies possible new active sites, and it identifies new sequences that contain a known active site. We estimate that both types of information can be valuable for biological analysis.

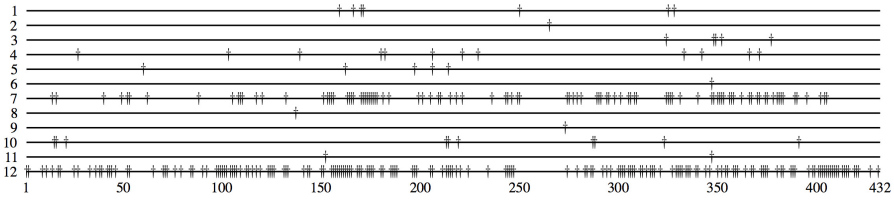


Figure 4.4: Identified polymorphic positions in first four levels of the Enolase tree. The line numbers refer to the numbering of the tree nodes in Figure B.4 in Appendix B.

4.6 Conclusions

In this chapter, we investigated the use of a divisive conceptual clustering algorithm for protein subfamily identification. The proposed method clusters protein sequences not only based on their overall similarity, but also based on the presence of conserved amino acid residues. It first builds a hierarchical tree using the divisive clustering method described in Chapter 3, which uses tests based on polymorphic positions to split the sequences. Then, it uses a post-pruning procedure to extract the predicted subfamilies from the tree. The polymorphic positions used to split the sequences result in a candidate list of functionally important sites. Moreover, new family members can easily be classified into one of the predicted clusters, by sorting them down the tree and checking the corresponding internal node tests.

We evaluated the proposed method on 11 datasets, and we compared its results with those of the phylogenomic method SCI-PHY. Next to analyzing the predicted clusters we also analyzed the underlying tree, for which we proposed two intuitive measures. Furthermore, we compared the classification results given by the tree output by our method with those given by profile hidden Markov models; and we compared the mutations that occur in the tree to known functional site annotations for one dataset.

We have shown that: (1) using splits based on polymorphic positions results in trees that are highly discriminating between subfamilies; (2) the tree topologies produced by our method have a better quality than the SCI-PHY trees; (3) our method produces a protein subfamily clustering at least as good as the ones predicted by SCI-PHY, and with the advantage of having in general a lower number of singleton clusters and a larger percentage of sequences in pure clusters; (4) the underlying decision tree classifies new sequences nearly as good as profile hidden Markov models; (5) there is evidence that the tree can identify active sites.

All these results are arguments in favor of using the proposed method for automated protein subfamily identification.

Chapter 5

Peptide identification using mass spectrometry data

In the previous two chapters, we explored the evolutionary information encoded in biological sequences for two related tasks: phylogenetic tree reconstruction and protein subfamily identification. To that aim, we compared sequences originating from different organisms/genes. In this chapter[1], we explore the information encoded in a DNA sequence in a different way. We use the full translation of the genome of an organism to obtain peptide sequences that would be potentially produced by that organism. We then use this information to assist peptide identification using mass spectrometry data.

5.1 Introduction

The study of all peptides expressed at a certain time in a certain organism, tissue, or cell type is the subject of peptidomics [96, 110, 51, 98, 64]. In this context, tandem mass spectrometry (MS/MS) is commonly used for peptide identification[2]. More precisely, an unknown peptide undergoes fragmentation, and its fragment masses are registered in a so-called peptide fragmentation spectrum (also called peptide mass spectrum or MS/MS spectrum). Then, computational methods infer the peptide sequence from its spectrum [120, 92].

[1] Based on the technical report published as supplementary material of the paper "PIUS: Peptide Identification by Unbiased Search" [Costa et al. 2013].

[2] MS/MS is also used in the context of proteomics, as we discuss in Section 5.2.1.

Currently, most researchers use database search methods for fragmentation spectrum analysis [34, 26, 27]: for each protein sequence in the database, potential fragments are predicted along with their theoretical fragmentation spectrum; a scoring function then measures how well these calculated spectra match the experimentally determined one, returning the top scoring solutions. This approach is commonly preferred over *de novo* sequencing because the success of the latter crucially depends on the quality of the MS/MS data. *De novo* sequencing methods infer the amino acid sequence from the mass differences between neighboring peaks in the fragmentation spectrum. However, from spectra of moderate quality these methods cannot extract enough information to unambiguously infer the complete amino acid sequence.

The database search approach also has its limitations. First, if the correct fragment is not derived from one of the proteins in the database, the search cannot provide the correct peptide identification. Second, considering post-translational modifications (PTMs)[3] [87] can drastically increase the computational cost associated with the search. And third, as peptide fragmentation [120] is a complex process which is not yet completely understood, designing scoring functions that represent it with high fidelity is still a challenge.

We introduce a new method that tackles the first limitation (incomplete search space). The main motivation for this work is that in peptidomics research there are still many fragmentation spectra of good quality that remain unidentified, presumably at least in part due to an incomplete search space [91, 44]. The proposed method, which we call Peptide Identification by Unbiased Search (PIUS), performs peptide identification from MS/MS spectra using the six-frame translation of the complete genome.

The novelty of PIUS lies in three aspects. First, a larger search space is considered than in related studies that have performed genome-wide peptide searches in the context of proteogenomics. Second, PIUS is designed for naturally occurring peptide identification rather than protein identification or gene/protein discovery. Third, it performs an exhaustive genome-wide search, which differs from the search strategy used by existing peptidomics methods. We will discuss these aspects in more detail later.

The remainder of the text is organized as follows. Section 5.2 gives an overview of how mass spectrometry data can be used for peptide identification, and discusses related work. Section 5.3 describes PIUS. Section 5.4 presents our experiments and results. They show that, for the mass spectrometry data used

[3]PTMs are chemical modifications that occur after the translation of nucleotides into amino acids (see Section 2.1.2). In this work, we will not distinguish between *in vivo* PTMs and chemical modifications that occur *in vitro* as they are both observed by the mass spectrometer as amino acids with an anomalous mass.

in our evaluation, the unbiased scan performed by PIUS yields top-scoring sequences identical to an accepted "gold standard" much more frequently than a non-exhaustive method. We conclude in Section 5.5.

5.2 Background and related work

Mass spectrometry (MS) can be defined as a technology that "weighs" unknown molecules. To that aim, the molecules are first ionized and then their mass over charge (m/z) is measured based on their trajectory inside the MS equipment.[4] The idea behind this strategy is that the information about the mass of a molecule can help in its identification. Tandem mass spectrometry (MS/MS) is a variant of this strategy. It combines two (or more) stages of MS. This combination allows further analysis of fragments of interest, facilitating the identification of the molecules being analyzed [15]. MS/MS is the technology commonly used for peptide identification.

5.2.1 Peptide identification in peptidomics and proteomics

Peptide identification using MS (or MS/MS) data is used in the context of both peptidomics and proteomics. Although the techniques and protocols used for both contexts have common features, they have prominent differences [15]. While in peptidomics the final goal is to identify naturally occurring peptides, in proteomics peptide identification is used as a means to obtain protein identification. As a result, for the former it is important to have a high recall rate in terms of identified peptides, while for the latter it generally suffices to identify some of the peptides originated from a genomic region to make a gene/protein prediction. Another difference is that in peptidomics proteins are digested into peptides *in vivo* (for example, by proconvertases [60]) at sites that are difficult to predict reliably, while in proteomics the protein digestion occurs *in vitro* at predictable cleavage sites.

As PIUS is designed for the identification of naturally occurring peptides, we discuss MS from the point of view of peptidomics.

5.2.2 MS and MS/MS experiments

A typical MS experiment (see Figure 5.1.a) for peptide identification starts by extracting peptides from the biological sample being analyzed (a brain

[4] The mass over charge is the mass of the ionized molecule divided by its charge.

tissue, for example). The peptide mixture is then separated in smaller samples. This can be done based on hydrophobicity by means of liquid chromatography (HPLC, high performance liquid chromatography), for example. This sample separation is necessary to reduce the complexity of the sample, since a mass spectrometer can only analyze a limited amount of ions within a certain time interval. After this stage, the samples are ready to be given as input, one at a time, to the mass spectrometer. The peptides are then ionized and their m/z is measured by a mass detector. This process results in an MS spectrum, which is a two-dimensional graph that displays the relative abundance (RA) of the ions (also called ion intensity) on the Y axis, and their m/z on the X axis.

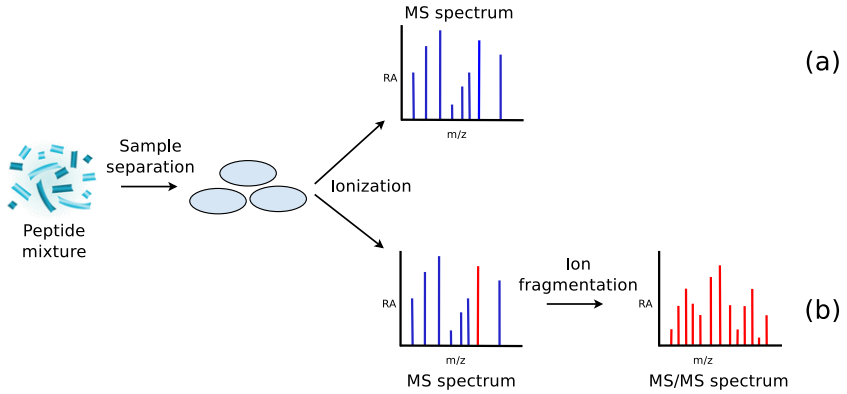


Figure 5.1: Illustration of (a) an MS experiment workflow and (b) an MS/MS experiment workflow.

The MS/MS experiment (see Figure 5.1.b) goes one step further. Namely, ions of interest are selected and subjected to fragmentation through collision. These ions are called precursor ions. The fragment ions (or product ions) generated from each precursor ion are then measured by a mass detector, resulting in an MS/MS spectrum. An MS/MS spectrum is similar to an MS spectrum, with the difference that in the former the peaks correspond to fragment ions of a peptide, instead of complete peptide ions.

5.2.3 Ion fragmentation

During ion fragmentation, the precursor ion typically breaks in two parts, generating one fragment containing the N-terminus of the original peptide

sequence and a complementary fragment containing the C-terminus.[5] The resulting fragments will only be detected if they carry at least one charge.

According to the standard nomenclature used for peptide fragment ions that arise from MS/MS [12, 109], ions are indicated with the letters a, b or c, if the charge is retained on the N-terminal fragment, and with the letters x, y or z, if the charge is retained on the C-terminal fragment. These letters are used with a subscript that indicates the number of residues in the fragment. Figure 5.2 shows an example of ion fragmentation using this nomenclature. Note that the distinction between a, b and c ions (and between x, y and z ions) lies in the place where the peptide bond is cleaved.[6]

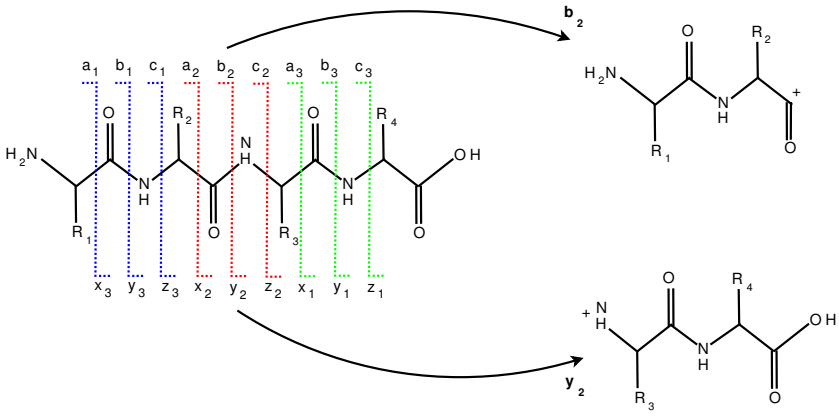


Figure 5.2: Example of ion fragmentation.

As several copies of the same precursor ion undergo fragmentation, different kinds of product ions can be produced (and then recorded in the MS/MS spectrum), as well as product ions of the same kind with different lengths (i.e., with different numbers of residues). In the last case we speak of an ion series. Figure 5.2 depicts three ions for each ion series. For example, ions b_1 and b_2 belong to ion series b, and their length differs in one amino acid.

Variants of the aforementioned ion series can also be observed. For example, ions may lose an ammonia molecule (reducing their mass by ca. 17 Da). These ions are indicated by the superscript "*": b^*, y^*, etc. Ions that lose a water

[5]The N-terminus refers to the start of the peptide/protein sequence and the C-terminus refers to its end.

[6]Apart from the ions retaining the N-terminus or the C-terminus, other types of ions can be produced during fragmentation, e.g. immonium ions and ions with partial side chain loss. Immonium ions are produced by the combination of a and y type cleavages and have just a single side chain.

molecule (reducing their mass by ca. 18 Da) are indicated by the superscript "0": b^0, y^0, etc.

5.2.4 MS/MS spectrum analysis

The information contained in an MS/MS spectrum can be used to identify the peptide sequence it originated from. This task is called MS/MS spectrum analysis or interpretation. As this is a laborious task to be performed manually, computational methods are used to assist peptide identification [120, 92]. In this context, there are two distinct computational approaches: *de novo* sequencing and database search. Some methods combine both approaches in a hybrid strategy. They use *de novo* sequencing to guide/assist the database search. We briefly describe these approaches next.

De novo sequencing

De novo sequencing methods try to identify a peptide using no other information than its mass spectrum. More specifically, they infer the peptide sequence from the mass differences between neighboring peaks of the same ion series; the idea behind this strategy is that if the difference between the mass of two product ions of the same ion type is equal to the mass of one amino acid, there is evidence that that amino acid is part of the peptide sequence. Figures 5.3 illustrates the *de novo* sequencing of the peptide FDKPRP. In this example, the sequence is obtained by calculating the mass differences from both b and y ion series. Examples of *de novo* sequencing methods are PepNovo [45] and Peaks [85].

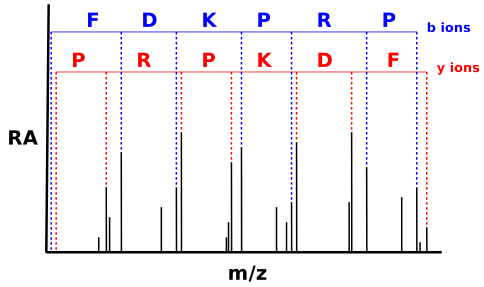


Figure 5.3: Example of *de novo* sequencing. The peptide is sequenced by calculating the mass differences between neighboring peaks from the same ion series.

The main limitation of *de novo* sequencing methods is that they can only identify a peptide completely if the generated MS/MS spectrum is of high quality. If a number of product ions are not detected by the MS equipment, these methods cannot extract enough information to unambiguously infer the complete sequence. Moreover, the presence of multiple and redundant fragment ion series is also a challenge for *de novo* sequencing [5]. In many cases, *de novo* sequencing is only able to identify short fragments of the sequence.

Database search

In the database search approach, for each protein sequence stored in the database, potential fragments are predicted along with their theoretical MS/MS spectrum. A scoring function then measures how well these calculated spectra match the experimentally determined peptide spectrum. The result is a list containing the top scoring solutions. Figures 5.4 illustrates the database search procedure. Examples of methods that implement this approach are: X!Tandem [27], MS-fit [68], OMSSA [50], Mascot[7], and SEQUEST [34].

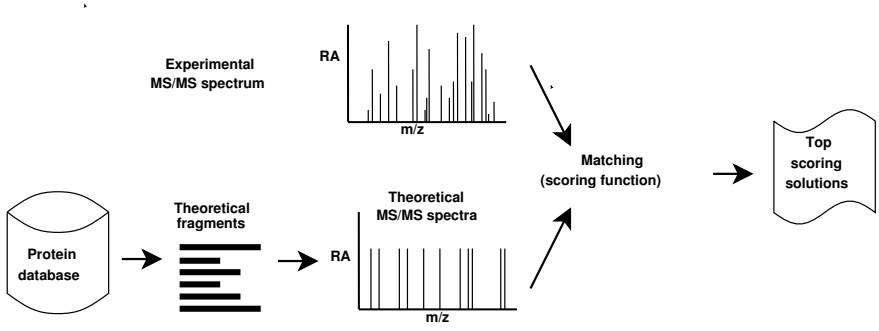


Figure 5.4: Database search approach.

However, as mentioned earlier, the success of this approach relies on the presence of the solution in the database used in the search: if the correct fragment is not derived from one of the proteins in the database, the search cannot provide the correct peptide identification. This may result from incorrect prediction of protein sequences from genomic DNA due to e.g. underprediction of short open reading frames, or errors in exon boundary prediction. A solution for this limitation is to consider the six-frame translation of the entire genome, as discussed in Section 5.2.5.

[7] http://www.matrixscience.com/search_form_select.html

Hybrid methods

Hybrid methods combine elements of the two aforementioned approaches. They use *de novo* sequencing in a first stage to generate peptide sequence tags (PSTs). The resulting tags are then used to filter protein databases, reducing the number of potential candidates in the database search. Inspect [46] and MS-Blast [114] are examples of hybrid methods. In the next section, we discuss three other hybrid methods, which have been proposed for genome-wide searches.

5.2.5 Search against the six-frame translation of the genome

To tackle the limitation of database search methods related to the incomplete search space, recent studies have pointed out the importance of considering the six-frame translation of the entire genome in the database search [39, 67, 92].

Although this idea has been investigated in the context of proteogenomics,[8] these studies were restricted to a limited search space, as we discuss in more detail in this section. Moreover, the final goal of these studies was protein/gene discovery and not peptide identification (see the discussion about differences between proteomics and peptidomics analyses in Section 5.2.1).

Recently, three peptidomics methods have been proposed for genome-wide searches: MS-Dictionary [73], MS-GappedDictionary [67], and IggyPep [91]. The three methods, which follow a hybrid approach, are briefly discussed in this section.

Proteogenomic studies

Yates et al. [140] were the first to demonstrate the benefit of considering peptide search against the complete translation of genomic data to improve genome annotation. Due to the unavailability of sequenced genomes at the time, this pioneering work used EST (expressed sequence tag) databases instead of raw genomic data.

Since then, this strategy has been applied to raw genomic data from several organisms. However, most of these studies considered small genomes, due to the computational cost of such a search, as argued by Kim et al. [73]. They mainly used genomes from bacteria [65, 134, 56, 55, 113], but also genomes from fungi [135, 99] and some plants [4, 113] were used. Moreover, many of

[8] Proteogenomics investigates the use of proteomic data (usually mass spectrometry data) to improve genomic annotation (gene prediction, correction of DNA sequencing errors, etc.).

these studies [77, 25, 71, 117, 4, 113] considered only tryptic peptides (peptides produced by cleavage by the enzyme trypsin) when searching the translation of the genome, which reduces the number of candidates to be analyzed considerably. Furthermore, some studies [56, 55, 124] limited the number of candidates to be analyzed by using PSTs to filter the search space. One other study [4] applied the same idea but with complete *de novo* reconstructions. Wang et al. [134] reduced the search space by considering only regions of the genome predicted by a gene-finding program.

In contrast to the aforementioned studies, Fermin et al. [39] did not apply any strategy to reduce the search space and considered a large genome (human genome). They performed standard database search using a cluster of computers.

Peptidomics methods for genome-wide searches

Sangtae et al. [73] proposed MS-Dictionary, an efficient method for peptide identification that allows searching against the six-frame translation of the genome. The method first creates a set of peptide reconstructions with high probability of containing the correct solution (called a spectral dictionary). This dictionary is then used to search against the database or the complete translation of the genome. However, as pointed out by Jeong et al. [67], this method has two main limitations. First, it is less efficient if MS/MS data derived from long peptide sequences is considered, since the number of reconstructions contained in the dictionary grows too much. Second, results from the search against the human genome showed that the method fails to identify many peptides [67].

To overcome these limitations, Jeong et al. [67] proposed MS-GappedDictionary, which builds a so-called "gapped spectral dictionary" before searching a database (or the complete translation of the genome). The idea is similar to the one used for MS-Dictionary, with the difference that a gapped spectral dictionary allows reconstructions with gaps. The gaps are generated for the positions for which the sequence reconstruction procedure is uncertain about which amino acids should be inserted. This change results in much smaller spectral dictionaries than those produced by MS-Dictionary, and in a larger number of correct peptide identifications [67].

IggyPep [91] queries the full genome translation using complete *de novo* reconstructions or PSTs. In contrast to MS-Dictionary and MS-GappedDictionary, IggyPep does compute the *de novo* reconstructions internally, but uses the output of a *de novo* sequencing method as input.

5.3 Proposed method

We introduce a new method that tackles the limitation related to the incomplete search space in the database search approach, discussed in Section 5.2.4. The proposed method, which we call Peptide Identification by Unbiased Search (PIUS), performs peptide identification from MS/MS spectra using the six-frame translation of the complete genome. PIUS differs from peptide genome-wide searches performed in the context of proteogenomics (see Section 5.2.5) in two aspects. First, PIUS considers a larger search space, since it is not limited to small genomes and does not make prior assumptions to reduce the search, such as assumptions about enzymatic cleavages. Second, the goal of PIUS is peptide identification while in the case of proteogenomics studies the goal is protein/gene discovery. In contrast to the peptidomics methods MS-Dictionary [73], MS-GappedDictionary [67], and IggyPep [91] that also allow a search against large genomes, PIUS does not limit the analysis of the genome to a small set of sequences that match a list of *de novo* reconstructions or PSTs. Instead, it performs an exhaustive scan of the translation of the six reading frames of the complete genome. Therefore, this search is not biased towards a subset of candidates, and eliminates the need for good-quality *de novo* reconstructions or PSTs.

In the next sections, we describe PIUS in more detail using the following notation. During a standard sequence fragmentation, ions retaining the N-terminus or the C-terminus of the original peptide sequence are produced. We call the former prefix fragment ions, and the latter suffix fragment ions. We also use the word prefix to denote a fragment of a candidate sequence containing its N-terminus.

5.3.1 Overview of the method

PIUS works as follows. Given an MS/MS spectrum and a specific genome as input, it analyzes how well each of the translated genomic sequences matches the spectrum. To investigate these sequences, the outermost loop of the algorithm iterates over all starting positions of each translated reading frame, and for each starting position PIUS considers the sequences in the order of increasing length. It moves to the next starting position when the calculated mass of the sequence exceeds the measured mass of the intact peptide that generated the experimental spectrum. This order of traversing the search space avoids iteration over candidates of which the mass is too large, and allows for ions detected in a prefix to be reused in the analysis of its extensions. Furthermore, this traversal order is also used for pruning, as discussed later in this chapter.

During the search, PIUS keeps a running list of the top k scoring solutions, where k is set by the user.

When analyzing a candidate sequence, PIUS first checks its calculated mass. If this matches the mass of the measured peptide, given a certain error tolerance, then the candidate is eligible to enter the top k solutions depending on its score. If the mass of the candidate is too small, PIUS evaluates whether the sequence can be a prefix of the correct solution. If not, it can prune from consideration all its extensions.

To calculate the quality of a prefix, we calculate all ion matches that can be analyzed up to that point,[9] including suffix fragment ions, and give a score to the prefix using the selected scoring function. The masses of the suffix fragment ions are calculated using the mass of the parent ion and the masses of the prefix fragment ions.[10] By calculating the masses of the suffix fragment ions in advance, we do not need to wait for the sequence to be complete to start verifying them. Having this information gives a better evaluation of the prefix, which benefits the pruning procedure (Section 5.3.3). If only prefix fragment ions were used in the prefix evaluation, our pruning procedure would be biased towards the presence of these ions in the candidate peptides.

Figure 5.5 shows the described search strategy in pseudocode. More specifically, it shows how PIUS analyzes a translated fragment of the genome. We consider a translated fragment an amino acid sequence without stop codons. PIUS output the top k list when all possible translated fragments of the genome have been analyzed. When AnalyzeTranslatedFragment is first called, the top k list is empty.[11] PruneSearch is the pruning procedure implemented in PIUS; we discuss it in Section 5.3.3.

[9] The contribution of all ion matches found in shorter prefixes is reused, so only few ions per prefix have to be evaluated.

[10] For single charged ions, for example, the mass of a suffix fragment ion is calculated by subtracting the mass of the corresponding prefix ion from the mass of the parent ion, plus the mass of a hydrogen atom. This calculation assumes that the final sequence, which has the current sequence as a prefix, will have a total mass that matches the mass of the parent peptide.

[11] The top k list contains the candidates with the k highest scores found until that moment in the search, along with the score of the prefixes for each candidate in the list. The information about the prefixes will be used in the pruning procedure described in Section 5.3.3.

```
AnalyzeTranslatedFragment

    // Input: Mass spectrum M, translated fragment T
    // Output: Top k list: list containing the top scoring candidates

    Starting position P := beginning of T
    Lenght of the current candidate L_s := 1
    while end of T has not been reached do
        S is a subsequence of T that starts in P and has length L_s
        if mass of S > mass of the intact peptide given by M then
            P := P +1
            L_s := 1
        else
            Look for ion series matches between S and M
            V(S) := calculate score of S
            if mass of S matches that of the intact peptide then
                UpdateTopkList
                P := P +1
                L_s := 1
            else
                if PruneSearch then
                    P := P +1
                    L_s := 1
                else
                    L_s := L_s + 1


UpdateTopkList

    // Input: Current top k list, candidate sequence S
    // Output: List after update

    if top k list is not full yet then
        Insert S into the list
    else
        if V(S) is larger than the lowest score of the list then
            Delete the lowest scoring solution
            Insert S into the list

    // When S is inserted to the list, the scores of its prefixes are stored as well.
    // This information is going to be used in the procedure PruneSearch.
```

Figure 5.5: Pseudocode of PIUS for the analysis of a translated fragment T. We discuss the pruning procedure - PruneSearch - in Section 5.3.3.

5.3.2 Scoring functions

The default scoring function in PIUS is derived from the one used in SEQUEST [34]. While the latter only considers ion series b and y, PIUS considers prefix fragment ion series b, b^*, b^0, a, a^*, and a^0, as well as suffix fragment ion series y, y^*, and y^0 (see Section 5.2.3 for more details about ion fragmentation.). We assign a weight to each ion series to give more importance to abundant ions. To define the weights, we measured the relative frequency of each ion series in a distinct dataset which we used only for this purpose and for which the correct identifications were known. The user can reuse our weights, define his own weights derived from a chosen training procedure, or enter frequencies obtained from the literature. We call the default scoring function MIWS (multi-ions weighted SEQUEST). It is defined as:

$$MIWS = \frac{\left(\sum_{j=1}^{9} w_j * SI_{mi}^{j}\right) * n_{mi} * (1+\beta) * (1+\rho)}{n_t} \tag{5.1}$$

where j iterates over the nine ion series considered; SI_{mi}^{j} is the sum of the intensities of matched ions[12] for ion series j; and n_{mi} is the total number of the matched ions. Correction factor $1+\beta$ increases the score when successive ions in an ion series are found. β is initially zero and is increased by 0.075 for each sequential ion in a series. Similarly, $1+\rho$ is a correction factor for the occurrence of certain immonium ions.[13] Specifically, if an immonium ion for the amino acids His, Tyr, Trp, Met, or Phe is present in the spectrum along with the associated amino acid in the sequence under consideration, then ρ, which is initially zero, is increased by 0.15. If the amino acid is not present in the evaluated sequence, then ρ is decreased by 0.15.[14] Finally, n_t is the total number of predicted sequence ions.

Alternatively, the user can select four other scoring functions. One of them is the original SEQUEST scoring function, which corresponds to Equation 5.1 when only the b and y series are considered, and the weights are set to 1. The other three functions are derived from the functions X_I, X_{II}, and X_{III} proposed by

[12] A matched ion is an ion whose mass corresponds to the mass of one of the peaks in the mass spectrum, within a certain error tolerance.

[13] We consider the same set of immonium ions as in the original SEQUEST function. For a definition of immonium ions, see Footnote 6 in Section 5.2.3.

[14] The idea behind considering immonium ions in the scoring function is explained with following example. Consider that the mass of the immonium ion for the amino acid Trp is found in the spectrum. In this case, it is likely that Trp is present in the sequence of the peptide we are trying to identify. Therefore, in our example, the scoring function will penalize candidates that do not have that amino acid, and reward those containing it. If the mass of that immonium ion is not present in the spectrum, there is no reward (or penalization) for the occurrence (or absence) of Tpr in the sequence.

Fenyö and Beavis [38]. As the only change that we performed was to consider more ion series than the original formulation (we consider the same nine ion series that we consider for MIWS), we keep the original names for the functions. We show the formulae for X_I, X_{II}, and X_{III} in Equations 5.2, 5.3, and 5.4, respectively. In the equations, n^j_{mi} is the total number of matched ions for ions series j.

$$X_I = \sum_{j=1}^{9} SI^j_{mi} \tag{5.2}$$

$$X_{II} = X_I * \prod_{j=1}^{9} n^j_{mi}! \tag{5.3}$$

$$X_{III} = X_I * \exp(\sum_{j=1}^{9} n^j_{mi}) \tag{5.4}$$

5.3.3 Pruning procedure

When evaluating a prefix sequence, we use a pruning procedure that compares the score calculated for that sequence to the lowest score obtained by a prefix of the same length in the top k list. If the current score is lower than α times the lowest prefix score, then the current sequence is not extended and we move the starting position pointer one position downstream. The parameter α determines the eagerness to prune; $0 \leqslant \alpha \leqslant 1$. Since a very short prefix is unlikely to contain sufficient statistical evidence to decide between pruning a sequence or not, pruning is only allowed after the prefix has reached a user-defined minimum length L_M. Moreover, pruning is only enabled once the top k list is full.

More formally, let $V(S)$ be the score of the sequence S and let $S(1\colon p)$ be the prefix of S with length L. Then, given that we want to compose a good top k of full sequences, we prune those sequences S for which the following condition is observed

$$\exists L \geqslant L_M : V(S(1\colon L)) < (t_L * \alpha), \tag{5.5}$$

where t_p is the lowest score among the prefixes of length p of all solutions present in the top k list. Figure 5.6 shows this procedure in pseudocode.

Note that for any $\alpha \neq 0$ PIUS does not compute the exact score of all possible candidate sequences. However, PIUS still exhaustively considers every starting position of the genome and it computes for all candidates at least an

PruneSearch

// **Input:** Top k list, $V(S)$: score of the current candidate S, L_s: length of S
// **Output:** "Prune" or "Do not prune"

L_M is the minimum length allowed for pruning
if $L_s < L_M$ **or** top k list is not full yet **then**
    Return "Do not prune"
else
    t_{L_S} := lowest score among the prefixes of length L_s in the top k list
    α is the parameter that defines the eagerness to prune
    if $V(S) < t_{L_S} * \alpha$ **then**
        Return "Prune'
    else
        Return "Do not prune"

Figure 5.6: Pseudocode of the pruning procedure performed by PIUS.

approximation of the scoring function: the score of a prefix of at least L_M amino acids.

5.3.4 Post-translational modifications

The user can ask PIUS to search in an even far larger search space than the one we have just described, namely, that of sequences containing PTMs. If this option is enabled, PIUS searches for (combinations of) modifications that are commonly observed in peptides, in particular: amidation, N-terminal pyroglutamate, acetylation, methylation, dimethylation, trimethylation, half of a disulfide bridge, sodium cation, deamidation, diacetylation, phosphorylation, oxidation, and dioxidation. Additionally, a 12 Da or 24 Da mass shift is allowed for the N-terminus of the peptide and the amino acids K, R, H, C, Y, W, and F, to account for possible reactions with formaldehyde, a contaminant in some of the solutions used for peptidomics. Additionally, the user can specify his own choice of arbitrary PTMs.

PIUS considers a modified amino acid residue as it was an additional residue. Thus, when PTMs are considered, two or more amino acid residues can be produced from the same codon of the genome: the unmodified amino acid and the modified one(s). PIUS investigates each one of these possibilities.

Of course, searching for combinations of PTMs has a large computational

cost, and the search space enlargement raises the probability to obtain false positives. How to deal with these issues is an interesting venue for future research. Currently, these issues are mitigated by offering to the user the option to limit the number of PTMs considered per candidate.

5.3.5 Limitations of the method and possible solutions

Even though PIUS addresses the incomplete search space limitation to a great extent (by considering the full translation of the genome), there are still cases that are not covered by our current search strategy. We discuss three cases next.

PIUS can search for modified amino acid sequences given a list of PTMs specified by the user. However, if the experimental peptide suffered a modification that is not specified in this list, PIUS cannot find the correct solution. One idea to solve this limitation is to apply strategies that can generate a list of potential PTMs for the spectra being analyzed, and give this list as input to PIUS. For example, sample-specific PTMs can be identified based on clustering approaches as described by Menschaert et al. [93]. Subsequently, these identified PTMs can be included in the PIUS search.

Another limitation is related to polymorphism. This phenomenon is characterized by the occurrence of two or more versions of a sequence or a set of sequences in the genome of an organism. As PIUS does not account for this, polymorphic peptides might not be in the search space it considers. One possibility to deal with this limitation is to allow PIUS to get, as input, genomes encoding polymorphic information, and to try all polymorphic translations. However, depending on the amount of polymorphism in the genome, this solution might be too computationally expensive, apart from increasing the rate of false positives. Another possibility is to deal with polymorphism in a peptidomics workflow in the spirit of the one proposed by Menschaert et al. [92]. This workflow would start with a conventional database search. Then, the spectra that remained unidentified after the first phase would be checked for polymorphism in a new database search. This search would take into account known information about polymorphism in the proteome (e.g., complement the database with different polymorphic versions of proteins). Then, the still remaining unidentified spectra would be given as input to PIUS, which would be the last resource in the workflow. By doing so, the spectra given as input to PIUS would have already been checked for polymorphism earlier in the workflow. Of course, this is a partial solution, in the sense that it would not account for unknown polymorphisms, nor for polymorphisms outside the known proteome of the species.

Finally, PIUS does not consider peptides originating from splicing processes. This limitation could be mitigated by giving to PIUS known information about splicing of coding regions in the genome, so that this information could be used in the genome translation. Of course, this solution relies on the current annotation of the genome, and can thus insert bias in the search.

5.4 Empirical evaluation

We start the evaluation by comparing the different scoring functions implemented in PIUS. Next, we evaluate the results returned by PIUS when PTMs are considered in the search. We then compare PIUS with the tool MS-Gapped-Dictionary [67]. Finally, we evaluate our pruning procedure.

5.4.1 Experimental setup

We evaluate PIUS on a subset of 109 spectra from a set of peptide mass spectra from a combined set of peptide mass spectra from different mouse tissue and cell line samples produced by MALDI-TOF-TOF[15] [128], [unpublished data[16]]. The subset was obtained as follows. Following the same idea discussed by Menschaert et al. [90], we used a combination of two search algorithms (X!Tandem and OMSSA) within SearchGui[17] to obtain the peptide identification. This search was performed against the protein sequence database SwissProt [8], without specifying any enzymatic cleavage, with precursor mass tolerance 0.8 Da, fragment mass tolerance 0.6 Da, and allowing the following PTMs: N-terminal acetylation, C-terminal amidation, oxidation on M, and pyroglutamination on N-terminal Q. We then analyzed the results with Peptide-Shaker[18] and retained only those with a confidence level of 100%, allowing us to use them as a gold standard. Among the 109 spectra, 6 spectra have a PTM: 1 spectrum with C-terminal amidation, 1 spectrum with N-terminal pyroglutamination, and 4 spectra with N-terminal acetylation. The size of our dataset is typical for peptidomics studies, in contrast to proteomics studies, where datasets are typically larger.

[15]MALDI-TOF-TOF is MS/MS by matrix-assisted laser desorption/Ionization with time-of-flight selection followed by fragmentation and time-of-flight measurement.

[16]Out of the 109 spectra used in the experiments, 19 spectra are part of a set of unpublished spectra from the SBO grant IWT-50164 of the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

[17]SearchGUI is a graphical user interface for configuring and running X!Tandem and OMSSA simultaneously. http://code.google.com/p/searchgui/.

[18]PeptideShaker is a search engine platform for visualization of peptide and protein identification results from multiple search engines. http://code.google.com/p/peptide-shaker/.

Table 5.1: Comparing different scoring functions .

	Sequest	MIWS	X_I	X_{II}	X_{III}
Top 1	85	85	22	95	82
Top 2	3	3	3	1	5
Top 10	3	8	14	4	10
Top 50	8	4	14	2	2
Top 100	2	1	1	0	0
Top 200	0	0	5	0	2
Top 500	1	0	4	1	0
Top 1000	0	2	2	0	1
Top 2000	1	0	2	0	0
Top 5000	0	0	6	0	0
Top 10000	0	0	5	0	0
Not found	0	0	25	0	1

In our experiments, we verify if PIUS can identify these gold standard solutions in the search space provided by the six-frame translation of the mouse genome. This translation contains approximately 200 times more amino acids than the mouse proteome in SwissProt. We use PIUS with an error tolerance of 0.5 Da for both peptide and fragment mass, which is large enough to account for measurement errors in MALDI-TOF-TOF data. We arbitrarily set $k = 10000$, $\alpha = 0.8$, and $L = 5$.

5.4.2 Comparing different scoring functions

In this experiment, we compare the different scoring functions implemented in PIUS: the original SEQUEST function, MIWS, X_I, X_{II}, and X_{III}. The results are presented in Table 5.1, which shows the position in the top k list occupied by the gold standard solutions. The perfect identification would be if the gold standard solution for all spectra would be placed as the top 1 solution. The numbers displayed in the table are not cumulative.

All scoring functions yield good results, except X_I. This scoring function ranks many gold standard solutions in low ranking positions, and misses 21 gold standard solutions at all. The poorer performance for this scoring function is not unexpected, since it encodes less information about the matched ions than the other functions.

The function X_{II} returns the gold standard solution at the top of the candidate list for a few more cases than functions MIWS and SEQUEST. This is surprising,

since the former does not use some of the information used by the two latter functions, namely, the match of immonium ions and consecutive ions within ion series. The function X_{III} also presents good results, but misses one gold standard solution.

These results show that, provided an adequate scoring function, PIUS obtains comparable results to those of conventional database search methods while considering a substantially larger search space. In the remaining experiments, we focus on the scoring functions MIWS and X_{II}.

5.4.3 Searching for post-translational modifications

We now evaluate PIUS for the case where PTMs are considered in the search. We consider the same PTMs used to obtain the gold standard solutions, and we restrict the number of PTMs per sequence to at most one. The results are summarized in Table 5.2. We show the results for the spectra with and without PTMs separately.

Table 5.2: Analyzing PIUS (with MIWS and X_{II}) for the case where PTMs are allowed in the search.

	Spectra with no PTM		Spectra with PTMs	
	MIWS	X_{II}	MIWS	X_{II}
Top 1	82	90	5	6
Top 2	5	6	0	0
Top 10	5	2	1	0
Top 50	5	4	0	0
Top 100	2	0	0	0
Top 200	1	0	0	0
Top 500	0	0	0	0
Top 1000	1	1	0	0
Top 2000	1	0	0	0
Top 5000	0	0	0	0
Top 10000	0	0	0	0
Not found	1	0	0	0

For the 6 spectra with PTMs, PIUS used with MIWS finds the gold standard solution as the top 1 solution for 5 cases, and as top 3 solution for 1 case. When the function X_{II} is used, PIUS finds the gold standard solution as the top 1 solution for all cases. For the 103 spectra without PTMs, PIUS used with MIWS finds the gold standard solution as the top 1 solution for 82 cases, as

the top 2 solution for 5 cases, in top 10 of solutions for 5 cases, and in a lower ranking position for 11 cases. In this experiment, 1 gold standard solutions is missed due to search space pruning. With X_{II}, the results are again slightly better than when MIWS is used: PIUS finds the gold standard solution as the top 1 solution for 90 cases, as the top 2 solution for 6 cases, in top 10 of solutions for 2 cases, and in a lower ranking position for 5 cases.

Note that these results for the 103 spectra with no PTM are slightly worse than for the case where the search is performed without PTMs (Table 5.1). This is not unexpected, since the search space is enlarged by also considering modified sequences, increasing the likelihood of returning false positives. However, for a large majority of the cases, the gold standard solution is still returned as the top-scoring solution. Moreover, PIUS successfully identifies the modified peptide in the 6 spectra with PTMs. These results together show that PIUS has a high recall rate even when PTMs are considered.

5.4.4 Comparing the results with MS-GappedDictionary

We now compare PIUS with the tool MS-GappedDictionary [67], using the parameters recommended by its authors. In particular, the charge range parameter of MS-GappedDictionary is set to 1, no enzymatic cleavage was specified, and all other parameters are left at their default values. In addition to using MS-GappedDictionary with its default value for error tolerance of 2.0 Da, we also test MS-GappedDictionary with the tolerance used by PIUS (0.5 Da). As this tool does not consider PTMs in the search, we only consider the spectra without PTMs. The results are summarized in Table 5.3. Note that the results we show for PIUS are for the case where we do not consider PTMs in the search.

With its default error tolerance, MS-GappedDictionary finds the gold standard solution for only 50 out of the 103 spectra. Out of these 50 cases, the gold standard solution is the top 1 solution for 48 cases and the top 2 solution for 2 cases. Among the 53 remaining spectra, there are 3 cases for which MS-GappedDictionary does not return any hits and 50 cases for which the gold standard solution is not among the returned solutions. With the error tolerance set to 0.5 Da, the gold standard solution is the top 1 solution for 57 cases, the top 2 solution for 2 cases, and in the top 10 of solutions for 1 case. There are 3 cases for which MS-GappedDictionary does not return any hits and 40 cases for which the gold standard solution is not among the returned solutions.

The large number of unidentified spectra returned by MS-GappedDictionary contrast sharply with the good results reported by Jeong et al. [67]. A possible explanation for this inferior performance is the source of the mass spectra.

Table 5.3: Comparison between MS-GappedDictionary and PIUS. We show results for MS-GappedDictionary with error tolerance 2.0 Da (default value) and with error tolerance 0.5 Da (default value in PIUS), and for PIUS (with MIWS and X_{II}).

	MS-GappedDictionary		PIUS	
	error tol. 2.0 Da	error tol. 0.5 Da	MIWS	X_{II}
Top 1	48	57	85	95
Top 2	2	2	3	1
Lower ranked	0	1	15	7
Not found	53	43	0	0

Jeong et al. [67] used only mass spectra produced by an LTQ linear ion trap tandem MS and an LCQ ion trap MS using ESI, which are generally more accurate than MALDI-TOF-TOF spectra.

From these experimental results we can conclude that PIUS has a higher recall rate than MS-GappedDictionary, at least for MALDI-TOF-TOF spectra.

5.4.5 Evaluating the pruning procedure

In this experiment, we evaluate the pruning procedure of PIUS. We used PIUS with the function MIWS, without considering PTMs.

In Figure 5.7, we display how many candidates are evaluated by PIUS when searching against the complete translation of the genome. The number of analyzed candidates according to their sequence length. We display this analysis for different values for the parameter α.

Note that for $\alpha = 0$, the pruning procedure is not used. The reason why the curve goes down even for $\alpha = 0$ is because PIUS does not extend a candidate when its mass is larger than the mass of the precursor ion and also due to the occurrence of stop codons in the genome translation.

We found that even the maximum $\alpha = 1$ produced results practically equal to $\alpha = 0$ for our dataset in terms of which candidates were output at the end of the search. The comparison between the resulting top k lists (with $k = 10000$) for PIUS with $\alpha = 1$ and $\alpha = 0$ revealed that PIUS with $\alpha = 1$ pruned, on average, 15 solutions which were output by PIUS without pruning. However, these pruned solutions were not among the best solutions in the list: for all spectra analyzed, the top 100 solutions of the two top k lists were the same; and for 61 out of 103 cases, the top 1000 solutions of the lists were the same. This shows that the pruning procedure does not affect the top scoring solutions.

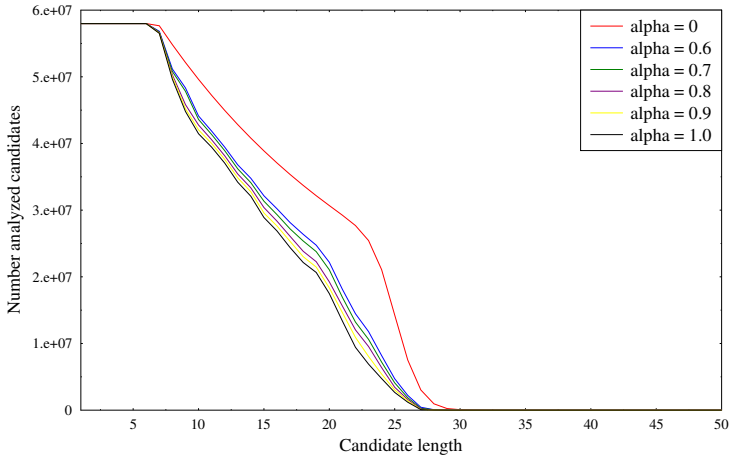


Figure 5.7: Evaluating the pruning procedure with different values for the parameter α.

However, if PIUS finds hundreds of false positives with a better score than the correct solution (given the scoring function in use), then there is the risk that this solution might be pruned away from the search space. A close analysis of the results revealed that this, in fact, happens for one of the 103 spectra: the gold standard solution, which is ranked at position 606 with no pruning, ends up being pruned away when $\alpha = 1$ is used. This is the same gold standard solution which is pruned away when PTMs are considered in the search (Table 5.2).

We also evaluate the computational cost for $\alpha = 0$ and $\alpha = 0.8$. Due to the computational work necessary to investigate all the candidates produced by the translation of the full genome (with $\alpha = 0$), the current C implementation has a throughput of 1.2 spectra per hour on an Intel Core i7-2600 when searching the 21 mouse chromosomes; when we apply the pruning procedure with $\alpha = 0.8$, the implementation has a throughput of 2 spectra per hour.

Taking into account the computational cost associated with the search performed with PIUS, we recommend to use PIUS in a layered peptidomics workflow [92] in case of large scale experiments. Using a similar idea as the one we discuss in Section 5.3.5, this workflow would first search databases in a conventional manner, and afterwards call on PIUS for the remaining unidentified high quality spectra. In case this step would become a bottleneck in the workflow, the algorithm can easily be parallelized to increase throughput.

5.5 Conclusions

In this chapter, we introduced a new method for peptide identification using MS/MS data. We call the proposed method Peptide Identification by Unbiased Search (PIUS). PIUS tackles the limitation related to the incomplete search space in the database search approach by considering the six-frame translation of the complete genome.

We evaluated PIUS on a set of 109 MS/MS spectra from a combined set of peptide mass spectra from different mouse tissue and cell line samples produced by a MALDI-TOF-TOF instrument. We compared our results with those of MS-GappedDictionary, which performs a non-exhaustive search against the genome, on the same dataset. We also evaluated the pruning procedure implemented in PIUS.

From the experimental results we can draw the following conclusions: (1) PIUS is very often able to reproduce successful identifications by conventional database search methods, even though it searches a much larger space; (2) PIUS has a higher recall rate than MS-GappedDictionary, at least for MALDI-TOF-TOF spectra; and (3) the pruning procedure partially mitigates the computational cost associated with the exhaustive genome-wide search, without deteriorating the quality of the results.

These results argue in favor of using PIUS for peptide identification. In particular, we recommend to use PIUS as a last resource tool to analyze spectra of good quality that cannot be identified by conventional database search methods.

Chapter 6

Estimating prediction certainty in decision trees

In Chapters 3 and 4, we considered two descriptive learning tasks which were solved using top-down clustering trees. In this chapter[1], we consider decision trees in the context of predictive learning. More specifically, we propose and investigate a method for estimating prediction certainty in classification decision trees.

6.1 Introduction

In classification, it is often useful to have classifiers that not only have high accuracy, but can also tell us how certain they are about their predictions. Classifiers that output some kind of reliability, likelihood or numeric assessment of the quality of each prediction are usually called soft classifiers [40]. The most common example of a soft classifier is a probability estimator, which estimates the probability that a data instance belongs to a certain class. Rankers and reliability estimators are other forms of soft classifiers.

The standard way of turning decision trees into soft classifiers consists of inferring the certainty of a prediction from the class distribution in the leaf responsible for the prediction. For example, if an instance x is classified in a leaf node with 90% of positive examples, we say that that x has 90% probability

[1] Based on the paper "Estimating prediction certainty in decision trees" [Costa, Verwer and Blockeel 2013, submitted].

of being positive. However, it has been shown that these proportions can be misleading: the smaller the leaf is, the more likely the proportion is accidental, and not inherent to the population distribution in the leaf [101, 141]; and, as decision tree learners try to make the leaves as pure as possible, the observed frequencies are systematically shifted towards zero and one [141].

In this chapter, we propose an alternative method to estimate prediction certainty in decision trees. Assume that, given data set D and k classes $c_1, \ldots, c_k$, we want to make a prediction for an unseen instance x. Suppose that we learn a tree T that predicts with high "certainty" (according to leaf proportions) that x has class c_1. Logically speaking, if we would give the learner the prior knowledge that x has class c_1 (by simply adding (x, c_1) to D), the learner should not return a tree T_{c_1} that predicts with less certainty that the class of x is c_1. If it does, there is a logical contradiction, in the sense that more evidence about some fact being true leads to less certainty about it. Moreover, if we add (x, c_2) to D, and it turns out that the tree learned from the new dataset T_{c_2} makes a different prediction than T_{c_1}, also with high certainty, then we, as observers, know that there is actually high uncertainty about the class.

More specifically, our method works as follows. Given an unseen instance x, we can, for $i = 1, \ldots, k$, add (x, c_i) to D, giving a dataset D_i from which a tree T_{c_i} is learned, and look at the prediction T_i makes for x. If all T_{c_i} predict the same class c, we can be quite certain that c is the correct class. If multiple trees predict different classes, each with high certainty, we must conclude that these predictions are highly uncertain. We also propose a way to combine the predictions of the resulting trees.

We perform an extensive evaluation of the proposed method on 48 randomly selected UCI datasets. We compare our results to those of a standard decision tree learner and a standard ensemble method. The results show that our method tends to produce (1) better ranking and reliability estimates, (2) comparable accuracy, (3) better probability estimates than the standard decision tree learner, and (4) comparable probability estimates to the ensemble method. Additionally, compared to a closely related method for reliability estimation, we show that our method produces better reliability estimates.

The remainder of the chapter is organized as follows. In Section 6.2 we discuss basic concepts related to prediction certainty in soft classifiers, and discuss related work. In Section 6.3 we describe our new method in detail. In Section 6.4 we present experiments and results, and in Section 6.5 we conclude.

6.2 Background and related work

We start this section by discussing prediction certainty estimations in soft classifiers; we discuss three different ways of interpreting prediction certainty and how to evaluate them. Then, we briefly recall how decision trees estimate prediction certainty, and discuss existing methods for improving their estimates.

6.2.1 Prediction certainty in soft classifiers

The notion of certainty associated to soft classifiers has been defined in different ways in the literature. We discuss three of them: probability, ranking and reliability estimations. For each one of them we present a measure to evaluate it; we use these measures to evaluate our method in the experimental section.

We say that a soft classifier is a probability estimator when it estimates for every possible class the true probability that a random instance with the given attribute values belongs to that class. Probability estimations are usually evaluated with the Brier score [18] (Equation 6.1), which is also called mean squared error. In the equation, j iterates over all the N predictions, i iterates over all the k classes, $t(c_i|x_j)$ is the true probability that instance x_j belongs to class c_i, and $p(c_i|x_j)$ is the estimated probability. When the true label of a prediction is given but its true probability is unknown, $t(c_i|x_j)$ is defined to be 1 if the true label of x_j is c_i, and 0 otherwise.

$$\text{Brier score} = \frac{\sum_{j=1}^{N}\sum_{i=1}^{k}(t(c_i|x_j) - p(c_i|x_j))^2}{N} \tag{6.1}$$

A ranking estimator orders the instances from high to low expectation that the instance belongs to a certain class c. For every pair of instances (x_1,x_2), the ranking defines if x_1 is more likely, equally likely or less likely to belong to c than x_2. With this information for all pairs of sequences together, the ranking defines a total order. As ranking estimation is defined in terms of pairs of sequences, we can say that this is a relative estimation. Probability estimation, on the other hand, is an absolute certainty estimation, since the estimation for each prediction can be interpreted on its own.

The ranking ability of a classifier is usually assessed using ROC analysis (see Section 2.2.3), by calculating the area under the ROC curve (AUC). When the classification is multi-class, one solution is to calculate the ROC curve for each class separately, and report the average AUC.

Finally, Kukar and Kononenko [75] use the term "reliability" to define the probability that a prediction is correct. This is, in principle, the same as probability estimation for the predicted class. However, Kukar and Kononenko [75] consider reliability in a more general way. They consider a prediction to be more "reliable" when it is has a higher probability to be correct, and evaluate the reliability skill of a classifier by assessing how well it can distinguish between correct and incorrect predictions based on the calculated reliability for each prediction. This is in fact a ranking evaluation where the predictions define only one rank over all classes together (in terms of correct and incorrect predictions), instead of internally to each class, as for standard ranking evaluation.

Kukar and Kononenko [75] evaluate this ranking of reliability scores in terms of information gain (we introduced information gain in Section 2.2.2). More specifically, they look at the predictions in the rank as a new dataset with two possible classes {correct predictions, incorrect predictions} and with a single numeric attribute {reliability-estimation}. For this new dataset, they verify which reliability score yields the best information gain when used to split the predictions in two populations. Then, they report the best information gain.

However, we argue that returning the information gain for the threshold that gives the best results does not give a global view of the reliability estimates of the classifier. Even though this procedure might be useful for automatically selecting a threshold to be used in the prediction of unseen examples, reporting results on the same instances used to choose the threshold only gives us information about the optimal cut-off for estimates. This evaluation might thus give an optimistic evaluation of the results. We therefore propose to use AUC to evaluate reliability estimates; we call it AUC reliability to avoid confusion with the aforementioned AUC calculation for evaluating ranking estimates. The advantage of using AUC is that it yields an evaluation that is independent of a fixed threshold.

In this modified evaluation framework, the best evaluated situation would be the one in which all correct predictions have a higher reliability score than the incorrect prediction (AUC reliability equal to 1). A smaller AUC reliability is obtained when (a number of) incorrect predictions have higher reliability scores than correct predictions.

6.2.2 Decision trees and certainty estimates

Standard decision tree learners estimate a classification's certainty by using the class distribution in the leaf responsible for the prediction. However, it is well-known that standard decision tree learners do not yield very good certainty estimates [101, 141]. Especially for small leaves, the sample class distribution can significantly deviate from the population class distribution, and it typically

deviates towards lower class entropy (or higher purity), due to the learning bias of the tree learner. Moreover, decision trees assign the same prediction certainty estimates for instances falling into the same leaf, and do not exploit the fact that even inside the same leaf node there might be prediction certainty differences. For example, it would be reasonable to assume that a borderline prediction in a leaf is more likely to be a misclassification than the other predictions in that leaf.

Several methods have been proposed in the literature to improve estimates in decision trees. One approach is to apply a smoothing correction (e.g., the Laplace or m-estimate smoothing) to unpruned trees [101, 40]. Another group of methods either modify the decision tree learning (e.g., by learning fuzzy [63] or lazy decision trees [88, 82]) or the way in which the predictions are made (e.g., by propagating the test instances across multiples branches of the tree and combining estimates from different leaf nodes [83], or by using internal nodes to make predictions [141]). Other methods use alternative probability calculations, e.g., by combining the class distribution from different nodes in the path taken by the test instance [133]. We discuss these methods in more detail in this section.

In contrast to these methods, which either develop a new type of decision tree learner or use different probability estimations, we propose a new way of using the results that can be obtained using any traditional decision tree learner. We do this by learning multiple trees, and combining their predictions. In contrast to ensemble methods, which also learn multiple trees, we modify the training data in a very restricted and controlled way to obtain different trees. We do this by just complementing the training data with a labeled version of the instance to be classified.

Our method is similar to the transductive method for reliable classification proposed by Kukar and Kononenko [75]. We discuss the main difference between their method and ours at the end of this section.

Smoothings

Prediction certainty estimates in decision trees are often improved using the Laplace or m-estimate smoothing [22]. These corrections are used to avoid extreme estimates[2] (i.e, extremely certain predictions): they shift the prediction certainty towards the uniform probability (0.5, in the case of binary problems) according to the size of the leaf, so that this effect will be larger in smaller

[2] A prediction certainty of 1 or 0 is never obtained when a smoothing procedure is applied, although the estimates might converge to those values in extremely large leaves.

leaves; the idea is to attribute less certainty to small leaves, since they tend to be less trustworthy than large leaves.

Equation 6.2 shows the m-estimate smoothing to calculate the probability $p_x(c)$ that x belongs to class c, where $p(c)$ is the prior estimate of the probability we want to determine, L is the leaf responsible for the prediction, $|L(c)|$ is the number of instances that belong to class c in L, and m is a parameter that determines the weight of $p(c)$ in the calculation. Equation 6.3 shows the Laplace smoothing calculation, where α determines the weight of the smoothing. Note that the two equations give the same estimates when $\alpha = 1$, $m = k$ and an uniform prior probability is assumed.

$$p_x(c) = \frac{|L(c)| + m * p(c)}{|L| + m} \tag{6.2}$$

$$p_x(c) = \frac{|L(c)| + \alpha}{|L| + \alpha * k} \tag{6.3}$$

Provost and Domingos [101] show that the Laplace smoothing applied to unpruned trees improves ranking estimation; these results are confirmed by other studies [40, 62, 63]. In terms of probability estimation, Zadrozny and Elkan [141] recommend the use the m-estimate smoothing (also in combination with not pruning) and argues that the Laplace smoothing yields little improvement in the estimates. Fierens et al. [41] show that the Laplace smoothing applied to unpruned trees yield poor probability estimates but relatively good rankings, especially when the number of classes is high. Fierens et al. [41] argue that the poor probability estimates result from poor calibration given the large size of the unpruned trees.

Ferri at al. [40] proposed a new smoothing method called m-branch. This smoothing takes the history of the samples during the splitting procedure to estimate prediction certainty. Namely, it combines the class distribution of all nodes from the root until the leaf node responsible for the prediction. In this procedure, the nodes that are closer to the leaf are given more weight. The results presented by Ferri at al. [40] were evaluated in terms of ranking estimation. They show that this smoothing yields better rankings than the Laplace and m-estimate smoothings. The disadvantage is that it requires more calculations than the other corrections, and it is, as a consequence, less straightforward to use.

Pruning procedures

Pruned trees have been shown to give worse ranking estimates than unpruned trees when the latter are used in association with a smoothing procedure [101, 40, 62, 63]. These observations were drawn from experiments with different kinds of pruning procedures, including classical pre-pruning and post-pruning methods [40], and a pruning procedure especially designed to improve probability estimates [101]. The results by Ferri et al. [40] showed that (1) when smoothing is disabled, pruning might be beneficial in some cases; (2) pruning associated with smoothing degrades the quality of the results; (3) the better the smoothing procedure is the worse the effect of pruning will be. In the context of probability estimation, Zadrozny and Elkan [141] also recommend the use of unpruned trees associated with some regularization method such as smoothing or curtailment, which we also introduce in this section.

Splitting criteria

Ferri et al. [40] compare the ranking estimates of different splitting criteria. The results revealed that the differences in the estimates were negligible. Zadrozny and Elkan [141] also show that changing the splitting criteria has little effect for probability estimation.

Ensemble methods

Provost and Domingos [101] show that bagging substantially improves ranking estimates, even more effectively than it does for accuracy estimates. They also show that when bagging is used, the use of pruning or smoothing makes little difference. In the results reported by Zadrozny and Elkan [141], the use of bagging is not as effective for probability estimation.

The two main drawbacks of this approach is the increase in computational cost and the decrease in comprehensibility of the results.

Curtailment

Zadrozny and Elkan [141] argue that unpruned trees might have some leaves with too few instances to yield good probability estimates, even when smoothing is applied. To solve this problem, they iteratively eliminate small leaves until the point where all leaves have v or more instances, where v is a parameter of the method; this procedure is called curtailment. Note that curtailment will

produce a tree in which the internal nodes have either one or two child nodes. As a result, nodes with only one child node will work either as a leaf or as an internal node, depending on the attribute values of the instance being classified.

Based on their experimental results, Zadrozny and Elkan [141] recommend the combination of curtailment with smoothing. They argue that this combination produces relatively small trees, which favors comprehensibility, while still giving good probability estimates.

Fuzzy decision trees

Hüllermeier and Vanderlooy [63] investigate the ranking ability of fuzzy decision trees. In this special kind of trees, the tests located in the internal nodes yield soft splits instead of hard splits. In a soft split, an instance is classified into the left child node with a membership degree $\mu(x)$ in the interval [0,1], and into the right child node with degree $1 - \mu(x)$. This membership degree is calculated in a way that it tends to 0.5 the closer the attribute value is of the threshold chosen for the test. At the end, the instance can be classified into different leaves, and for each leaf a support value is calculated based on the membership degrees on the path followed by the instance from the root to the leaf. Then the class with the highest overall support is returned.

Hüllermeier and Vanderlooy [63] argue that the resulting estimates are not necessarily good probability estimates, but that they yield a good ordering in terms of confidence, resulting in a good ranking performance. This good ordering is given by the "tie breaking effect" that comes along with an increased number of scores resulting from a fuzzy classification: while in a standard decision tree learner there can be at most as many different scores as the number of leaves, in fuzzy decision trees there can be as many different scores as the number of test instances.

Combining predictions of all leaf nodes

Ling and Yan [83] propose a method that propagates a test instance along multiple paths simultaneously: for each internal node, the test instance is propagated along both branches emerging from the node, but with a smaller weight for the branch that does not satisfy the test. As a result, the instance will end up in all leaf nodes of the tree. The final probability for the instance being classified is a weighted average of the contribution of all leaves.

Note that the idea of propagating sequences along multiple branches of the tree is similar to the one used in fuzzy decision trees. One of the main differences

between the two methods is that for the latter the tree induction procedure is modified, while the method by Ling and Yan [83] uses a standard procedure. The estimates are also calculated in a different way.

Shrinkage and weighted probability estimation

Wang and Zhang [133] investigate two procedures to improve prediction certainty estimates - shrinkage and weighted probability estimation (WPE), which can be used independently or combined.

Shrinkage, which was first proposed in the context of text classification [89], is a procedure that calculates the probability distribution of a leaf node by combining the probabilities from different nodes in the path taken by the test instance. A weight is assigned to each node of the path. Wang and Zhang [133] provide an algorithm to determine the weights. Note that this procedure is related to the m-branch smoothing proposed by Ferri at al. [40].

WPE is an instance-based procedure proposed by Wang and Zhang [133] that allows instances falling into the same leaf node to have different probability estimates. When an instance is classified into a leaf node, the probability distribution is calculated based on the similarity of the instance with the training instances present in that leaf.

Wang and Zhang [133] evaluate both procedures in terms of ranking estimation, and show that a combination of both outperforms all the following methods: m-branch smoothing, Ling and Yan's method [83], bagging, and Laplace smoothing applied to unpruned trees.

Transductive procedure for reliability estimation

All the aforementioned methods were proposed and evaluated in terms of probability and ranking estimations. Kukar and Kononenko [75] propose a procedure that is aimed to improve reliability estimation, which we defined in Section 6.2.1.

Their method estimates a reliability score for each prediction based on a two-step approach: first a standard inductive learning step is performed, followed by a transductive learning[3] step. These reliability scores can then be used to distinguish between unreliable and reliable predictions.

[3]In transductive learning [142], given a dataset and one or more specific unlabeled instances, the goal is to generate a classifier that makes predictions for that/those instance(s). In this case, it is not required that a generally applicable classifier is learned, as for inductive learning.

More specifically, given an unseen instance x, the method makes a prediction for x using a standard machine learning method (a decision tree learner, for example); this is the inductive step. The output of this step is then used as input to the transductive step: x is labeled with the predicted label c given by the inductive step; the instance (x, c) is added to the training data D; and a new classifier is learned. Finally, the probability distributions output by both steps are compared in order to infer the reliability of predicting x as belonging to class c. The idea behind this reliability estimation is that the more different the probability distributions are, the less reliable the prediction is. This idea is based on the theory of randomness deficiency [81, 132].

Once a reliability score has been calculated for every prediction, the predictions are ranked according to their reliability, and a threshold is defined to separate the predictions into two populations: unreliable and reliable predictions. Kukar and Kononenko [75] propose a supervised procedure to find this threshold automatically.

Note that this method is similar to the one we propose in the sense that they are both based on a transductive strategy. Moreover, they both make one prediction at a time. The main difference is that we measure the sensitivity of the learned model with respect to all possible labels, instead of only using the label which is believed (predicted) to be the correct one. Our hypothesis is that measuring this sensitivity is crucial to obtaining good certainty estimates for decision trees.

6.3 Proposed method

In this section, we introduce our proposed method for improving prediction certainty in decision trees. We start by giving the intuition of our method. Then, we describe its algorithm. Finally, we illustrate its calculations with one example.

6.3.1 Intuition of the proposed method

Consider, for example, the decision tree represented in Figure 6.1. This tree was generated for an iteration of the leave-one-out validation procedure for the well-known Iris dataset [7]. The test instance x_1 belongs to class *Iris-virginica*, but it ends up being classified in leaf 3, where 98% of the training instances belong to class *Iris-versicolor*. This high confidence misclassification happens because x_1 is a "border case" (i.e, x_1 is somewhere in the border

between classes *Iris-versicolor* and *Iris-Virginica*) and is therefore difficult to be correctly classified.

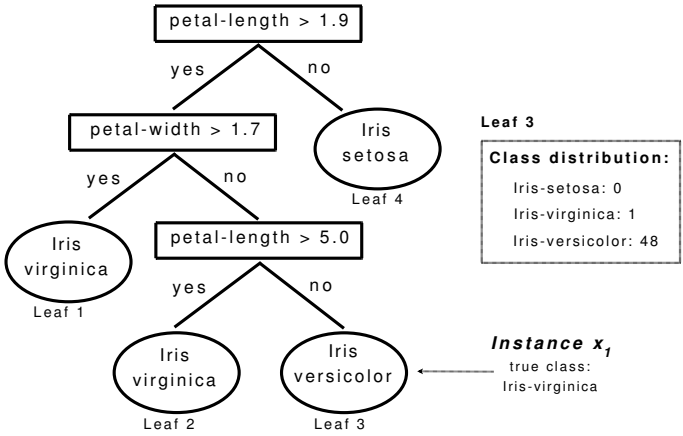


Figure 6.1: Example of a misclassification with high prediction certainty for the Iris dataset. This prediction was obtained from an iteration of a leave-one-out validation procedure.

The same problem occurs for the other six misclassified instances of Iris (which has 150 instances in total). All of them are misclassified with a similar prediction certainty as the correctly classified instances. This illustrates how the prediction certainty given by decision trees may be overconfident and, consequently, untrustworthy.

One idea to overcome this problem is to identify instances that might present some difficulties to be correctly classified, and to attribute some uncertainty to their prediction. This can be achieved by introducing small changes in the training data and checking how this affects the outcome of the decision tree; the underlying idea being that these changes should not have a strong effect on the outcome of the decision tree, unless we are dealing with cases for which the classification is more challenging, such as border cases.

In our method, the idea is to analyze the effect of changing the label of a single instance in the resulting decision tree model. In particular, we want to investigate the following questions with respect to the instance we want to classify: (1) "If we add the test instance to the training set with a different label than the correct one, will the decision tree learner find a tree that is consistent with this instance according to the wrong label?"; (2) "How certain will the

tree be about this prediction?"; (3) "How does this situation compare to the one where the instance is added to the training set with the correct label?".

Our method is therefore based on the idea that the learned decision tree can be dependent on the label of a single instance. This effect is illustrated in Figure 6.2. The two trees shown in the figure were learned for the same data used in Figure 6.1, with the difference that the test instance x_1 was included in the training data. For the left tree, x_1 was included with the correct label, *Iris-virginica*, while for the right one, x_1 was included with label *Iris-versicolor*. Note that the trees make different predictions based on the pre-defined label for x_1. The difference in the prediction is due to the change of the threshold value for the test "petal-lenght > 5.0". By adding the label *Iris-virginica* to x_1, this value is affected.

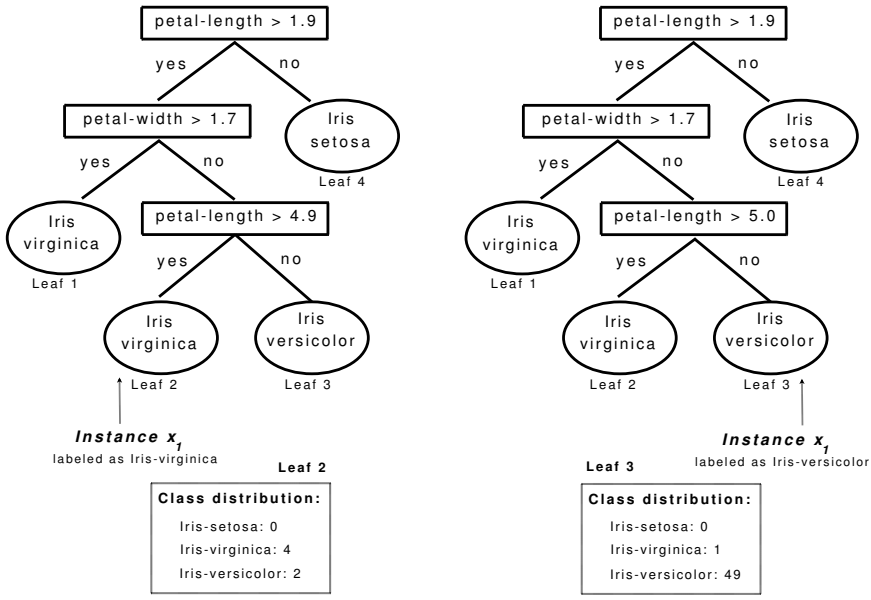


Figure 6.2: Illustration of how the tests selected during the induction of a decision tree can be dependent on the label of a single instance. The decision trees were built for the same data used in Figure 6.1, with the difference that the test instance was included in the training data. Left: x_1 was included with the correct label, *Iris-virginica*. Right: x_1 was included with the wrong label *Iris-versicolor*. For both cases the learned decision tree predicts the instance with the same label that we pre-define it.

But the effect of changing the label of one instance may also be more global and influence the choice of features for different nodes of the tree. One example

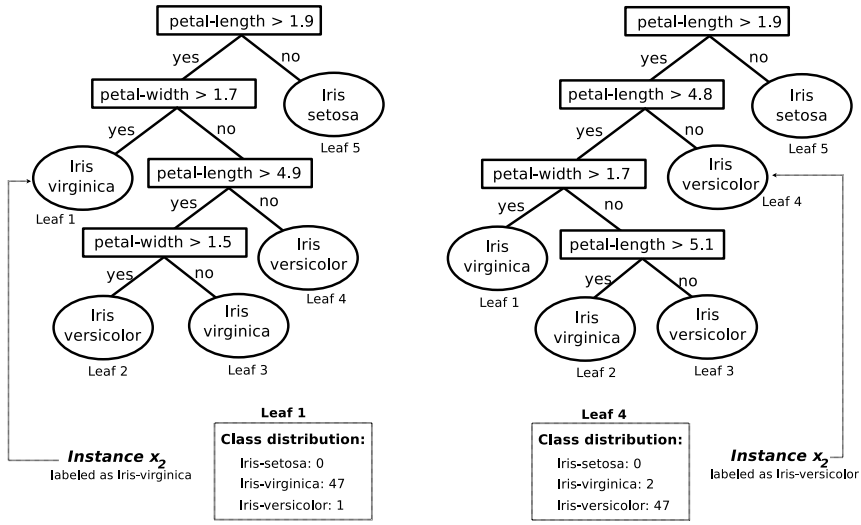


Figure 6.3: Illustration of how the tests selected during the induction of a decision tree can be dependent on the label of a single instance - example 2. The two trees were built for the same instance (x_2) of the Iris dataset. Left: x_2 was included in the training data with the correct label, *Iris-virginica*. Right: x_2 was included with the wrong label *Iris-versicolor*. For both cases the learned decision tree predicts the instance with the same label that we pre-define it.

of this situation can be seen in Figure 6.3, which shows two decision trees built to classify another instance (x_2), also from the Iris dataset. Again, we learned the left tree with the correct label and the right tree with the wrong label. For both cases, the tree predicts the pre-defined label with a very high certainty.

These two examples show situations where the learned decision tree is dependent on the label of the instance we want to classify. Intuitively, we cannot be very certain about the predicted label when the prediction model itself depends on the label we give to the instance. This uncertainty is not reflected in a certainty measure based only on leaf proportions.

6.3.2 Description of the method

Using the intuition just described, the proposed method estimates the prediction certainty by comparing trees generated for the test instance x using different labels. As the correct label of x is not known to the method, we try all possibilities. More specifically, to classify an instance x, the method builds k

trees, where k is the number of possible labels for the target attribute. For each label, we insert x in the training set with that label and induce a decision tree. In the end, the final prediction and its certainty for x are obtained by combining the prediction of all the trees.

To combine the predictions, different strategies can be used. The one evaluated in this chapter (Equation 6.4) first calculates the prediction estimations for each tree by applying the Laplace smoothing (with $\alpha = 1$) on the class distribution; and then averages over the predicted values. With this smoothing procedure, we avoid that small leaves have too much effect in the final prediction, by adding some weight in the way we combine the trees' predictions. In Equation 6.4, $\mathsf{Pred}(c)$ is the prediction for (probability of) class c, L_i is the leaf node responsible for the prediction in tree T_{c_i}, $|T|$ is the number of trees (note that $|T|$ equals the number of classes), $|L_i(c)|$ is the number of instances belonging to class c within L_i, and $|L_i(\neg c)|$ is the number of instances belonging to a different class within L_i.

$$\mathsf{Pred}(c) = \frac{\sum_{i=1}^{|T|} \frac{|L_i(c)|+1}{|L_i(c)|+|L_i(\neg c)|+|T|}}{|T|} \tag{6.4}$$

Alternatively to the aforementioned strategy, one could simply average the prediction of the different trees without applying the Laplace Smoothing (Equation 6.5). Another possible strategy (Equation 6.6) could be to first combine the leaf nodes responsible for the prediction, considering all trees, and then calculate the prediction certainty based on the class distribution in the combined leaf. In fact, during the development and fine-tuning of our method, we have investigated the three aforementioned strategies to combine the predictions. To that aim, we used 6 UCI datasets that we separated for validation purposes.[4] As the strategy that uses the Laplace smoothing presented the best results on the validation datasets, we chose it to be further evaluated on the 48 datasets used in the evaluation.

$$\mathsf{Pred'}(c) = \frac{\sum_{i=1}^{|T|} \frac{|L_i(c)|}{|L_i(c)|+|L_i(\neg c)|}}{|T|} \tag{6.5}$$

$$\mathsf{Pred''}(c) = \frac{\sum_{i=1}^{|T|} |L_i(c)|}{\sum_{i=1}^{|T|} |L_i(c)| + |L_i(\neg c)|} \tag{6.6}$$

[4]These are different datasets than those considered in the experimental section of this chapter. For a list of all datasets used in the development/fine-tuning and evaluation of our method, see Appendix C.

The validation of our method also showed that our combination strategy benefits from pruning. In our strategy, pruning is important to avoid predictions from very small leaves. These leaves might be created to cover the test instance for the cases where it receives a wrong label. Thus, we can say that pruning avoids "overfitting" on the wrong label. This conclusion is, however, based on experiments where only one pruning procedure was considered. More specifically, we used the default pruning procedure in the decision tree learner Clus (see Section 2.2.2), on top of which we implemented our method. This pruning procedure is an implementation of the procedure used by the decision tree learner C4.5 [102]. The investigation of the effect of different pruning procedures in our strategy is an interesting venue for future work.

Pseudocode

The following pseudocode shows the described algorithm. More specifically, for every possible label $c \in C$, it calculates the estimated expectation that an instance x belongs to that class. Note that, as we have a double loop in the procedure CombinePredictions, we need two variables (i and j) to iterate over the k possible labels in C. We use the variable j in the outer loop so that the formula to combine the trees' predictions is consistent with the one we showed in Equation 6.4.

This procedure works fine in many cases, but sometimes an additional modification of the tree induction procedure LearnTree is needed. Trees typically handle continuous attributes by comparing them with a threshold (see, e.g., Figure 6.2). This threshold is put between two values belonging to instances with different class values. If we add a test instance to the training set, this introduces an additional possible threshold. This can have undesired effects, as shown in Figure 6.5. In this example, the instance x, which is represented in the figure as a circle, is a positive instance. As can be seen in the figure, x is far enough from the negative class (so that a standard decision tree would not have problems classifying it correctly), but, among all the positive instances, x is the closest to the negative ones. If we allow our method to use the attribute values of x, it will always choose a decision boundary that perfectly separates the instances depending on the label attributed to x (as shown in Figures 6.5.b and 6.5.c). This would lead our method to conclude that x is a difficult case to be classified, while actually it is not. To avoid this, the attribute values of x are not used when the learning method determines the possible split (test) values. However, they are still used when determining the heuristic value (information gain) of these possible splits.

ObtainPrediction

// **Input:** Training data D, set of possible classes C, test instance x
// **Output:** Vector $\langle pred_1,...,pred_k\rangle$, with k the number of classes and
// $pred_i \in [0,1]$ for all $1 \leq i \leq k$

for each possible class $c_i \in C$ **do**
    Add labeled instance (x, c_i) to D
    Learn decision tree T_{c_i} on the resulting data
CombinePredictions of $\boldsymbol{T} = \langle T_{c_1},T_{c_2},...,T_{c_k}\rangle$

CombinePredictions

// **Input:** Learned trees $\boldsymbol{T}$, set of possible classes C
// **Output:** Vector $\langle pred_1,...,pred_k\rangle$, with k the number of classes and
// $pred_i \in [0,1]$ for all $1 \leq i \leq k$

for each possible class $c_j \in C$ **do**
    $\mathsf{Pred}(c_j) := 0.0$
    for each tree $T_{c_i} \in \boldsymbol{T}$ **do**
        L_i is the leaf in T_{c_i} responsible for the prediction of x
        $|L_i(c_j)|$ is the number of instances in L_i that belong to c_j
        $|L_i(\neg c_j)|$ is the number of remaining instances in L_i
        $\mathsf{Pred}(c_j) := \mathsf{Pred}(c_j) + \frac{|L_i(c_j)|+1}{|L_i(c_j)|+|L_i(\neg c_j)|+|T|}$
    $\mathsf{Pred}(c_j) := \frac{\mathsf{Pred}(c_j)}{|T|}$

Figure 6.4: Pseudocode to obtain the prediction for an instance x.

6.3.3 Example of the calculations

In this section, we illustrate the calculations performed by our method, step by step; we consider the same example we used in Section 6.3.1 (see Figure 6.2), where we want to classify instance x_1, whose actual class is *Iris-virginica*.

To perform the prediction certainty calculations for x_1, we induce three decision trees, since Iris has three possible classes: *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica*. For each one of the trees that we will induce, x_1 will be labeled with one of the class values. For example, to induce the tree for which x_1 is labeled as *Iris-setosa*, we do the following. We first labeled x_1 as belonging to class

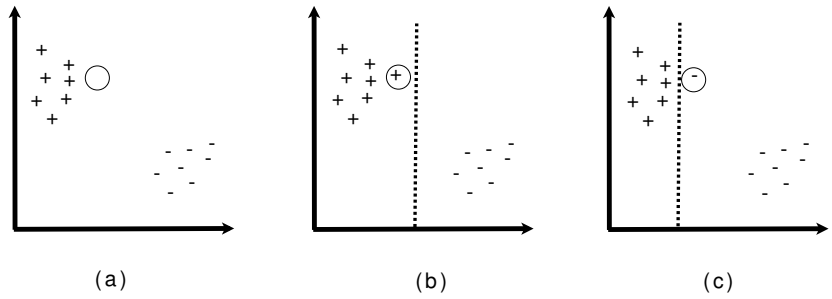


Figure 6.5: Undesired effects of using a test instance when constructing decision boundaries.

Iris-setosa, add it to the training example and induce the tree. Then, we check the class distribution of the leaf node where x_1 is placed. Table 6.1 shows the class distribution of the leaf node responsible for the prediction of x_1 for the three trees.

Table 6.1: Example calculations: class distribution of the leaf node responsible for the prediction of instance x_1 for the three induced trees. Each row of the table corresponds to one leaf node.

Given label	Class distribution		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	1	2	3
Iris-versicolor	0	49	1
Iris-virginica	0	2	4

Now we need to combine the prediction of the different trees. To that aim, we apply the combination procedure represented in Equation 6.4. As this procedure uses probabilities corrected with the Laplace smoothing, we first illustrate the smoothing calculation: we show the Laplace smoothing calculation for the first tree (i.e., the one for which we labeled x_1 as *Iris-setosa*) next. The notation $p_x(c_1|(x, c_2))$ refers to the probability that x has label c_1, given that the tree was induced with x labeled as c_2; for ease of notation we abbreviate *Iris-setosa*, *Iris-versicolor*, and *Iris-virginica* as SE, VE and VI, respectively. The resulting probabilities for all trees are shown in Table 6.2.

$$p_{x_1}(SE|(x_1, \mathsf{SE})) = \frac{1+1}{6+3} = 0.222$$

$$p_{x_1}(VE|(x_1, \mathsf{SE})) = \frac{2+1}{6+3} = 0.333$$

$$p_{x_1}(VI|(x_1, \mathsf{SE})) = \frac{3+1}{6+3} = 0.445$$

Table 6.2: Example calculations: probability distribution for the leaves shown in Table 6.1 after applying the Laplace smoothing.

Given label	Class distribution		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	0.222	0.333	0.445
Iris-versicolor	0.019	0.943	0.038
Iris-virginica	0.111	0.333	0.556

To obtain the final predictions, we combine the probabilities corrected with the Laplace smoothing for every class (Equation 6.4). We show the calculation for class *Iris-setosa* next; note that the calculation corresponds to averaging the column corresponding to *Iris-setosa* in Table 6.2. The final predictions are shown in Table 6.3.

$$\mathsf{Pred}(SE) = \frac{p_{x_1}(SE|(x_1, \mathsf{SE})) + p_{x_1}(SE|(x_1, \mathsf{VE})) + p_{x_1}(SE|(x_1, \mathsf{VI}))}{3} =$$

$$\frac{0.222 + 0.019 + 0.111}{3} = 0.117$$

Table 6.3: Example calculations: final predictions

Iris-setosa	Iris-versicolor	Iris-virginica
0.117	0.537	0.346

Note that according to the class distribution displayed in Table 6.3, *Iris-Versicolor* is the predicted class. Even though the prediction is not correct, our procedure was able to assign more uncertainty to this prediction. As shown in Figure 6.1, a standard decision tree learner assigns a very high certainty to this wrong prediction (98%).

6.4 Empirical evaluation

In this section, we present an extensive evaluation of the proposed method, which we implemented as an extension of the decision tree learner Clus (see Section 2.2.2); we call it Clus-TPCE (transductive prediction certainty estimation).

6.4.1 Experimental setup

In total we use 48 randomly selected UCI datasets [7] in the experiments. For the datasets with no pre-defined test set, we use leave-one-out validation. For the list of datasets used in the experiments, see Appendix C.

With these experiments we want to answer the following question: "Does the proposed method yields better prediction certainty estimates than a standard decision tree"; for this comparison we use the original version of Clus, which we refer to as Clus-Orig. One could argue, however, that this comparison is not entirely fair since our method uses multiple trees to make the final prediction, and it is known that ensembles tend to yield better estimates than single trees [101]. Therefore, to ensure that an improvement of Clus-TPCE is not simply due to "ensemble effects", we also compare to standard bagging (Clus-Ens) with the same number of trees as we use for Clus-TPCE. For all methods, we use information gain as the splitting criterion. For Clus-TPCE and Clus-Orig we use pruning. As ensemble methods often do not use pruning, we use unpruned trees for Clus-Ens.

As discussed in Section 6.2.1, prediction certainty can be interpreted and evaluated in different ways. For this reason, we evaluate the experimental results as (a) probability estimates, (b) ranking estimates, and (c) reliability estimates. For the sake of completeness, we also report the accuracy of the results.

For the reliability estimation evaluation, we include the results for the procedure proposed by Kukar and Konenko [75], which was described in Section 6.2.2; we implemented this procedure to calculate the reliability estimation of the predictions given by the original version of Clus, and we call it Clus-K&K.

For each comparison, additionally to the results themselves, we also report the p-value of a two-sided Wilcoxon signed-rank. With this test we verify the hypothesis that the method with the largest number of wins, in terms of the evaluation measure in consideration, is statistically superior to the other one.

6.4.2 Evaluating the accuracy of the predictions

We first evaluate the accuracy of the predictions. The results are shown in Figure 6.6 and summarized in Tables 6.4 and 6.5.

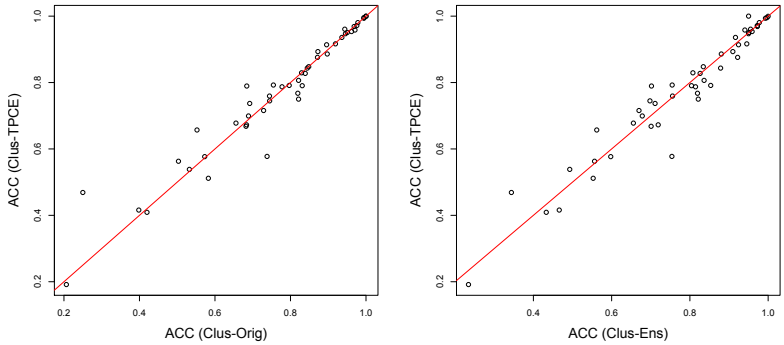


Figure 6.6: Results in terms of accuracy: "Clus-TPCE vs. Clus-Orig" (left) and "Clus-TPCE vs. Clus-Ens" (right).

Table 6.4: Comparison of the results in terms of accuracy. We show the number of wins for each method in the pairwise comparisons "Clus-TPCE vs. Clus-Orig" and "Clus-TPCE vs. Clus-Ens", along with the p-value resulting from a two-sided Wilcoxon signed-rank test.

Clus-TPCE vs. Clus-Orig			Clus-TPCE vs. Clus-Ens		
Clus-TPCE	Clus-Orig	Ties	Clus-TPCE	Clus-Ens	Ties
22	23	3	23	25	0
p-value = 0.8026			p-value = 0.5892		

Regarding accuracy, Clus-TPCE obtains 22/3/23 wins/ties/losses compared to Clus-Orig, and 23/0/25 wins/ties/losses compared to Clus-Ens. The average accuracy is very similar for the three methods, with Clus-TPCE being slightly better than Clus-Orig and slightly worse than Clus-Ens. These results show that the proposed method yields equally accurate results.

Table 6.5: Average accuracy for Clus-Orig, Clus-TPCE and Clus-Ens.

Accuracy		
Clus-Orig	Clus-TPCE	Clus-Ens
0.773	0.777	0.780

6.4.3 Evaluating probability estimation

In this section, we evaluate the results in terms of probability estimation using the Brier score. The results are shown in Figure 6.7 and summarized in Tables 6.6 and 6.7.

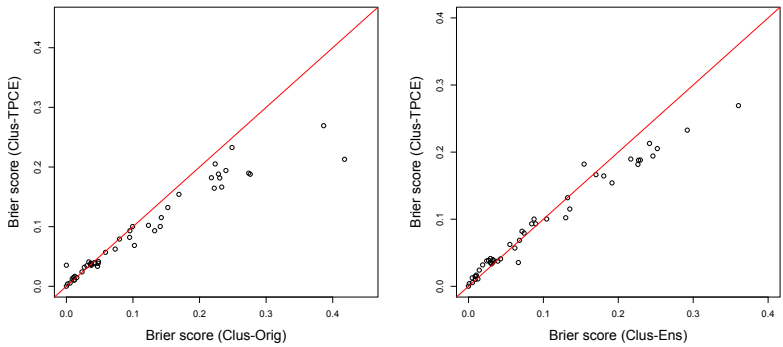


Figure 6.7: Results in terms of the Brier score: "Clus-TPCE vs. Clus-Orig" (left), and "Clus-TPCE vs. Clus-Ens" (right).

Table 6.6: Comparison of the results in terms of the Brier score. We show the number of wins for each method in the pairwise comparisons "Clus-TPCE vs. Clus-Orig" and "Clus-TPCE vs. Clus-Ens", along with the p-value resulting from a two-sided Wilcoxon signed-rank test.

Clus-TPCE vs. Clus-Orig			Clus-TPCE vs. Clus-Ens		
Clus-TPCE	Clus-Orig	Ties	Clus-TPCE	Clus-Ens	Ties
36	12	0	21	27	0
p-value < 0.0001			p-value $= 0.5686$		

Clus-TPCE obtains 36/0/12 wins/ties/losses compared to Clus-Orig, and a smaller average Brier score. When compared to Clus-Ens, Clus-TPCE obtains

Table 6.7: Average Brier score for Clus-Orig, Clus-TPCE, and Clus-Ens.

Brier score		
Clus-Orig	Clus-TPCE	Clus-Ens
0.109	0.087	0.096

a smaller number of wins (21 wins against 27 wins for Clus-Ens), but has a smaller average Brier score. Interestingly, for the cases where both Clus-TPCE and Clus-Ens have a larger Brier score, the advantage is always in favor of Clus-TPCE (see Figure 6.7). For the cases with low scores, the results are in favor of Clus-Ens, but with a smaller difference. This explains why Clus-TPCE has a smaller average Brier score, even though it has a smaller number of wins.

6.4.4 Evaluating ranking estimation

We now compare the methods regarding their ranking ability. We generate a ROC curve for each class against all the other ones and calculate the AUC for that curve; then, we average all AUC values to obtain the final AUC. The results are shown in Figure 6.8 and summarized in Tables 6.8 and 6.9.

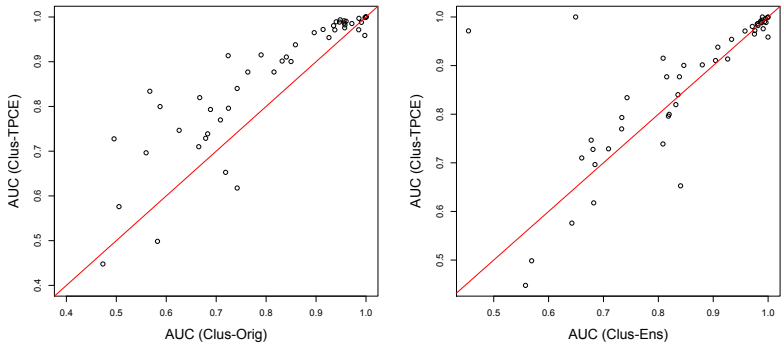


Figure 6.8: Results in terms of AUC: "Clus-TPCE vs. Clus-Orig" (top left), and "Clus-TPCE vs. Clus-Ens" (right).

The results show that Clus-TPCE has a better ranking ability than the other methods. It obtains 37/3/8 wins/ties/losses compared to Clus-Orig and 29/1/18 wins/ties/losses compared to Clus-Ens. Note that these results are more in favor of Clus-TPCE than for the probability estimation evaluation. This

Table 6.8: Comparison of the results in terms of AUC. We show the number of wins for each method in the pairwise comparisons "Clus-TPCE vs. Clus-Orig" and "Clus-TPCE vs. Clus-Ens", along with the p-value resulting from a two-sided Wilcoxon signed-rank test.

Clus-TPCE vs. Clus-Orig			Clus-TPCE vs. Clus-Ens		
Clus-TPCE	Clus-Orig	Ties	Clus-TPCE	Clus-Ens	Ties
37	8	3	29	18	1
p-value < 0.0001			p-value $= 0.1706$		

Table 6.9: Average AUC for Clus-Orig, Clus-TPCE, and Clus-Ens.

AUC		
Clus-Orig	Clus-TPCE	Clus-Ens
0.814	0.868	0.849

is not completely unexpected. Clus-TPCE tends to shift the probability distribution output for some instances away from 1 (or 0), and towards the uniform probability (0.5, in case of a binary problem). This effect is stronger for the cases for which the method finds evidence that there is a high degree of uncertainty associated to their predictions. However, as the Brier score assumes that the method should ideally report a probability of 1 for the true class and 0 for the other classes, having a number of predictions with probabilities closer to the uniform probability will have a negative effect on the final Brier score.

With these results, we can conclude that ranking estimation is a better application of our method than probability estimation.

6.4.5 Evaluating reliability estimation

Finally, we evaluate how well Clus-TPCE estimates the reliability of a prediction. To that aim, we consider the probability output for the predicted class as the reliability that the prediction is correct. We apply the same procedure for Clus-Orig and Clus-Ens. For Clus-K&K, we use the procedure proposed by Kukar and Kononenko [75].

To evaluate the reliability estimates of the four methods, we use the same evaluation framework used by Kukar and Kononenko [75], with the difference that we use AUC instead of information gain; we call it AUC reliability (see

Section 6.2.1). The results are shown in Figure 6.9 and summarized in Tables 6.10 and 6.11.

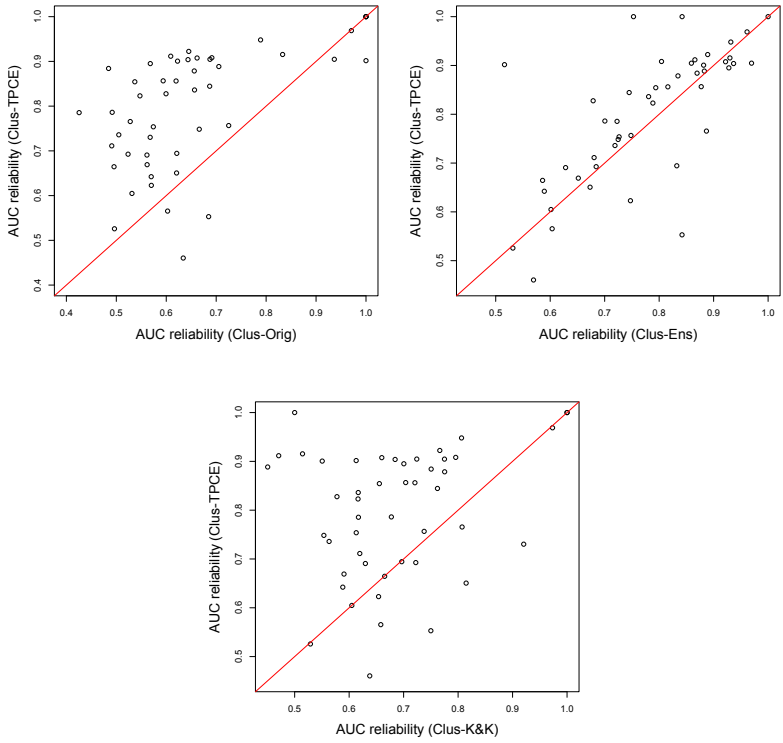


Figure 6.9: Results in terms of AUC reliability: "Clus-TPCE vs. Clus-Orig" (top left), "Clus-TPCE vs. Clus-Ens" (top right), and "Clus-TPCE vs. Clus-K&K" (bottom).

Clus-TPCE outperforms the other three methods in terms of reliability estimation: it obtains 39/3/6 wins/ties/losses compared to Clus-Orig, 33/1/14 wins/ties/losses compared to Clus-Ens, and 33/2/13 wins/ties/losses compared to Clus-K&K. Furthermore, Clus-TPCE obtains the largest average AUC reliability, and the statistical tests indicate that the difference in the results is statistically significant. These results confirm the good performance of our method in ranking probabilities.

Table 6.10: Comparison of the results in terms of AUC reliability. We show the number of wins for each method in the pairwise comparisons "Clus-TPCE vs. Clus-Orig", "Clus-TPCE vs. Clus-Ens", and "Clus-TPCE vs. Clus-K&K", along with the p-value resulting from a two-sided Wilcoxon signed-rank test.

Clus-TPCE vs. Clus-Orig			Clus-TPCE vs. Clus-Ens			Clus-TPCE vs. Clus-K&K		
Clus-TPCE	Clus-Orig	Ties	Clus-TPCE	Clus-Ens	Ties	Clus-TPCE	Clus-K&K	Ties
39	6	3	33	14	1	33	13	2
p-value < 0.0001			p-value $= 0.0209$			p-value $= 0.0001$		

Table 6.11: Average AUC reliability for Clus-Orig, Clus-TPCE, Clus-Ens, and Clus-K&K.

AUC reliability			
Clus-Orig	Clus-TPCE	Clus-Ens	Clus-K&K
0.6498	0.7969	0.7681	0.6836

6.5 Conclusions

In this chapter, we proposed a method for estimating prediction certainty. The new method was implemented as an extension of the decision tree learner Clus, but the ideas investigated in this chapter can also be applied to other machine learning methods, in particular those where the label of a single example can influence the learned model.

Our new method builds a decision tree for an input data consisting of the training data plus the instance to be classified, which is labeled with one of the possible class values. This procedure is repeated for every class, and in the end all induced trees are compared. This comparison allows us to identify instances that might present difficulties to be correctly classified, and to attribute some uncertainty to their predictions. We evaluated our method on 48 UCI datasets, and compared it to the original Clus and to standard bagging. The results showed that the new method yields better ranking and reliability estimates than the other methods. Regarding probability estimation, the proposed method yields better estimates than the original method and comparable estimates to the ensemble method. We also compared to the method by Kukar and Kononenko [75] with respect to reliability estimation, and show that our method produces better estimates.

Since our method is complementary to the other methods suggested in the literature (see Section 6.2.2), they can be easily combined: simply use those methods instead of Clus to learn multiple trees for different versions of the test instances.

Note that our method makes predictions for only one instance at a time, which increases its computational cost. This raises the question if the method can be extended in order to be able to perform the whole process once for a whole batch of unseen instances. A straightforward extension of the method consists of generating the same number of trees as the number of possible labels (once for all instances), where for every tree each instance to be classified receives a random label, with the constraint that the same instance will never receive the same label in two or more trees. This constraint assures that each instance receives each possible label once, allowing us to apply the same strategy to combine predictions used in this chapter. However, as each generated tree is not only subject to the influence of the labeling of a single instance (but a batch of instances instead), it is not trivial to analyze if the prediction obtained for an instance is a result of how that instance was labeled, and/or how the other instances were labeled. In fact, we have performed preliminary experiments with this extended method to test its ranking ability, and the results showed that it produces better results than a standard tree learner, but slightly worse results than standard bagging. Therefore, the extension of the proposed method remains for future research.

Chapter 7

Conclusions

In this chapter, we first summarize and discuss the most important results and contributions of this thesis. We then point out possible improvements in the methods we proposed. Finally, we discuss further application opportunities.

7.1 Thesis summary

The overall goal of this thesis is the development of algorithmic solutions for bioinformatics problems that involve analyzing biological sequences. We focused on the use of machine learning techniques to do so.

We first considered the **phylogenetic tree reconstruction** problem in Chapter 3. This problem involves inferring the most likely phylogenetic tree that explains the evolutionary history for a set of sequences. To tackle this problem, we proposed a method which is built on top of an existing decision tree learner (Clus). The method, which we call Clus-φ, builds a conceptual clustering tree using tests based on polymorphic positions of the aligned input sequences. To select the tests used in the tree building process, we designed a heuristic oriented to the phylogenetic context. One important feature of the phylogenetic trees generated by this strategy is that the identified tests give an evolutionary trace. This additional information can be used to investigate which mutations gave rise to different evolutionary lineages in the tree. We have shown that (1) our proposed heuristic is more suitable for phylogenetic tree reconstruction than standard decision tree heuristics, and (2) Clus-φ produces results with comparable quality to existing phylogenetic methods.

Second, we considered the **protein subfamily identification** problem in Chapter 4. In this problem, one is interested in finding subgroups of functionally closely related sequences within a protein family. This can be achieved by looking for clusters which are evolutionarily closely related. Therefore, this problem is related to the phylogenetic tree reconstruction problem of Chapter 3. To tackle this problem, we proposed a method that first builds a phylogenetic tree using Clus-φ and then extracts clusters from this tree by using a post-pruning procedure. We have shown that this strategy produces results comparable with those of a state-of-art method, with the advantage that it provides additional information about the predicted subfamilies. More specifically, the polymorphic positions used to build the tree provide a candidate list for functional sites, and can be used to classify new sequences into one of the predicted subfamilies.

The work presented in Chapter 3 and 4 points out the advantages of using top-down conceptual clustering in the context of biological applications, where bottom-up clustering strategies are predominantly used.

Third, we considered the problem of inferring **peptide identification of mass spectrometry data** in Chapter 5. To tackle this problem, we proposed a method called PIUS (Peptide Identification by Unbiased Search). PIUS searches for peptides against the full translation of a genome and it is specially suitable for the identification of naturally occurring peptides. In contrast to peptidomics methods that also allow a genome-wise search, PIUS does not reduce its search space based on *de novo* reconstructions. Instead, it exhaustively scans the genome. We have shown that this strategy produces better results than a non-exhaustive method. The main contribution of these results is that they point out the importance of alternative strategies (such as exhaustive search) to the filtering of the search space provided by *de novo* reconstructions. However, in the case of exhaustive search, the improvement in the results comes at a price, that is, the increase in computational cost. For this reason, in case of large scale experiments, we recommend PIUS to be used as a last resource tool, for spectra of good quality that cannot be identified by conventional database search methods.

Finally, as an additional contribution to the thesis, we considered a machine learning problem involving **prediction certainty estimation in decision trees** in Chapter 6. We presented a transductive procedure to estimate the certainty of a prediction. We have evaluated our proposed procedure under different definitions of prediction certainty. The results have shown that this procedure is more suitable for ranking and reliability estimation than for probability estimation. One extra contribution of this study is the detailed discussion of related work of prediction certainty estimation for decision trees that we presented in this chapter.

7.2 Discussion

The three biological problems that we investigated in this thesis illustrate how biological sequence analysis can be used to bridge the gap between the molecular information of an organism and its biology. In the context of mass spectrum interpretation, for example, we have shown that the analysis of the six-frame translation of the genome can be used to assist peptide identification.

If we compare the first two problems - phylogenetic tree reconstruction and protein subfamily identification - with the peptide identification problem, we can notice that they have different specificities, such as their final goal and the kind of data that is typically available for the task. Also, the computational methods that we propose are different. Indeed, in bioinformatics, computational solutions can differ considerably from one problem to another. Our method for peptide identification, PIUS, is one example of a solution that is specifically designed for a particular problem. On the other hand, some methods can be adapted to solve related problems. In the case of phylogenetic tree reconstruction and protein subfamily identification, we have shown that an extension of the clustering method proposed for the former led to a method for the latter.

Computational solutions not only differ from one task to another. We can also have a variety of methods to solve the same problem. One important advantage of considering machine learning methods is that they can automatically find/select interesting information in the data that is given as input for a certain task. In the case of protein subfamily identification, for example, our proposed method finds which positions are the most suitable, according to a certain criterion, to partition a protein family into subfamilies. Moreover, when we use symbolic learning, such as decision tree learning, the resulting model (or hypothesis) can be easily understood and validated by a domain expert. This allows biologists to understand the reasoning behind the output of such a model, instead of considering it as a "black box" that simply outputs predictions when data is given as input.

While the proposed methods for the phylogenetic tree reconstruction and protein subfamily identification problems fit the machine learning framework, the same does not hold entirely for our method for peptide identification. This method is in its essence a search algorithm, even though we can look at its pruning procedure as a learning process that automatically acquires information to recognize potentially good candidates based on their prefix sequences. In the next section, we discuss one example of how this method could benefit from machine learning.

7.3 Possible improvements

We have argued that conceptual clustering methods can infer informative cluster hierarchies of biological sequences by using polymorphic positions. Our method for protein subfamily identification exploits this important advantage to identify a candidate list of functionally important sites. Combined with the good results in terms of predicted subfamilies, this feature provides an important argument in favor of the use of our method. One possibility to exploit this feature further is to incorporate (to the method) a user-friendly interface for the visual analysis of candidate functional sites along with the clustering results. A direct benefit of such a visualization tool would be to encourage the use of our method by domain experts. Moreover, this could also open opportunities to improve the method itself by using, for example, visual analytics. This can be defined as "the science of analytical reasoning supported by interactive visual interfaces" [125, 72]. This kind of reasoning has been shown to be useful in many domains, including biological data analysis (e.g., [97]).

We have shown that PIUS can be used as a last resource tool for the identification of naturally occurring peptides. However, there are certain cases that would be missed by PIUS in its current version. First, as PIUS only considers modified peptides that can be derived from a user-defined set of post-translation modifications (PTMs), it does not support discovery of PTMs of unknown mass. Second, PIUS does not consider solutions originating from splicing processes. Third, it does not account for polymorphism in the genome (i.e., that there could be peptides generated from a slightly different genome than the one used in the translation). Dealing with these issues is challenging, since it affects the computational cost of the search and increases the likelihood of returning false-positive solutions. One idea to deal with false positives, for example, is to consider genomic information (such as codon usage and conservation rate) to disregard part of the solutions. This strategy can, however, introduce some bias in the results, since it is based on what is currently known about genomes, and eliminate a potentially correct solution from the list of candidates. Another idea is to use machine learning techniques to learn fragmentation patterns from known naturally occurring peptides and use this information to design a more discriminative scoring function.

Machine learning can also be used to define the sets of PTMs to be considered in the search performed by PIUS. As we mentioned in Section 5.3.5, clustering can be used in this context to identify sample-specific PTMs. Conceptual clustering can thus also be used to tackle this task, provided that an adequate descriptive language for mass spectra is used. One idea could be to enumerate all possible peak values occurring in the spectra to be clustered, and use these values as binary descriptive features, where a spectrum has value one for a feature when

it has a peak that matches (within an error tolerance) the peak value of that feature, and zero otherwise. This strategy would allow us to look for the most discriminative peaks when clustering the spectra.

Regarding our alternative procedure for estimating prediction for decision trees, one limitation is the computational cost associated to making predictions for one sequence at a time. Ideally, one would like to be able to obtain estimates for several instances at once. The extension of our procedure in that direction is therefore a possible improvement of our procedure in terms of computational efficiency. Moreover, the combination of our procedure with existing ones could also lead to improvement in the quality of the results.

7.4 Further application opportunities

In bioinformatics, there are many problems involving sequence analysis that can benefit from the quality and comprehensibility of the results provided by machine learning strategies such as predictive decision trees and conceptual clustering. Consider, for example, the problem of identifying transposable elements (TEs), which are DNA sequences that can move and duplicate, autonomously or with the assistance of other elements, within genomes.

TEs make up a large portion of the DNA in eukaryotic organisms and are responsible for the many variations within and across species. These DNA sequences are classified in a hierarchical system that includes (in hierarchical order) the levels of class, subclass, order, superfamily, family and subfamily [138].

In an on-going project between our research group at the KU Leuven and researchers from two universities in Brazil - São Paulo State University and University of São Paulo, we are currently investigating the use of predictive decision trees to identify TEs and to classify them on the superfamily level.

Classification of TEs on more specific levels is, however, more challenging [80]. A strategy to tackle this problem consists of exploiting evolutionary relationships among TEs. This offers an interesting application opportunity for the divisive clustering approach we used for protein subfamily identification.

Appendix A

Heuristic function used by Clus-φ

In this appendix, we provide more details about the calculation of the heuristic function used by Clus-φ, our method for phylogenetic tree reconstruction presented in Chapter 3. This function is used to guide the top-down induction of the tree and its goal is to minimize the final total branch length of the tree.

As explained in Section 3.3, we need two different formulations for this function: one for splitting the root node and one for splitting the other internal nodes. We explain these two formulations in the first two sections of this appendix, respectively. We finish the appendix with an example that illustrates the calculations performed by Clus-φ when choosing the best split.

A.1 Heuristic function to split the root node

Clus-φ starts from a single cluster containing all sequences for which we want to infer the phylogenetic tree. Using the same reasoning as for the neighbor joining algorithm (see Section 3.2.2), we can consider this initial stage as a fully unresolved star-shaped tree (Figure A.1.a). When the root node is split, we obtain a tree such as the tree T shown in Figure A.1.b.

To calculate the total branch length of T, we need to calculate the total branch length of the subtree in the left (T_L), the branch length L_{XY}, and the total branch length of the subtree in the right (T_R).

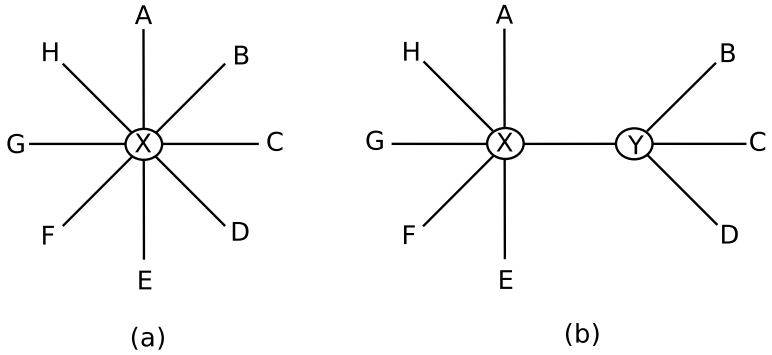


Figure A.1: Illustration of the first split performed by Clus-φ: (a) fully unresolved star-shaped tree; (b) tree given by the split of the root node.

T_L and T_R are both a star-shaped topology. As we showed in Chapter 3, the total branch length of a star-shaped tree T^* can be calculated by the following formula:

$$TBL(T^*) = \frac{1}{|T^*| - 1} \sum_{\substack{o_i, o_j \in T^* \\ i<j}} D_{o_i o_j}. \tag{A.1}$$

The calculation of L_{XY} is given by

$$L_{XY} = \frac{1}{|T_L||T_R|}[\sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} - |T_R| \sum_{o_i \in T_L} L_{o_i X} - |T_L| \sum_{o_i \in T_R} L_{o_i Y}].$$

This formula is a generalization of the one showed in Equation 3.1 (Section 3.2.2), which has $|T_L| = 2$, $|T_R| = |T| - 2$, and $|T|$ the number of sequences in the tree for which we want to calculate the total branch length.

To be able to calculate the second and third summations of the formula, we need to replace the branch lengths $L_{o_i X}$ (with $o_i \in T_L$) and $L_{o_i Y}$ (with $o_i \in T_R$) by the pairwise distances $D_{o_i o_j}$ between the sequences within T_L and within T_R, respectively; these distances are given by the distance matrix. We can do it by replacing $\sum_{o_i \in T_L} L_{o_i X}$ by $\frac{1}{|T_L|-1} \sum_{o_i, o_j \in T_L i<j} D_{o_i o_j}$, and $\sum_{o_i \in T_R} L_{o_i Y}$ by $\frac{1}{|T_R|-1} \sum_{o_i, o_j \in T_R i<j} D_{o_i o_j}$. With these substitutions, we can rewrite the calculation of L_{XY} as follows:

$$L_{XY} = \frac{1}{|T_L||T_R|}[\sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} - \frac{|T_R|}{|T_L|-1} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \frac{|T_L|}{|T_R|-1} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}].$$

Adding the calculations for $TBL(T_L)$, $TBL(T_R)$, and L_{XY} together, and making the right rearrangements, we obtain the following function:

$$TBL(T\{T_L, T_R\}) = \frac{1}{|T_L||T_R|} \sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} + \frac{1}{|T_L|} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + \frac{1}{|T_R|} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}. \quad \text{(A.2)}$$

We can eliminate unnecessary calculations from this formula by using the following equality:

$$\sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} = \sum_{\substack{o_i, o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} - \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}, \quad \text{(A.3)}$$

More specifically, we replace the first summation of Equation A.2 by the right side of Equation A.3. With the right rearrangements, we obtain the following formula:

$$TBL(T\{T_L, T_R\}) = \frac{1}{|T_L||T_R|}(\sum_{\substack{o_i, o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} + (|T_R|-1) \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + (|T_L|-1) \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}).$$

This formula is more efficient than the previous one because its first summation is constant for the node being split. This allows us to calculate this summation only once when choosing the best split for a node.

A.2 Heuristic function to split the other internal nodes

For splitting the other internal nodes, a more complex heuristic function is needed, since the particular split influences the length of other branches in the tree, and hence, the total branch length. In Figure A.2, we give an example of such a split, where the sequences directly connected to X are split into two subtrees. To keep the same notation that we used in the previous section, we use the symbols T_L and T_R to denote the two resulting subtrees. The subtree containing the remaining sequences is denoted by T_O.

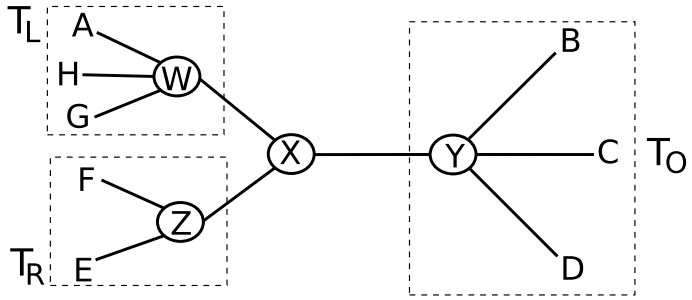


Figure A.2: Illustration of the tree topology considered by Clus-φ when splitting non-root nodes.

To calculate the total branch length of the tree T in Figure A.2, we need to calculate the total branch length of the three star-shaped trees (T_L,T_R, and T_O), and the branch lengths L_{XW}, L_{XZ}, and L_{XY}. For the star-shaped trees, we use the same calculation we used in the previous section (see Equation A.1). Next, we show the calculations for the branch lengths L_{XW}, L_{XZ}, and L_{XY}.

To calculate the sum of L_{XW}, L_{XZ}, and L_{XY}, we calculate the sum of the branch lengths L_{WZ}, L_{YZ}, and L_{YW}, and divide the result by two, as follows:

$$L_{XW} + L_{XZ} + L_{XY} = \frac{1}{2}(L_{WZ} + L_{YZ} + L_{YW}) =$$

$$\frac{1}{2}(\frac{1}{|T_L||T_R|}[\sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} - \frac{|T_R|}{|T_L|-1} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \frac{|T_L|}{|T_R|-1} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}] +$$

$$\frac{1}{|T_O||T_R|}[\sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j} - \frac{|T_R|}{|T_O|-1} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j} - \frac{|T_O|}{|T_R|-1} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j}] +$$

$$\frac{1}{|T_L||T_O|}[\sum_{\substack{o_i \in T_L \\ o_j \in T_O}} D_{o_i o_j} - \frac{|T_O|}{|T_L|-1} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \frac{|T_L|}{|T_O|-1} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j}]) =$$

$$\frac{1}{2|T_L||T_R|} \sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} - \frac{1}{|T_L|(|T_L|-1)} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \frac{1}{|T_R|(|T_R|-1)} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} +$$

$$\frac{1}{2|T_O||T_R|} \sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j} - \frac{1}{|T_O|(|T_O|-1)} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j} + \frac{1}{2|T_L||T_O|} \sum_{\substack{o_i \in T_L \\ o_j \in T_O}} D_{o_i o_j}.$$

The total branch length of T is obtained by adding this calculation to the total branch length of the three star-shaped trees, as shown next:

$$TBL(T\{T_L, T_R, T_O\}) = L_{XW} + L_{XZ} + L_{XY} + TBL(T_L) + TBL(T_R) + TBL(T_O) =$$

$$\frac{1}{2|T_L||T_R|} \sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} - \frac{1}{|T_L|(|T_L|-1)} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} - \frac{1}{|T_R|(|T_R|-1)} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} +$$

$$\frac{1}{2|T_O||T_R|} \sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j} - \frac{1}{|T_O|(|T_O|-1)} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j} + \frac{1}{2|T_L||T_O|} \sum_{\substack{o_i \in T_L \\ o_j \in T_O}} D_{o_i o_j} +$$

$$\frac{1}{|T_L|-1} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + \frac{1}{|T_R|-1} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{1}{|T_O|-1} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j}.$$

Making the right rearrangements, we obtain the following formula:

$$TBL(T\{T_L, T_R, T_O\}) =$$

$$\frac{1}{2|T_L||T_R|} \sum_{\substack{o_i \in T_L \\ o_j \in T_R}} D_{o_i o_j} + \frac{1}{2|T_O||T_R|} \sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j} + \frac{1}{2|T_L||T_O|} \sum_{\substack{o_i \in T_L \\ o_j \in T_O}} D_{o_i o_j} +$$

$$\frac{1}{|T_L|} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + \frac{1}{|T_R|} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{1}{|T_O|} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j}.$$

Using the equality shown in Equation A.3, we obtain the following formula:

$$TBL(T\{T_L, T_R, T_O\}) =$$

$$\frac{1}{2|T_L||T_R|} \sum_{\substack{o_i, o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} - \frac{1}{2|T_L||T_R|} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} -$$

$$\frac{1}{2|T_L||T_R|} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{1}{2|T_O||T_R|} \sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j} + \frac{1}{2|T_L||T_O|} \sum_{\substack{o_i \in T_L \\ o_j \in T_O}} D_{o_i o_j} +$$

$$\frac{1}{|T_L|} \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} + \frac{1}{|T_R|} \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{1}{|T_O|} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j}.$$

Making the right rearrangements, we obtain the following formula:

$$TBL(T\{T_L, T_R, T_O\}) = \frac{1}{2|T_L||T_R|} (\sum_{\substack{o_i, o_j \in T_L \cup T_R \\ i<j}} D_{o_i o_j} + \quad (2|T_R| - 1) \sum_{\substack{o_i, o_j \in T_L \\ i<j}} D_{o_i o_j} +$$

$$(2|T_L| - 1) \sum_{\substack{o_i, o_j \in T_R \\ i<j}} D_{o_i o_j} + \frac{|T_R|}{|T_O|} \sum_{\substack{o_i \in T_O \\ o_j \in T_L}} D_{o_i o_j} + \frac{|T_L|}{|T_O|} \sum_{\substack{o_i \in T_O \\ o_j \in T_R}} D_{o_i o_j}) + \frac{1}{|T_O|} \sum_{\substack{o_i, o_j \in T_O \\ i<j}} D_{o_i o_j}.$$

As for the formula we derived in the previous section, the first summation of this formula is constant for the node being split and only needs to be calculated once. The last summation is also constant for the node being split, and as it is multiplied by a factor which is also constant (i.e., $\frac{1}{|T_O|}$), we do not need to compute this summation when choosing the best split.

A.3 Illustration of the calculations performed by the Clus-φ heuristic

In this section, we illustrate the calculations performed by the Clus-φ heuristic function with an example. We show the calculations for the split of the root node.

Consider the hypothetical tree topology shown in Figure A.3, which illustrates the evolutionary relationships among six sequences (A, B, C, D, E, and F), and its corresponding distance matrix, shown in Table A.1. The numbers in the figure correspond to branch lengths and they reflect the information given by the distance matrix.[1]

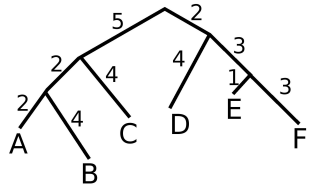


Figure A.3: Example: tree topology.

	A	B	C	D	E
B	6				
C	8	10			
D	15	17	15		
E	15	17	15	8	
F	17	19	17	10	4

Table A.1: Example: Distance Matrix.

Suppose that, in the first step of the tree reconstruction process, Clus-φ has to decide between splitting the set of sequences in two different ways: {(A,B,C),(D,E,F)}, which corresponds to the correct split according to the tree in Figure A.3, and {(A,E,F),(B,C,D)}. To be able to decide which split is the best one, Clus-φ uses its heuristic function to estimate the total branch length of the tree that would be generated by each one of the splits. We show the heuristic calculation for the two splits. For the sake of simplicity, we use Equation A.2, instead of its more efficient formulation.

The total branch length of the tree generated by the split {(A,B,C),(D,E,F)} is:

[1] In the scope of this thesis, we consider branch lengths as being proportional to the amount of character change between the nodes of the tree.

$$TBL(\{(A,B,C),(D,E,F)\}) =$$

$$\frac{1}{9}\sum_{\substack{o_i\in\{A,B,C\}\\ o_j\in\{D,E,F\}}} D_{o_io_j} + \frac{1}{3}\sum_{\substack{o_i,o_j\in\{A,B,C\}\\ i<j}} D_{o_io_j} + \frac{1}{3}\sum_{\substack{o_i,o_j\in\{D,E,F\}\\ i<j}} D_{o_io_j} =$$

$$16.37 + 8 + 7.33 = 31.66.$$

The total branch length of the tree generated by the split {(A,E,F),(B,C,D)} is:

$$TBL(\{(A,E,F),(B,C,D)\}) =$$

$$\frac{1}{9}\sum_{\substack{o_i\in\{A,E,F\}\\ o_j\in\{B,C,D\}}} D_{o_io_j} + \frac{1}{3}\sum_{\substack{o_i,o_j\in\{A,E,F\}\\ i<j}} D_{o_io_j} + \frac{1}{3}\sum_{\substack{o_i,o_j\in\{B,C,D\}\\ i<j}} D_{o_io_j} =$$

$$12.78 + 12 + 14 = 38.78.$$

As Clus-φ aims at minimizing the total branch length of the tree that will be finally obtained, it chooses the split that gives the smallest total branch length at each iteration. Therefore, in our example, Clus-φ chooses the split {(A,B,C),(D,E,F)}.

Note that the first term of Equation A.2 corresponds to the calculation performed to maximize the average inter-cluster distance, which is one of the heuristics that we investigated in Chapter 3. At a first glance, it might look counter-intuitive that the Clus-φ heuristic, which is a minimization heuristic function, uses the same calculation performed by a maximization heuristic function in its formula. Thus, one could wrongly deduce that Clus-φ tries to minimize the average inter-cluster distance, which would be a bad strategy to obtain a good clustering. However, a partition that gives a small value for the first term of the heuristic function can have subsets with a large intra-cluster difference,[2] leading to a large total branch length. This is the case for the second split we tested in our example - {(A,E,F),(B,C,D)}. This split gives a smaller value for the first term of the function in comparison with the first split (12.78 vs. 16.37); however, the larger values given by the other two terms lead to a larger total branch

[2]The second and the third terms of the heuristic function calculate intra-cluster differences.

length than the one given by the first split. There is, therefore, a trade-off between the value resulting for the first term of the Clus-φ heuristic function and the values resulting for the other two terms.

Appendix B

Protein subfamily identification - tree topologies

In this appendix, we show the tree topologies generated for the EXPERT dataset Enolase, in the context of protein subfamily identification (Chapter 4). Figures B.1, B.2, and B.3 show the edited tree topologies[1] obtained from the Clus-φ, SCI-PHY, and neighbor joining (NJ) trees, respectively. Figures B.4 and B.5 show the trees (after pruning) output by Clus-φ-ECC and SCI-PHY, respectively. For the ease of notation, we abbreviate the Enolase subfamilies as shown in Table 4.13 (Section 4.5.6), and we omit the tests identified for the internal nodes of the Clus-φ trees. For each tree leaf, we indicate the number of sequences per subfamily. Each color corresponds to a different subfamily.

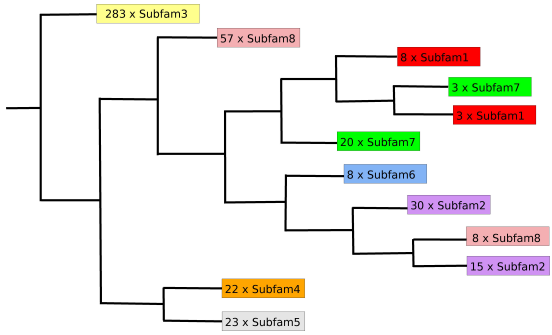


Figure B.1: Clus-φ edited tree for the EXPERT dataset Enolase.

[1] We defined edited trees in Section 4.4.1.

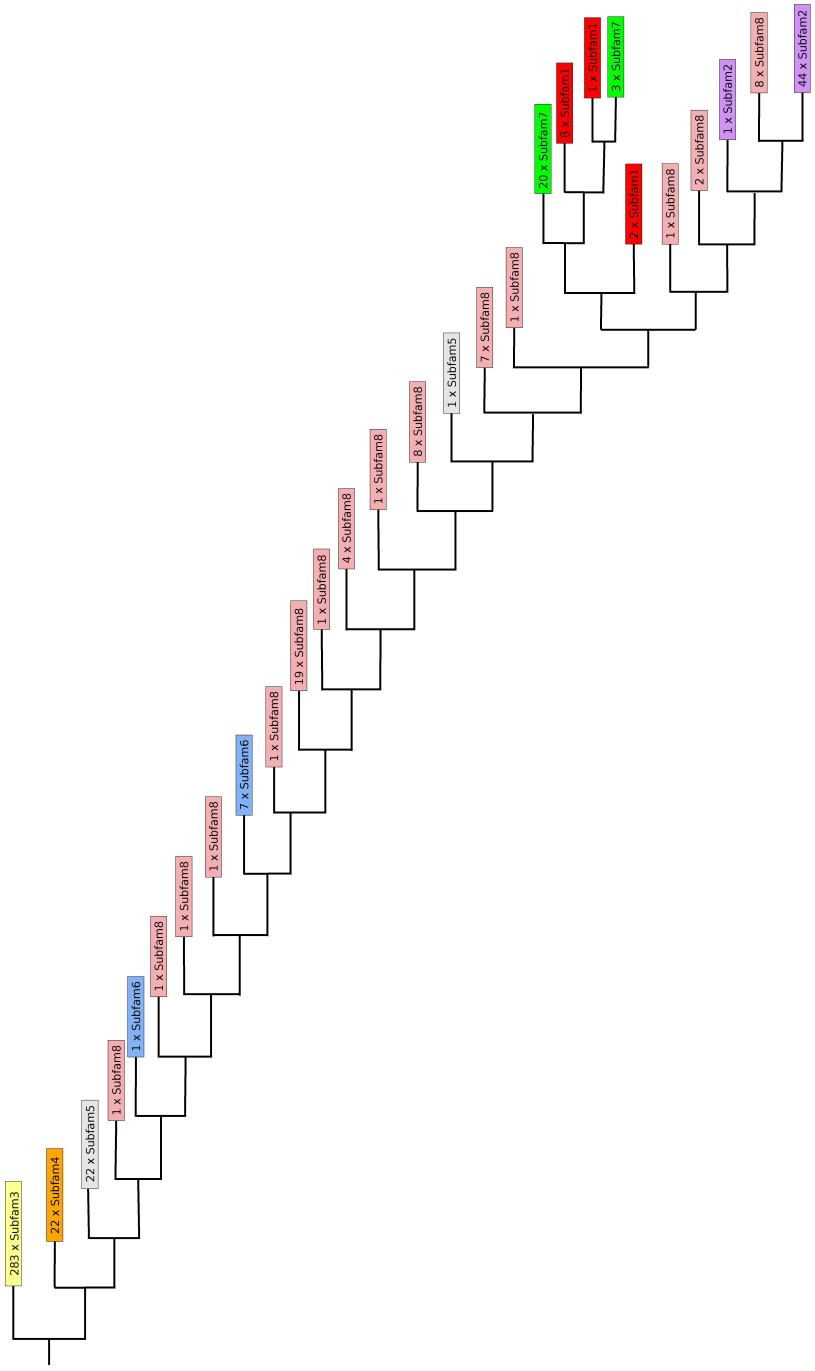


Figure B.2: SCI-PHY edited tree for the EXPERT dataset Enolase.

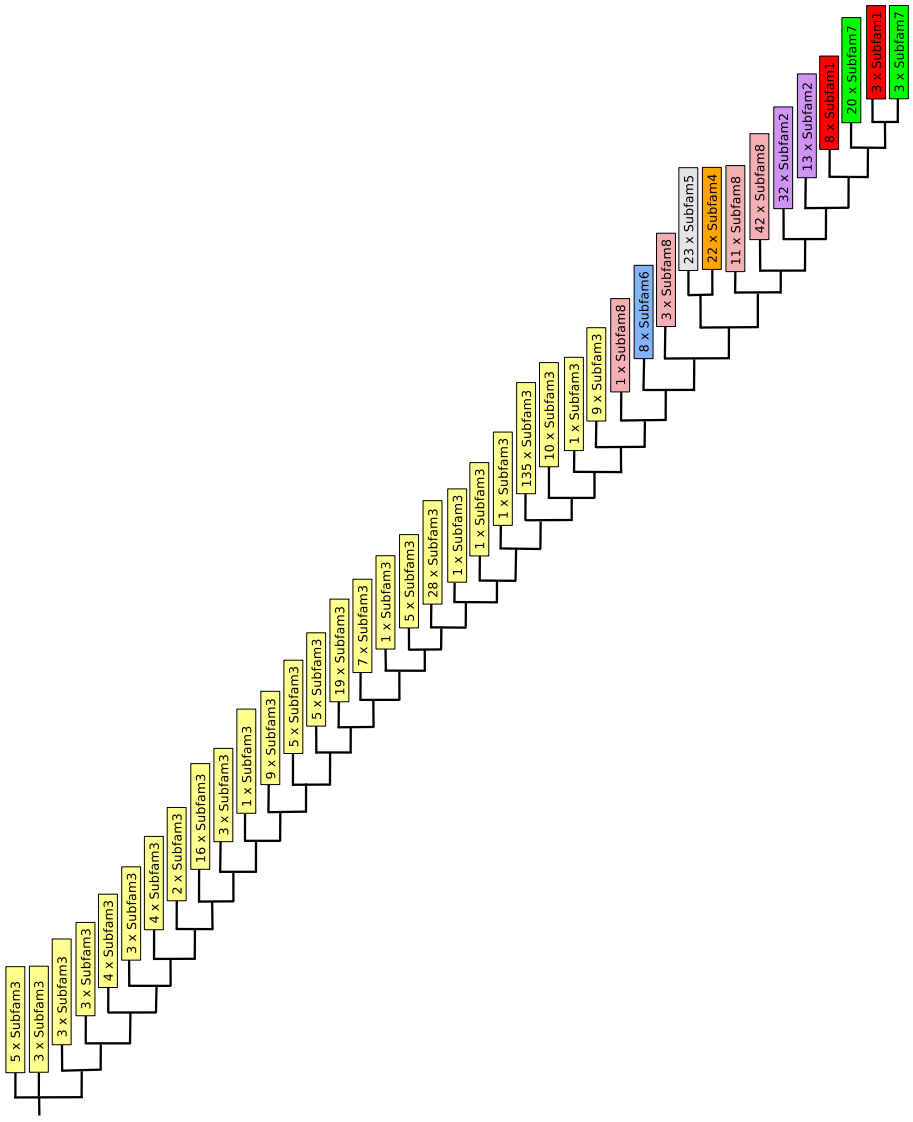


Figure B.3: NJ edited tree for the EXPERT dataset Enolase.

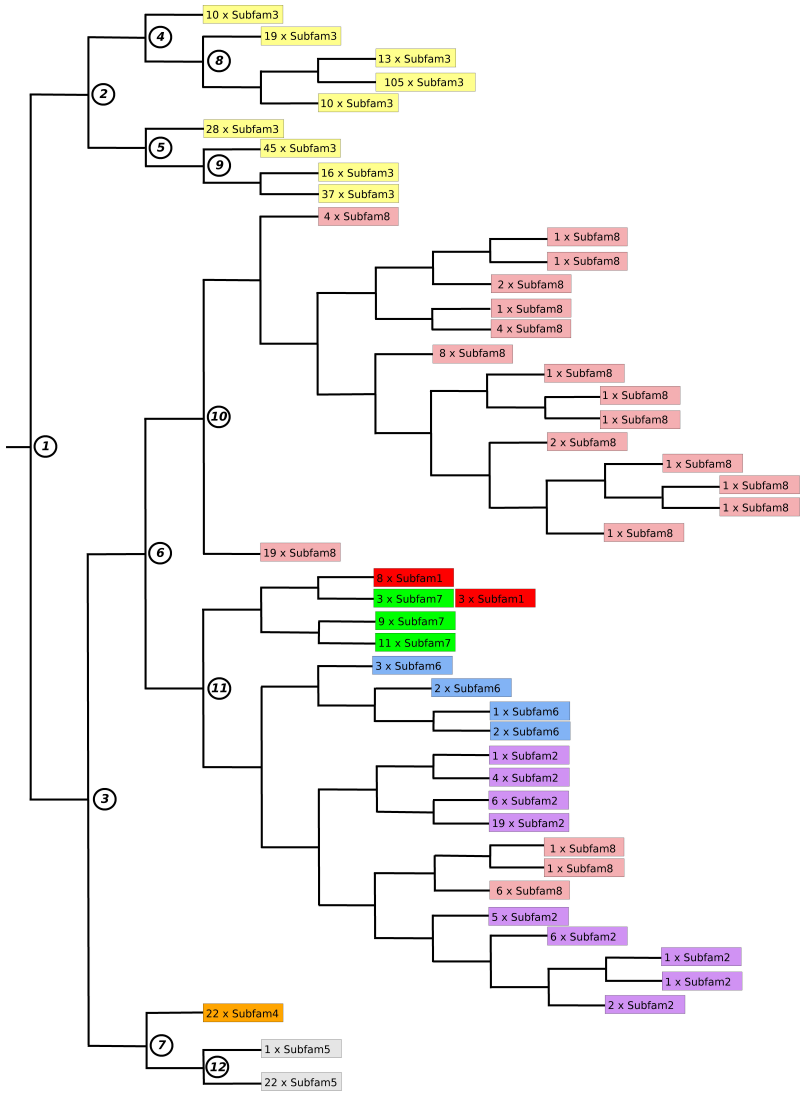


Figure B.4: Clus-φ-ECC tree for the EXPERT dataset Enolase. The numbers associated with the internal nodes at the first four levels of the tree refer to the numbering in Figure 4.4 (Section 4.5.6).

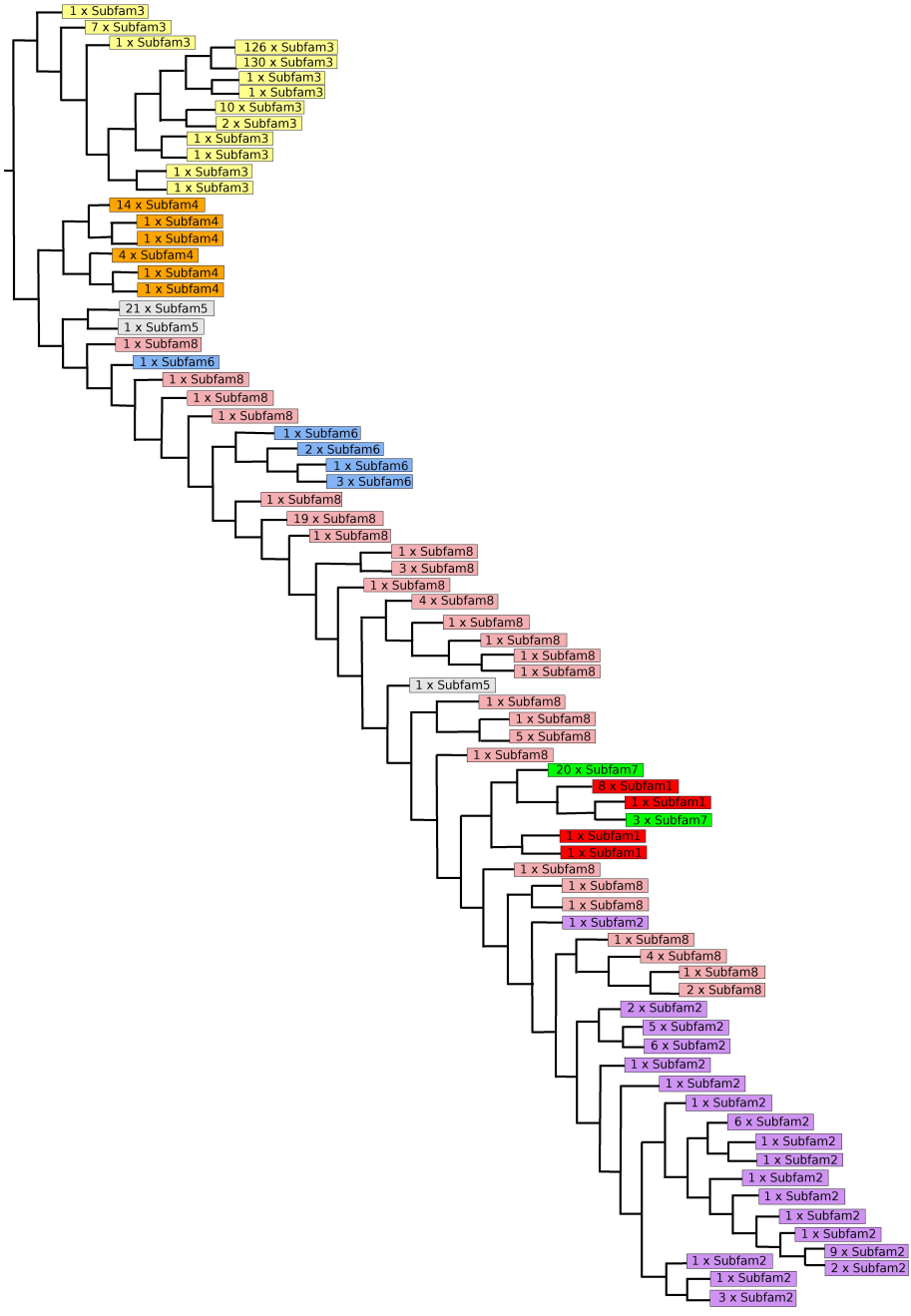


Figure B.5: SCI-PHY tree for the EXPERT dataset Enolase.

Appendix C

List of datasets used in Chapter 6

In this appendix, we provide the list of the UCI datasets [7] that were used to develop/fine-tune and evaluate our method for estimating prediction certainty in decision trees (Clus-TPCE), which presented in Chapter 6. For each dataset, we report the total number of instances, the attribute types, the number of descriptive attributes, and the number of classes. Table C.1 lists the datasets used for developing/fine-tuning Clus-TPCE. Tables C.2 and C.3 list the datasets used in the evaluation we reported in Section 6.4. In the tables, N and C stand for numeric and categorical, respectively.[1]

Table C.1: Datasets used for developing/fine-tuning Clus-TPCE.

Dataset	Nb instances	Attribute Types	Nb descriptive attributes	Nb classes
Annealing	898	N, C	38	5
Iris	150	N	4	3
Congressional Voting Records	435	C	16	2
Connectionist Bench (Sonar, Mines vs. Rocks)	208	N	60	2
Statlog (Vehicle Silhouettes)	846	N	18	4
Zoo	101	N, C	17	7

[1]For the dataset Japanese Vowels (Table C.2), we used its version (kdd_JapaneseVowels) available in the repository `http://repository.seasr.org/Datasets/UCI/arff/` (last visit: 24 June 2013), since it was already converted to the format (arff) used by our method. This version has, however, 3 more attributes than the original version, and a larger number of instances. For the other datasets, we considered their original versions.

Table C.2: Datasets used for the evaluation presented in Section 6.4 (Chapter 6). List continues in Table C.3.

Dataset	Nb instances	Attribute Types	Nb descriptive attributes	Nb classes
Abalone	4177	N, C	8	28
Adult	48842	N, C	14	2
Arrhythmia	452	N, C	279	13
Balance Scale	625	N	4	3
Balloons	20	C	4	2
Breast Cancer	286	C	9	2
Breast Cancer Wisconsin (Original)	699	N	9	2
Car Evaluation	1728	C	6	4
Chess (King-Rook vs. King-Pawn)	3196	C	36	2
Connectionist Bench (Vowel Recognition - Deterding Data)	990	N	10	11
Contraceptive Method Choice	1473	N, C	9	3
Credit Approval	690	N, C	15	2
Dermatology	366	N, C	34	6
Diabetes	768	N	8	2
Ecoli	336	N	7	8
Glass Identification	214	N	9	6
Haberman's Survival	306	N,C	3	2
Hayes-Roth	160	N	4	3
Hepatitis	155	N, C	19	2
Hill-Valley (without noise)	1212	N	100	2
Ionosphere	351	N	34	2
Japanese Vowels	9970	N	14	9
Labor Relations	57	N, C	16	2
Letter Recognition	20000	N	16	26
Lung Cancer	32	C	56	3
Lymphography	148	N, C	18	4
Molecular Biology (Promoter Gene Sequences)	106	C	57	2
Molecular Biology (Splice-junction Gene Sequences)	3190	C	60	3
Mushroom	8124	C	22	2
Nursery	12960	C	8	5
Optical Recognition of Handwritten Digits	5620	N	64	10
Page Blocks Classification	5473	N	10	5
Pen-Based Recognition of Handwritten Digits	10992	N	16	10
Pittsburgh Bridges	105	N, C	11	6
Post-Operative Patient	90	C	8	3
PrimaryTumor	339	C	17	21

Table C.3: Datasets used for the evaluation presented in Section 6.4 (Chapter 6) - Part II.

Dataset	Nb instances	Attribute Types	Nb descriptive attributes	Nb classes
Solar Flare (flare.data2)	1066	C	10	8
Spambase	4601	N	57	2
SPECTF Heart	267	N	44	2
Statlog (German Credit Data)	1000	N,C	20	2
Statlog (Heart)	270	N	13	2
Statlog (Image Segmentation)	2310	N	19	7
Statlog (Landsat Satellite)	6435	N	36	6
Teaching Assistant Evaluation	151	N, C	5	3
Thyroid Disease (version allbp)	3772	N, C	29	3
Tic-Tac-Toe Endgame	958	C	9	2
Wine	178	N	13	3
Yeast	1484	N	8	10

Twelve of the datasets listed in Tables C.2 and C.3 are separated in training and test set. Tables C.4 lists these datasets along with their total number of instances, number of training instances, and number of test instances. For the remaining datasets, we performed the experiments with leave-one-out validation.

Table C.4: Datasets with separated test set.

Dataset	Total nb instances	Nb training instances	Nb test instances
Adult	48842	32561	16281
Connectionist Bench (Vowel Recognition - Deterding Data)	990	528	462
Hayes-Roth	160	132	28
Hill-Valley (without noise)	1212	606	606
Japanese Vowels	9970	4274	5696
Letter Recognition	20000	16000	4000
Optical Recognition of Handwritten Digits	5620	3823	1797
Pen-Based Recognition of Handwritten Digits	10992	7494	3498
SPECTF Heart	267	80	187
Statlog (Landsat Satellite)	6435	4435	2000
Statlog (Image Segmentation)	2310	210	2100
Thyroid Disease (version allbp)	3772	2800	972

Bibliography

[1] Agrawal, R., Imielinski, T., and Swami, A. N. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington D.C., USA, May 26-28* (1993), ACM Press, pp. 207–216.

[2] Agrawal, R., Srikant, R., et al. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB), Santiago de Chile, Chile, September 12-15* (1994), vol. 1215, Morgan Kaufmann, pp. 487–499.

[3] Albayrak, A., Otu, H., and Sezerman, U. Clustering of protein families into functional subtypes using relative complexity measure with reduced amino acid alphabets. *BMC Bioinformatics 11* (2010), 428.

[4] Allmer, J., Markert, C., Stauber, E., and Hippler, M. A new approach that allows identification of intron-split peptides from mass spectrometric data in genomic databases. *FEBS Letters 562*, 1 (2004), 202–206.

[5] Altelaar, A. M., Navarro, D., Boekhorst, J., van Breukelen, B., Snel, B., Mohammed, S., and Heck, A. J. Database independent proteomics analysis of the ostrich and human proteome. *Proceedings of the National Academy of Sciences 109*, 2 (2012), 407–412.

[6] Arslan, A. N., and Bizargity, P. Phylogeny by top down clustering using a given multiple alignment. In *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering (IEEE BIBE), Boston, MA, USA, October 14-17* (2007), IEEE, pp. 809–814.

[7] Bache, K., and Lichman, M. UCI machine learning repository [http://archive.ics.uci.edu/ml], 2013. University of California, Irvine, School of Information and Computer Sciences.

[8] Bairoch, A., and Boeckmann, B. The SWISS-PROT protein sequence data bank. *Nucleic Acids Research 19*, Suppl (1991), 2247–2249.

[9] Benson, D. A., Boguski, M. S., Lipman, D. J., and Ostell, J. Genbank. *Nucleic Acids Research 25*, 1 (1997), 1–6.

[10] Berger, J. *Statistical Decision Theory and Bayesian Analysis*. Springer, 1985.

[11] Bickel, P., Kechris, K., Spector, P., Wedemayer, G., and Glazer, A. Finding important sites in protein sequences. *Proceedings of the National Academy of Sciences 99*, 23 (2002), 14764–14771.

[12] Biemann, K. Appendix 5. Nomenclature for peptide fragment ions (positive ions). *Methods in Enzymology 193* (1990), 886–887.

[13] Blockeel, H. *Machine Learning and Inductive Inference*. ACCO, 2010.

[14] Blockeel, H., De Raedt, L., and Ramon, J. Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning (ICML), Madison, Wisconsin, USA, July 24-27* (1998), Morgan Kaufmann, pp. 55–63.

[15] Boonen, K., Landuyt, B., Baggerman, G., Husson, S. J., Huybrechts, J., and Schoofs, L. Peptidomics: The integrated approach of MS, hyphenated techniques and bioinformatics for neuropeptide analysis. *Journal of Separation Science 31*, 3 (2008), 427–445.

[16] Bray, T., Chan, P., Bougouffa, S., Greaves, R., Doig, A., and Warwicker, J. SitesIdentify: A protein functional site prediction tool. *BMC Bioinformatics 10* (2009), 379.

[17] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[18] Brier, G. W. Verification of forecasts expressed in terms of probability. *Monthly Weather Review 78*, 1 (1950), 1–3.

[19] Brown, D., Krishnamurthy, N., Dale, J., Christopher, W., and Sjölander, K. Subfamily HMMs in functional genomics. In *Pacific Symposium on Biocomputing (PSB), Hawaii Island, USA, January 4-8* (2005), vol. 10, Citeseer, pp. 322–333.

[20] Brown, D., Krishnamurthy, N., and Sjölander, K. Automated protein subfamily identification and classification. *PLOS Computational Biology 3*, 8 (2007), e160.

[21] Cavalli-Sforza, L. L., and Edwards, A. W. Phylogenetic analysis. models and estimation procedures. *American Journal of Human Genetics 19* (1967), 233–257.

[22] Cestnik, B. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI), Stockholm, Sweden, August 6-10* (1990), Pitman, pp. 147–149.

[23] Chen, F., Mackey, A. J., Vermunt, J. K., and Roos, D. S. Assessing performance of orthology detection strategies applied to eukaryotic genomes. *PloS One 2*, 4 (2007), e383.

[24] Cheng, G., Qian, B., Samudrala, R., and Baker, D. Improvement in protein functional site prediction by distinguishing structural and functional constraints on protein family evolution using computational design. *Nucleic Acids Research 33*, 18 (2005), 5861–5867.

[25] Choudhary, J., Blackstock, W., Creasy, D., and Cottrell, J. Interrogating the human genome using uninterpreted mass spectrometry data. *Proteomics 1*, 5 (2001), 651–667.

[26] Cottrell, J., and London, U. Probability-based protein identification by searching sequence databases using mass spectrometry data. *Electrophoresis 20*, 18 (1999), 3551–3567.

[27] Craig, R., and Beavis, R. Tandem: Matching proteins with tandem mass spectra. *Bioinformatics 20*, 9 (2004), 1466–1467.

[28] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[29] Dayhoff, M., and Schwartz, R. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, M. O. Dayhoff, Ed. National Biomedical Research Foundation, 1978.

[30] Declarative Languages and Artificial Intelligence group (KU Leuven) and Department of Knowledge Technologies (Jožef Stefan Institute). Clus: A predictive clustering system. Available from `https://dtai.cs.kuleuven.be/clus/`.

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

[32] Eddy, S. Profile hidden Markov models. *Bioinformatics 14*, 9 (1998), 755–763.

[33] Eisen, J. Phylogenomics: Improving functional predictions for uncharacterized genes by evolutionary analysis. *Genome Research 8*, 3 (1998), 163–167.

[34] Eng, J., McCormack, A., and Yates, J. An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. *Journal of the American Society for Mass Spectrometry 5*, 11 (1994), 976–989.

[35] Estabrook, G. F., McMorris, F., and Meacham, C. A. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Systematic Biology 34*, 2 (1985), 193–200.

[36] Fälth, M., Sköld, K., Svensson, M., Nilsson, A., Fenyö, D., and Andren, P. Neuropeptidomics strategies for specific and sensitive identification of endogenous peptides. *Molecular & Cellular Proteomics 6*, 7 (2007), 1188–1197.

[37] Felsenstein, J. Phylip (phylogeny inference package) version 3.68. *Distributed by the author. Department of Genetics, University of Washington, Seattle (WA)* (2008).

[38] Fenyö, D., and Beavis, R. A method for assessing the statistical significance of mass spectrometry-based protein identifications using general scoring schemes. *Analytical Chemistry 75*, 4 (2003), 768–774.

[39] Fermin, D., Allen, B., Blackwell, T., Menon, R., Adamski, M., Xu, Y., Ulintz, P., Omenn, G., et al. Novel gene and gene model detection using a whole genome open reading frame analysis in proteomics. *Genome Biology 7*, 4 (2006), R35.

[40] Ferri, C., Flach, P., and Hernández-Orallo, J. Improving the AUC of probabilistic estimation trees. In *Proceedings of the 14th European Conference on Machine Learning (ECML), Cavtat, Croatia, September 22-26*, vol. 2837 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 121–132.

[41] Fierens, D., Ramon, J., Blockeel, H., and Bruynooghe, M. A comparison of pruning criteria for probability trees. *Machine Learning 78*, 1 (2010), 251–285.

[42] Fitch, W. Toward defining the course of evolution: Minimum change of specified tree topology. *Systematic Zoology 20* (1971), 406–416.

[43] Flach, P., and Matsubara, E. T. On classification, ranking, and probability estimation. In *Probabilistic, Logical and Relational Learning -*

A Further Synthesis (2008), no. 07161 in Dagstuhl Seminar Proceedings, Internationales Begegnungs-und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[44] Frank, A. A ranking-based scoring function for peptide-spectrum matches. *Journal of Proteome Research 8*, 5 (2009), 2241–2252.

[45] Frank, A., and Pevzner, P. PepNovo: *De novo* peptide sequencing via probabilistic network modeling. *Analytical Chemistry 77*, 4 (2005), 964–973.

[46] Frank, A., Tanner, S., Bafna, V., and Pevzner, P. Peptide sequence tags for fast database search in mass-spectrometry. *Journal of Proteome Research 4*, 4 (2005), 1287–1295.

[47] Fricker, L. Neuropeptide-processing enzymes: Applications for drug discovery. *The AAPS Journal 7*, 2 (2005), 449–455.

[48] Gascuel, O. BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution 14*, 7 (1997), 685–695.

[49] Gascuel, O., et al. A note on Sattath and Tversky's, Saitou and Nei's, and Studier and Keppler's algorithms for inferring phylogenies from evolutionary distances. *Molecular Biology and Evolution 11*, 6 (1994), 961–963.

[50] Geer, L., Markey, S., Kowalak, J., Wagner, L., Xu, M., Maynard, D., Yang, X., Shi, W., and Bryant, S. Open mass spectrometry search algorithm. *Journal of Proteome Research 3*, 5 (2004), 958–964.

[51] Gelman, J., Wardman, J., Bhat, V., Gozzo, F., and Fricker, L. Quantitative peptidomics to measure neuropeptide levels in animal models relevant to psychiatric disorders. *Methods in Molecular Biology 829* (2012), 487–503.

[52] Gilks, W., Audit, B., de Angelis, D., Tsoka, S., and Ouzounis, C. Percolation of annotation errors through hierarchically structured protein sequence databases. *Mathematical Biosciences 193*, 2 (2005), 223–234.

[53] Gluck, M., and Corter, J. E. Information, uncertainty and the utility of categories. In *Proceedings of the 7th Annual Conference on Cognitive Science Society (CogSci), Irvine, CA, USA, August 15-17* (1985), pp. 283–287.

[54] Gross, J. *Mass Spectrometry: A Textbook*. Berlin: Springer, 2011.

[55] Gupta, N., Benhamida, J., Bhargava, V., Goodman, D., Kain, E., Kerman, I., Nguyen, N., Ollikainen, N., Rodriguez, J., Wang, J., et al. Comparative proteogenomics: Combining mass spectrometry and comparative genomics to analyze multiple genomes. *Genome Research 18*, 7 (2008), 1133–1142.

[56] Gupta, N., Tanner, S., Jaitly, N., Adkins, J., Lipton, M., Edwards, R., Romine, M., Osterman, A., Bafna, V., Smith, R., et al. Whole proteome analysis of post-translational modifications: Applications of mass-spectrometry for proteogenomic annotation. *Genome Research 17*, 9 (2007), 1362–1377.

[57] Hagen, J. B. The origins of bioinformatics. *Nature Reviews - Genetics 1* (2000), 231–236.

[58] Hall, B. G. Simulating DNA coding sequence evolution with evolveagene 3. *Molecular Biology and Evolution 25*, 4 (2008), 688–695.

[59] Hogeweg, P. The roots of bioinformatics in theoretical biology. *PLOS Computational Biology 7*, 3 (2011), e1002021.

[60] Hook, V., Funkelstein, L., Lu, D., Bark, S., Wegrzyn, J., and Hwang, S. Proteases for processing proneuropeptides into peptide neurotransmitters and hormones. *Annual Review of Pharmacology and Toxicology 48* (2008), 393.

[61] Horn, F., Vriend, G., and Cohen, F. Collecting and harvesting biological data: The GPCRDB and NucleaRDB information systems. *Nucleic Acids Research 29*, 1 (2001), 346–349.

[62] Huang, J., and Ling, C. X. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering 17*, 3 (2005), 299–310.

[63] Hüllermeier, E., and Vanderlooy, S. Why fuzzy decision trees are good rankers. *IEEE Transactions on Fuzzy Systems 17*, 6 (2009), 1233–1244.

[64] Husson, S., Clynen, E., Boonen, K., Janssen, T., Lindemans, M., Baggerman, G., and Schoofs, L. Approaches to identify endogenous peptides in the soil nematode caenorhabditis elegans. *Methods in Molecular Biology 615* (2010), 29–47.

[65] Jaffe, J., Berg, H., and Church, G. Proteogenomic mapping as a complementary method to perform genome annotation. *Proteomics 4*, 1 (2004), 59–77.

[66] Jain, A. K., and Dubes, R. C. *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

[67] Jeong, K., Kim, S., Bandeira, N., and Pevzner, P. Gapped spectral dictionaries and their applications for database searches of tandem mass spectra. In *Proceedings of the 14th Annual International Conference on Research in Computational Molecular Biology (RECOMB), Lisbon, Portugal, August 12-15* (2010), Springer, pp. 208–232.

[68] Jiménez, C. R., Huang, L., Qiu, Y., and Burlingame, A. L. Searching sequence databases over the internet: Protein identification using MS-tag. *Current Protocols in Protein Science* (2001), 16.6.1–16.6.7.

[69] Jones, D. T., Taylor, W. R., and Thornton, J. M. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences 8* (1992), 275–282.

[70] Jukes, T. H., and Cantor, C. R. Evolution of protein molecules. In *Mammalian Protein Metabolism*, H. Munro, Ed. Academic Press, 1969, pp. 21–132.

[71] Kalume, D., Peri, S., Reddy, R., Zhong, J., Okulate, M., Kumar, N., and Pandey, A. Genome annotation of anopheles gambiae using mass spectrometry-derived data. *BMC Genomics 6*, 1 (2005), 128.

[72] Keim, D. A., Bak, P., Bertini, E., Oelke, D., Spretke, D., and Ziegler, H. Advanced visual analytics interfaces. In *Proceedings of the International Conference on Advanced Visual Interfaces (AVI), Rome, Italy, May 25-29* (2010), ACM, pp. 3–10.

[73] Kim, S., Gupta, N., Bandeira, N., and Pevzner, P. Spectral dictionaries integrating *de novo* peptide sequencing with database search of tandem mass spectra. *Molecular & Cellular Proteomics 8*, 1 (2009), 53–69.

[74] Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43*, 1 (1982), 59–69.

[75] Kukar, M., and Kononenko, I. Reliable classifications with machine learning. In *Proceedings of the 13th European Conference on Machine Learning (ECML), Helsinki, Finland, August 19-23* (2002), Springer, pp. 1–8.

[76] Kullback, S. *Information Theory and Statistics*. Dover Publications, 1997.

[77] Kuster, B., Mortensen, P., J.S., A., and Mann, M. Mass spectrometry allows direct identification of proteins in large genomes. *Proteomics 1*, 5 (2001), 641–650.

[78] Lazareva-Ulitsky, B., Diemer, K., and Thomas, P. On the quality of tree-based protein classification. *Bioinformatics 21*, 9 (2005), 1876–1890.

[79] Lee, D., Rentzsch, R., and Orengo, C. Gemma: Functional subfamily classification within superfamilies of predicted protein structural domains. *Nucleic Acids Research 38*, 3 (2010), 720–737.

[80] Lerat, E. Identifying repeats and transposable elements in sequenced genomes: How to find your way through the dense forest of programs. *Heredity 104*, 6 (2009), 520–533.

[81] Li, M., and Vitanyi, P. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, 1997.

[82] Liang, H., and Yan, Y. Improve decision trees for probability-based ranking by lazy learners. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Washington D.C., USA, November 13-15* (2006), IEEE Computer Society, pp. 427–435.

[83] Ling, C. X., and Yan, R. J. Decision tree with better ranking. In *Proceedings of the 20th International Conference on Machine Learning (ICML), Washington D.C., USA, August 21-24* (2003), Morgan Kaufmann, pp. 480–487.

[84] Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137.

[85] Ma, B., Zhang, K., Hendrie, C., Liang, C., Li, M., Doherty-Kirby, A., and Lajoie, G. Peaks: Powerful software for peptide *de novo* sequencing by tandem mass spectrometry. *Rapid Communications in Mass Spectrometry 17*, 20 (2003), 2337–2342.

[86] Mailund, T., and Pedersen, C. N. Qdist-quartet distance between evolutionary trees. *Bioinformatics 20*, 10 (2004), 1636–1637.

[87] Mann, M., Jensen, O., et al. Proteomic analysis of post-translational modifications. *Nature Biotechnology 21*, 3 (2003), 255–261.

[88] Margineantu, D. D., and Dietterich, T. G. Improved class probability estimates from decision tree models. In *Nonlinear Estimation and Classification; Lecture Notes in Statistics*, D. D. Denison, M. H.

Hansen, C. C. Holmes, B. Mallick, and B. Yu, Eds., vol. 171. Springer-Verlag, 2001, pp. 169–184.

[89] McCallum, A., Rosenfeld, R., Mitchell, T., Ng, A. Y., et al. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML), Madison, Wisconsin, USA, July 24-27* (1998), Morgan Kaufmann, pp. 359–367.

[90] Menschaert, G., Van Criekinge, W., Notelaers, T., Koch, A., Crappé, J., Gevaert, K., and Van Damme, P. Deep proteome coverage based on ribosome profiling aids MS-based protein and peptide discovery and provides evidence of alternative translation products and near-cognate translation initiation events. *Molecular & Cellular Proteomics* (2013).

[91] Menschaert, G., Vandekerckhove, T., Baggerman, G., Landuyt, B., Sweedler, J., Schoofs, L., Luyten, W., and Van Criekinge, W. A hybrid, *de novo* based, genome-wide database search approach applied to the sea urchin neuropeptidome. *Journal of Proteome Research 9*, 2 (2010), 990–996.

[92] Menschaert, G., Vandekerckhove, T., Baggerman, G., Schoofs, L., Luyten, W., and Criekinge, W. Peptidomics coming of age: A review of contributions from a bioinformatics angle. *Journal of Proteome Research 9*, 5 (2010), 2051–2061.

[93] Menschaert, G., Vandekerckhove, T., Landuyt, B., Hayakawa, E., Schoofs, L., Luyten, W., and Van Criekinge, W. Spectral clustering in peptidomics studies helps to unravel modification profile of biologically active peptides and enhances peptide identification rate. *Proteomics 9*, 18 (2009), 4381–4388.

[94] Mitchell, T. M. *Machine Learning*. Burr Ridge, IL: McGraw Hill, 1997.

[95] Mount, D. W. *Bioinformatics: Sequence and Genome Analysis*. CSHL Press, 2004.

[96] Neupert, S., Johard, H., Nässel, D., and Predel, R. Correction to single cell peptidomics of drosophila melanogaster neurons identified by gal4-driven fluorescence. *Analytical Chemistry 84*, 11 (2012), 5164–5164.

[97] Nielsen, C. B., Jackman, S. D., Birol, I., and Jones, S. J. Abyss-explorer: Visualizing genome sequence assemblies. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 881–888.

[98] Nilsson, A., Stroth, N., Zhang, X., Qi, H., Fälth, M., Sköld, K., Hoyer, D., Andrén, P., and Svenningsson, P. Neuropeptidomics of mouse hypothalamus after imipramine treatment reveal somatostatin as a potential mediator of antidepressant effects. *Neuropharmacology 62*, 1 (2011), 347–357.

[99] Oshiro, G., Wodicka, L., Washburn, M., Yates, J., Lockhart, D., and Winzeler, E. Parallel identification of new genes in saccharomyces cerevisiae. *Genome Research 12*, 8 (2002), 1210–1220.

[100] Pop, M., and Salzberg, S. L. Bioinformatics challenges of new sequencing technology. *Trends in Genetics 24*, 3 (2008), 142–149.

[101] Provost, F., and Domingos, P. Tree induction for probability-based ranking. *Machine Learning 52*, 3 (2003), 199–215.

[102] Quinlan, J. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[103] Quinlan, J. R. Induction of decision trees. *Machine learning 1*, 1 (1986), 81–106.

[104] Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*, 336 (1971), 846–850.

[105] Reeves, G. A., Talavera, D., and Thornton, J. M. Genome and proteome annotation: Organization, interpretation and integration. *Journal of the Royal Society Interface 6*, 31 (2009), 129–147.

[106] Remm, M., Storm, C. E., Sonnhammer, E. L., et al. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *Journal of Molecular Biology 314*, 5 (2001), 1041–1052.

[107] Rennert, J., Coffman, J. A., Mushegian, A. R., and Robertson, A. J. The evolution of Runx genes I. A comparative study of sequences from phylogenetically diverse model organisms. *BMC Evolutionary Biology 3* (2003), 4.

[108] Robinson, D., and Foulds, L. R. Comparison of phylogenetic trees. *Mathematical Biosciences 53*, 1 (1981), 131–147.

[109] Roepstorff, P., and Fohlman, J. Proposal for a common nomenclature for sequence ions in mass spectra of peptides. *Biomedical Mass Spectrometry 11*, 11 (1984), 601.

[110] Romanova, E., Lee, J., Kelleher, N., Sweedler, J., and Gulley, J. Comparative peptidomics analysis of neural adaptations in rats repeatedly exposed to amphetamine. *Journal of Neurochemistry* (2012).

[111] Saitou, N., and Nei, M. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution 4*, 4 (1987), 406–425.

[112] Salemi, M., and Vandamme, A. *The phylogenetic Handbook: A Practical Approach to DNA and Protein Phylogeny.* Cambridge Univ Press, 2003.

[113] Savidor, A., Donahoo, R., Hurtado-Gonzales, O., VerBerkmoes, N., Shah, M., Lamour, K., and McDonald, W. Expressed peptide tags: An additional layer of data for genome annotation. *Journal of Proteome Research 5*, 11 (2006), 3048–3058.

[114] Shevchenko, A., Sunyaev, S., Loboda, A., Shevchenko, A., Bork, P., Ens, W., and Standing, K. G. Charting the proteomes of organisms with unsequenced genomes by maldi-quadrupole time-of-flight mass spectrometry and blast homology searching. *Analytical Chemistry 73*, 9 (2001), 1917–1926.

[115] Sjölander, K. Bayesian evolutionary tree estimation. In *Proceedings of the 11th International Conference on Mathematical and Computer Modelling and Scientific Computing (ICMCM & SC) - Computational Biology Section, Washington D.C., USA, March 31 - April 3* (1997).

[116] Sjölander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I., and Haussler, D. Dirichlet mixtures: A method for improved detection of weak but significant protein sequence homology. *Computer Applications in the Biosciences 12*, 4 (1996), 327–345.

[117] Smith, J., Northey, J., Garg, J., Pearlman, R., and Siu, K. Robust method for proteome analysis by MS/MS using an entire translated genome: Demonstration on the ciliome of *Tetrahymena thermophila*. *Journal of Proteome Research 4*, 3 (2005), 909–919.

[118] Sokal, R., and Michener, C. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin 38* (1958), 1409–1438.

[119] Sourdis, J., and Krimbas, C. Accuracy of phylogenetic trees estimated from DNA sequence data. *Molecular Biology and Evolution 4*, 2 (1987), 159–166.

[120] Steen, H., and Mann, M. The abc's (and xyz's) of peptide sequencing. *Nature Reviews Molecular Cell Biology 5*, 9 (2004), 699–711.

[121] Stein, L., et al. Genome annotation: From sequence to biology. *Nature Reviews Genetics 2*, 7 (2001), 493–503.

[122] Studier, J. A., Keppler, K. J., et al. A note on the neighbor-joining algorithm of saitou and nei. *Molecular Biology and Evolution 5*, 6 (1988), 729–731.

[123] Swofford, D. *Phylogenetic Analysis Using Parsimony*. Sinauer Associates, 1998.

[124] Tanner, S., Shen, Z., Ng, J., Florea, L., Guigó, R., Briggs, S., and Bafna, V. Improving gene annotation using peptide mass spectrometry. *Genome Research 17*, 2 (2007), 231–239.

[125] Thomas, J. J., and Cook, K. A. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press, 2005.

[126] U.S. National Library of Medicine. Genetics home reference [internet]. [updated April 22, 2013; cited April 27, 2013]; Available from `http://ghr.nlm.nih.gov/handbook/basics/dna`.

[127] Van de Peer, Y., and Salemi, M. Phylogeny inference based on distance methods. In *The Phylogenetic Handbook: A Practical Approach to DNA and Protein Phylogeny*, M. Salemi and A. Vandamme, Eds. Oxford University Press, 2003.

[128] Van Dijck, A., Hayakawa, E., Landuyt, B., Baggerman, G., Van Dam, D., Luyten, W., Schoofs, L., and De Deyn, P. Comparison of extraction methods for peptidomics analysis of mouse brain tissue. *Journal of Neuroscience Methods* (2011), 231–237.

[129] Vandamme, A. Basic concepts of molecular evolution. In *The Phylogenetic Handbook: A Practical Approach to DNA and Protein Phylogeny*, M. Salemi and A. Vandamme, Eds. Oxford University Press, 2003.

[130] Varshavsky, R., Horn, D., and Linial, M. Global considerations in hierarchical clustering reveal meaningful patterns in data. *PLoS ONE 3*, 5 (2008), e2247.

[131] Venter, J. C., Remington, K., Heidelberg, J. F., Halpern, A. L., Rusch, D., Eisen, J. A., Wu, D., Paulsen, I., Nelson, K. E., Nelson, W., et al. Environmental genome shotgun sequencing of the Sargasso Sea. *Science 304*, 5667 (2004), 66–74.

[132] Vovk, V., Gammerman, A., and Saunders, C. Machine-learning applications of algorithmic randomness. In *Proceedings of the 16th International Conference on Machine Learning (ICML), Bled, Slovenia, June 27-30* (1999), Morgan Kaufmann, pp. 444–453.

[133] Wang, B., and Zhang, H. Improving the ranking performance of decision trees. In *Proceedings of the 17th European conference on Machine Learning (ECML), Berlin, Germany, September 18-22* (2006), Springer-Verlag, pp. 461–472.

[134] Wang, R., Prince, J., and Marcotte, E. Mass spectrometry of the *M. smegmatis* proteome: Protein expression levels correlate with function, operons, and codon bias. *Genome Research 15*, 8 (2005), 1118–1126.

[135] Washburn, M., Wolters, D., Yates III, J., et al. Large-scale analysis of the yeast proteome by multidimensional protein identification technology. *Nature Biotechnology 19*, 3 (2001), 242–247.

[136] Waterman, M. S., and Smith, T. F. On the similarity of dendrograms. *Journal of Theoretical Biology 73*, 4 (1978), 789–800.

[137] Wicker, N., Perrin, G., Thierry, J., and Poch, O. Secator: A program for inferring protein subfamilies from phylogenetic trees. *Molecular Biology and Evolution 18*, 8 (2001), 1435–1441.

[138] Wicker, T., Sabot, F., Hua-Van, A., Bennetzen, J. L., Capy, P., Chalhoub, B., Flavell, A., Leroy, P., Morgante, M., Panaud, O., et al. A unified classification system for eukaryotic transposable elements. *Nature Reviews Genetics 8*, 12 (2007), 973–982.

[139] Wikimedia Commons (user: Kelvinsong). Simple illustration of upstream and downstream on a double-stranded piece of DNA [internet]. [updated December 15, 2012; cited June 27, 2013]; Available from `http://upload.wikimedia.org/wikipedia/commons/5/5d/Upstream_and_downstream.svg`.

[140] Yates III, J., Eng, J., and McCormack, A. Mining genomes: Correlating tandem mass spectra of modified and unmodified peptides to sequences in nucleotide databases. *Analytical Chemistry 67*, 18 (1995), 3202–3210.

[141] Zadrozny, B., and Elkan, C. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the 18th International Conference on Machine Learning (ICML), Williamstown, MA, USA, June 28 - July 1* (2001), Morgan Kaufmann, pp. 609–616.

[142] Zhu, X., Goldberg, A. B., Brachman, R., and Dietterich, T. *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers, 2009.

[143] Zurada, J. M. *Introduction to Artificial Neural Systems*. West St. Paul, Minn., 1992.

List of publications

Journal Articles

- E. P. Costa, C. Vens, and H. Blockeel, Top-down clustering for protein subfamily identification. Evolutionary Bioinformatics, volume 9, pages 185-202, 2013.

- E. P. Costa, G. Menschaert, W. Luyten, K. De Grave, and J. Ramon, PIUS: Peptide Identification by Unbiased Search, Bioinformatics - application notes, doi: 10.1093/bioinformatics/btt298, 2013, [two-page article, supplementary material: 23-page technical report].

Conference Papers

- C. Vens, E. P. Costa, H. Blockeel. Top-down induction of phylogenetic trees, European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO), Istanbul, Turkey, 7-9 April 2010, Lecture Notes in Computer Science, volume 6023, pages 62-73, Springer.

- C. Vens, E. P. Costa, H. Blockeel. Top-down phylogenetic tree reconstruction, Pattern Recognition in Bioinformatics (PRIB), Sheffield, United Kingdom, 7-9 September 2009, [short paper: six pages].

Posters and Presentations at Miscellaneous Events

- E. P. Costa, S. Verwer, H. Blockeel. New procedure for estimating prediction certainty in decision trees, Belgian-Dutch Conference on

Machine Learning, Ghent, 24-25 May 2012, Proceedings of the 21st Belgian-Dutch Conference on Machine Learning (BENELEARN), page 60, [one-page abstract].

- E. P. Costa, C. Vens, H. Blockeel. Protein Subfamily Identification using Clustering Trees, Belgian Dutch Conference on Machine Learning, The Hagge, 20 May 2011, Proceedings of the 20th Belgian-Dutch Conference on Machine Learning (BENELEARN), pages 105-6, [two-page abstract, poster].

- E. P. Costa, C. Vens, H. Blockeel. Identification and classification of protein subfamilies using top-down phylogenetic tree reconstruction, European Conference on Computational Biology (ECCB), Vienna, 17-19 July 2011, [half-page abstract, poster].

- E. P. Costa, C. Vens, H. Blockeel. Evaluating the use of clustering trees for protein subfamily identification, BeNeLux Bioinformatics Conference (BBC), Luxembourg, 12-13 December 2011, [one-page abstract, poster].

- E. P. Costa, C. Vens, H. Blockeel. Reconstructing phylogenetic trees from clustering trees, European Conference on Computational Biology (ECCB), Ghent, Belgium, 26-29 September 2010, [half-page abstract, poster].

- E. P. Costa, C. Vens, H. Blockeel. Top-down phylogenetic tree reconstruction: a decision tree approach, International Workshop on Machine Learning in Systems Biology (MLSB), Ljubljana, Slovenia, 5-6 September 2009, [one-page abstract, poster].

- E. P. Costa, C. Vens, H. Blockeel. Top-down phylogenetic tree reconstruction, Benelux Bioinformatics Conference, Liege, 14-15 December 2009, [one-page abstract, poster].

Submitted

- E. P. Costa, S. Verwer, and H. Blockeel, Estimating prediction certainty in decision trees. Submitted to the 12th International Symposium on Intelligent Data Analysis (IDA). Submitted in May 2013.

- T. Fannes, E. P. Costa, J. Ramon et al., Machine learning in proteomics. Submitted to the Nectar Track of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 2013 (ECMLPKDD 2013). Submitted in June 2013.

- E. P. Costa, L. Schietgat, R. Cerri, C. Vens, C. N. Fischer, C. M. A. Carareto, J. Ramon, H. Blockeel, Annotating transposable elements in the genome using relational decision tree ensembles. Submitted to the 23rd International Conference on Inductive Logic Programming (ILP). Submitted in July 2013, [short paper: six pages].

Curriculum vitae

Eduardo de Paula Costa graduated in computer science at the São Paulo State University (UNESP), Brazil, in December 2005. He then joined the master program in computer science and computational mathematics from the University of São Paulo (USP), Brazil, in February 2006. He obtained his master title in March 2008. His master thesis was entitled "Investigation of hierarchical classification techniques for bioinformatics problems".

In May 2008, he joined the Declarative Languages and Artificial Intelligence group at the KU Leuven, Belgium, as a pre-doctoral student. After being approved in the pre-doctoral program, he started his doctoral studies in January 2009. His Ph.D. research, entitled "Algorithms for analyzing biological sequences" was supervised by Hendrik Blockeel and Jan Ramon.

FACULTY OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE
DECLARATIVE LANGUAGES AND ARTIFICIAL INTELLIGENCE
Celestijnenlaan 200A
B-3001 Heverlee