Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Master of Science Applied Mathematics

Classifying Candida species using Mixed Integer Optimization based optimal classification trees

Mick van Dijk

4604040

28 January 2019

Supervisors:

Dr. ir. L.J.J. van Iersel

Prof. dr. L. Stougie

Dr. ir. S. Kelk

Abstract

Global medical use of azole antifungals and echinocandins has led to an enormous increase in resistant *Candida* species, that are most commonly associated with fungal infections. A possible mechanism causing resistance are single or simultaneous point mutations in the genes responsible for encoding antifungal target enzymes. The aim of this thesis is to apply and compare several classification algorithms, in particular decision tree algorithms, on *Candida* data sets received from the Westerdijk Fungal Biodiversity Institute. Bertsimas and Dunn recently introduced a novel formulation based on Mixed Integer Optimization to generate optimal classification trees. We have implemented this method and applied it on *C. albicans* and *C. glabrata* data sets to construct univariate and multivariate classification trees. We were able to correctly classify 68-72% of the *C. albicans* isolates and 76.5-82.5% of *C. glabrata* isolates. Moreover, by changing the objective function and adding constraints to the original MIO formulation, we constructed trees that take into consideration false negative errors, decreasing this type of error by 64-80% for *C. albicans* and 56-66% for *C. glabrata*. To deal with ambiguous nucleotides in the *C. albicans* data set we introduced a novel formulation to construct non-binary classification trees. It turned out that ternary trees are a good representation of the *C. albicans* data set, performing strong in terms of out-of-sample accuracy. Finally, we identified combinations of amino acid substitutions and nucleotide mutations possibly related to resistance in *C. albicans* and *C. glabrata*.

Acknowledgements

I would like to express my sincere appreciation to:

- Leo van Iersel, my first supervisor at the TU Delft,
- Leen Stougie, my daily supervisor at CWI,
- Teun Boekhout, from the Westerdijk Fungal Biodiversity Institute,
- Steven Kelk, from Maastricht University.

Contents

Chapter 1
Introduction

Every year approximately 1.5 million people die because of fungal infections. Mortality associated with fungal diseases is more than that of malaria and similar to tuberculosis [3]. Fungi are biologically different from viruses and bacteria, but they are as much of a threat to life and health [29]. Despite these alarming numbers, there is still little attention for curing and prevention of fungal infections, even though, according to Bongomin et al. (2017), "most deaths from fungal diseases are avoidable" [3, p. 1].

Candida is a genus responsible for a majority of fungal diseases [3]. Approximately 750.000 people suffer from invasive candidiasis, an infection induced by any type of *Candida* yeast [3]. While historically *C. albicans* is most commonly associated with candidiasis, there has been a worrying increase in the emergence of other infectious species, such as *C. glabrata*, *C. parapsilosis*, *C. tropicalis*, *C. krusei* and *C. auris*. These species are more likely to be resistant to antifungals, and therefore dangerous for outbreaks [3]. Especially resistance to fluconazole, a globally used medication for treating *Candida* infections, is threatening.

Fluconazole is a triazole derivative that acts as an antifungal by inhibiting fungal growth. It interferes with the synthesis of ergosterol, which is crucial as a component of fungal cell membranes [35]. Fluconazole disrupts the biosynthesis of ergosterol through inhibition of 14-α-sterol demethylase (CYP51A1) encoded by the ERG11 gene [35]. Primary therapeutic treatment with fluconazole has led to mechanisms of fluconazole resistance in both *C. albicans* and non-*albicans Candida* species.

Point mutations in the ERG11 gene is considered to be one of the several mechanisms of fluconazole resistance of *C. albicans* [35]. Amino acid substitutions, due to point mutations, can alter the structure of the CYP51A1 protein in such a way that binding of fluconazole to CYP51A1 becomes impossible. Studies identified different amino acid substitutions associated with fluconazole-resistant candidiasis [35].

Another class of antifungal drugs are echinocandins. Wide use of echinocandins as primary medication against candidiasis affects the susceptibility of *C. glabrata*. Combinations of point mutations in genes FKS1 and FKS2, encoding subunits of the target enzyme 1,3-β-glucan synthase, might be responsible for echinocandin resistance in *C. glabrata*.

The enormous increase in the number of resistant candidiasis cases stresses the importance of prior treatment resistance classification. However, clinically testing new isolates on resistance is time consuming and expensive. Usually, the broth microdilution method is applied to test the susceptibility of the fungal isolates to different azoles, including fluconazole. This method requires the tested fungal agent to be a pure substance, individual mistakes in the stock preparation process can easily lead to inaccurate results [17].

With the rise of bioinformatics and the emergence of machine learning techniques in biology it is

possible to do non-clinical analysis on biological data. Over the past decades there has been an enormous increase in the computational power of computers, solvers are becoming highly efficient and algorithms are being continually improved. This, together with the possibility to extract large amounts of biological data, e.g. genome data and gene expression data, creates opportunities for in silico analysis. Instead of clinically testing new species on resistance it is possible to use machine learning approaches for classification of species and identification of resistance responsible mutations.

Decision trees, also called classification trees, are widely used machine learning techniques for classification problems [2], i.e. the problem of identifying to which group or class a new observation belongs. Given a set of labeled data points, the objective is to partition the data space recursively by selecting a feature or multiple features to split on at each step. A label will be assigned to each of the obtained partitions according to a majority principle. Finally, the constructed classification tree can be used to classify future data. Besides decision trees there is a wide range of methods for solving classification problems. One major advantage of decisions trees over the other methods is their interpretability [2]. Moreover, by analyzing decision paths in the tree, they allow the user to identify the exact location of single or combinations of mutations possibly associated with resistance.

There are several decision tree algorithms available, including C45, CART, SPRINT and SLIQ [19]. These methods share a top-down approach that is greedy in nature, at each step a local optimization problem is solved in isolation [2]. Bertsimas and Dunn (2017) formulate the problem of creating an optimal classification tree as a Mixed Integer Optimization (MIO) problem, which can efficiently be solved by state of the art MIO solvers.

One key advantage of using MIO is that this approach enables global minimization of the misclassification error, instead of solving multiple local optimization problems as part of a top-down induction process. This yields optimal classification trees, while other traditional algorithms solve the problem by a heuristic. Moreover, as we will show in this thesis, altering the original MIO formulation for constructing optimal classification trees as presented by Bertsimas and Dunn (2017) gives the opportunity to model false positive and false negative errors and non-binary classification trees.

The aim of the present study is to apply different tree constructing algorithms to classify *C. albicans* and *C. glabrata* isolates. The constructed trees are compared in terms of accuracy after which the best models are selected. A second objective is to identify resistance-related mutations in ERG11, FKS1 and FKS2. We will examine if there is an interdependent mechanism between mutations in FKS1 and FKS2 that can be related to resistance in *C. glabrata*.

For comprehensibility purposes this thesis is divided in two main parts, the first part aims to capture the biological importance of this research, explains the data and describes decision trees as a method for classification. The second part, which requires a stronger mathematical background, contains a global description of the MIO formulation for creating optimal classification trees and several modifications of this formulation to obtain: non-binary trees, multivariate trees and model false positive and false negative. Multivariate trees have been described by Bertsimas and Dunn in [2], the other two extensions are novel.

Finally, Chapter 6 reports the experimental accuracy results obtained by applying different classifi-

cation methods on *C. albicans* and *C. glabrata* data sets. In this chapter we also investigate which mutations or combinations of mutations are possibly related to resistance and compare them to previously reported resistance-related mutations and substitutions.

1.1 Related work

State-of-the-art decision tree algorithms apply heuristics that run fast and provide reasonably good solutions [2, 12]. Greedy methods, like C4.5 [26] and CART [4], use local optimization to select the best split at internal nodes, this does not necessarily result in an optimal tree, where optimality is regarded as minimizing the misclassification error. Common measures to determine the quality of a split are Gini impurity (CART) and Information gain (C4.5). Recently, there are several methods proposed to obtain optimal classification trees by using integer optimizaton techniques. In the paper *Optimal classification trees* (2017), Bertsimas and Dunn formulate the problem of finding optimal classification trees as a mixed integer optimization problem [2]. On average, the out-of-sample accuracy improved 1-2% for univariate trees and 3-5% for multivariate trees compared to CART [2].

Since the appearance of the paper written by Bertsimas and Dunn, a lot of research has been carried out related to the topic of using mixed integer optimization for constructing classification trees. Solving the problem of constructing classification trees by using MIO turns out to be time consuming for large or high dimensional data sets. Data sets with thousands of points can only be handled for tree depths up to four, the time required to generate solutions for data sets exceeding this number of points or deeper trees is prohibitive [2].

It was shown in [2], that injecting a warm start improves the running time significantly. In [11], W. Feijen presents several other methods to decrease the solving time for the Iris data set including: reduction of the big M-variable, adding valid constraints, discretization of the solution space and adjustments in the branch-and-bound method. Running times were decreased up to 95% for the Iris data set from [10] after applying a single modification or combination of these modifications. Adding smart branching strategies was the most effective method to decrease running times for both univariate and multivariate tree constructing MIO methods. For univariate trees, running times were decreased to 8-25% of default and for multivariate trees to 1-4% of default [11].

In [12], the authors explore the use of Column Generation (CG) to construct univariate classification trees where the formulation is based on paths in decision trees. Numerical experiments show that this method outperforms CART in terms of in-sample and out-of-sample accuracy. The CG approach also performs better than other state-of-the-art MIO based methods suggested in recent years [12].

Chen et al. report on the application of classification trees to a variety of bioinformatic problems [6]. Genome based resistance classification by using decision trees has, for example, been done by Beerenwinkel et al. in [1] and Murray et a. in [22]. In [1] they use the C4.5 algorithm to construct classification trees to predict phenotypic resistance from genotypes. Instead of C4.5, we apply the MIO formulation to predict resistance and identify resistance related mutations.

1.2 Contribution

Our contribution is related to the implementation of the MIO formulation from Bertsimas and Dunn [2] on genome data sets. We use this formulation to classify *C. albicans* and *C. glabrata* yeast isolates, identify mutations and model false positive en false negative. In [23], the authors propose a method that also takes into consideration false positive and false negative errors. They define a new function, instead of Information gain, and implement it in the C4.5 algorithm. We, however, model false positive and false negative by altering the objective function in the MIO formulation resulting in a novel formulation.

In previous work, multiple resistance related mutations and amino acid substitutions were identified for *C. albicans* and *C. glabrata*. This work examines the use of classification trees to identify mutations and amino acid substitutions associated with resistance. Classification trees are easy interpretable and they allow to analyse the effect of simultaneous mutations and substitutions on resistance. We discovered a combination of three simultaneous mutations on ERG11 possibly related to resistance. For *C. glabrata* we found a combination of simultaneous mutations on FKS1 and FKS2 present in multiple resistant sequences, which could indicate interdependency between mutations on FKS1 and FKS2 causing resistance.

Finally, this thesis examines the application of MIO based non-binary classification trees. Ternary classification trees are used to classify *C. albicans* isolates and quaternary trees to identify combinations of resistance related amino acid substitutions. Both extensions of the original MIO formulation for non-binary classification trees are novel and effective for analyzing the *C. albicans* data set.

Chapter 2
Candida albicans and *Candida glabrata*

Normally, *C. albicans* is a harmless commensal organism residing in the gut, genito-urinary tract and skin [3]. However, it can become an opportunistic pathogen for immunocompromised people. Major drivers for invasive candidiasis are: HIV, COPD, asthma and tuberculosis [3]. Historically, between 70-80% of the cases of invasive candidiasis were caused by *C. albicans*. However, in the last couple of decades there has been a shift towards non-*albicans* species causing fungal infections, especially in South-Asian countries where in 70-90% of the cases non-*albicans* species were reported. In this research we will focus on *C. albicans* and *C. glabrata*, both commonly associated with infections. The classification methods applied on these data sets can be used for the classification of any kind of numerical or categorical data.

Global use of azole antifungals as treatment against infections caused by *C. albicans* and other non-*albicans* species has lead to an increase in azole resistance. Several mechanisms have been reported as potential reasons for azole resistance in *Candida* species [30, 24].

In this chapter we will discuss how the commensal fungal species *C. albicans* and *C. glabrata* can propagate as a pathogen for vulnerable groups causing infections that are referred to as candidiasis. We will look at the epidemiology of invasive candidiasis and its risk factors. Furthermore, we explain several mechanisms of drug resistance in both species focusing on genetic alteration related resistance. Finally, we describe the *C. albicans* and *C. glabrata* data sets that we received from the Westerdijk Fungal Biodiversity Institute. Both data sets will be used to construct classification models.

2.1 Invasive candidiasis

Since 1940 there has been an enormous increase in the occurrence of hematogenously disseminated candidiasis. Today, invasive candidiasis represents the third-to-fourth most frequent nosocomial infection. One of the reasons for this increase is the propensity of *Candida* species to infect medical devices. The ongoing development and placement of medical implantation devices, like indwelling catheters, artificial hips, knees and shoulders and prosthetic heart components, has lead to more *Candida* infections [34]. Besides prosthetic devices, organ or stem cell transplantation can also increase the risk of candidiasis. A second reason is the increase in the number of individuals sensitive to invasive fungal infections in recent years. Due to advances in medical treatment many diseases such as HIV/AIDS pose much less of a direct threat to humans. However, people suffering from these diseases are more sensitive to opportunistic *Candida* infections because of reduced immunity [36].

Globally, there are more than 150 different *Candida* species reported but only 15 of them are isolated as pathogenic agents from patient. In the last 20-30 years, 95% of the *Candida* infections involved *C. albicans*, *C. glabrata*, *C. parapsilosis*, *C. tropicalis*, and *C. krusei* [36].

2.2 Drug resistance mechanisms in *Candida albicans*

Most *C. albicans* infections are treated by antifungal azoles. Especially triazoles, including fluconazole, itraconazole, posaconazole and voriconazol, are commonly used as medication against *Candida* infections. Among these, fluconazole is a major option for the treatment of *C. albicans* related diseases [30].

Fluconazole acts as an antifungal by targeting the enzyme 14α-lanosterol demethylase (CYP51A1), which plays a role in the biosynthesis of ergosterol. Ergosterol is a crucial component of fungal cell membranes. Modification of ergosterol leads to loss of membrane integrity which will ultimately result in cell death [30].

There are several mechanisms of antifungal resistance of *C. albicans*. In [30], Sanglard separates these mechanisms into the following three categories: (1) decrease of effective drug concentration within the cell, (2) drug-target alterations, and (3) metabolic bypasses. In this research we are interested in alterations of the target enzyme via genetic modification as a defence mechanism against the antifungal. Even though we will not explain resistance mechanisms from the first and third category, it is important to keep in mind that resistance to azoles can have multiple origins.

The gene ERG11 encodes for the enzyme CYP51A1 in *C. albicans*. Point mutations or simultaneous mutations in ERG11 can lead to increased azole resistance. Mutations in ERG11 have different effects on each of the individual triazoles. It turns out that most known ERG11 mutations lead to a decreased affinity to fluconazole [30]. There are four mutation scenarios possible: (1) silent mutation, this type of mutation does not affect the amino acid order, therefore there is no change in the CYP51A1 protein structure and thus azoles can bind to it, (2) there is a mutation but it does not affect the folding of the protein, therefore the structure remains unchanged and azoles can bind, (3) the mutation changes the protein structure but not enough to prevent azoles from binding, (4) the mutation alters the structure sufficiently to stop azoles from binding.

2.3 *Candida albicans* data set

Point mutations in the ERG11 gene is one of the biological mechanisms of azole resistance. Analysing the ERG11 coding regions of several *C. albicans* isolates is important to identify the mutations that are possibly responsible for resistance and to classify future isolates by considering the positions of appearance of these mutations in ERG11.

For this research the Westerdijk Fungal Biodiversity Institute provided a data set containing encoded ERG11 genes derived from 94 different *C. albicans* isolates, 45 of the isolates are tested as being resistant and the other 49 are known to be susceptible. Each ERG11 gene sequence consists of 1543 nucleotides, besides the main nucleotides A, T, C and G, the data set contains ambiguity characters W, S, M, K, R and Y representing all combinations of two main nucleotides as formalized by the International Union of Pure and Applied Chemistry (IUPAC) [7, 9]. These ambiguity symbols emerge when there is uncertainty between two main nucleotides or to code heterozygote sites of the gene.

2.3.1 Ambiguous nucleotides

C. albicans is a diploid organism, which means that it has two copies of the same gene in the system. There are two possibilities for each position on the gene: (1) both genes contain the same nucleotide on the position, (2) the genes contain different nucleotides on the same position. The first situation defines homozygous sites while the second situation represents heterozygosity. The heterozygous sites are described by the symbols that are also used for uncertain sites, as described in Table 2.1.

Figure 2.1 shows the genome reading process of two human individuals, Individual A and Individual B. In diploid organisms, such as humans but also *C. albicans*, each cell contains two copies of genes. On the left side of the figure there are two strands of DNA representing parts of two copies of genes. On position three of the genome of Individual A the genes contain different symbols, C and A, this site is heterozygous and therefore indicated by using the ambiquity symbol M. Individual B, however, is homozygote on site three, both genes contain nucleotide A on this position, and therefore this site is indicated by A.

Symbol	Bases	Description
A	**A**	**Adenine**
C	**C**	**Cytosine**
T	**T**	**Thymine**
G	**G**	**Guanine**
W	**A or T**	**Weak**
S	**C or G**	**Strong**
M	**A or C**	**aMino**
K	**G or T**	**Keto**
R	**A or G**	**puRine**
Y	**C or T**	**pYirimidine**
N	**any base**	**any Nucleotide**

Table 2.1: IUPAC nucleotide system

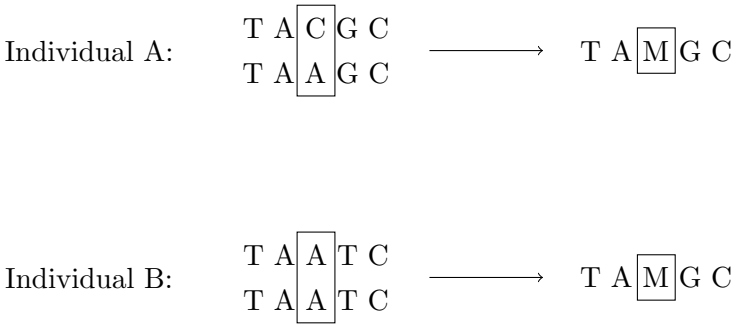


Figure 2.1: Genome sequencing of to individuals

2.4 *Candida glabrata*

"After *Candida albicans*, *Candida glabrata* is one of the most prevalent pathogenic fungi in humans" [28, p. 1]. *C. glabrata* has been considered as a relatively non-pathogenic saprophyte for many years. In recent years however, there has been an increase in the number of mucosal and systemic infections caused by *C. glabrata*. Increased use of immunosuppressive therapy together with broadspectrum antibiotic and antifungal therapies resulted in this growing number of cases of *C. glabrata* infections. Compared with all other *Candida* species, *C. glabrata* infections are the most threatening since the mortality rate associated with this species is the highest [28].

2.4.1 Drug resistance mechanisms in *Candida glabrata*

Global use of fluconazole as primary antifungal therapy against *Candida* infections has led to an increase in azole resistance. There are several mechanisms that can cause azole resistance of *C. glabrata*. The most common mechanism is induction of efflux pumps encoded by ABC-transporter genes. These

proteinaceous transporters are responsible for removing biological and chemical compounds like antibiotics out of the cell. Upregulation of these genes can result in increased resistance to azole drugs.

Due to an increase in the number of azole resistant *glabrata* isolates, echinocandins are used as a primary therapy for the treatment of invasive candidiasis. Recent studies, however, show that the wide use of echinocandins affects the susceptibility of *Candida* species, especially *C. glabrata*. Between 2001 and 2010, resistance to echinocandins increased from 4.9% to 12.3% according to a study performed from patients with *C. glabrata* bloodstream infections [28].

Echinocandins are antifungals that operate via inhibition of the enzyme 1,3-β glucan synthase, which is involved in the biosynthesis of beta-glucan, a component of fungal cell walls [31, 28]. Mutations in hot-spot regions of FKS1 and FKS2 genes might be responsible for reduced susceptibility of *C. glabrata* to echinocandins [31]. Westerdijk Fungal Biodiversity Institute has provided us with a data set containing encoded FKS1 and FKS2 genes of 83 *C. glabrata* isolates.

2.4.2 Preprocessing of the data

There are several preprocessing steps that we apply on the *C. albicans* data set. First we remove all identical copies of sequences that are present in the data. After this step, the data set contains 80 sequences, half of them is resistant and the other half susceptible. The second step is to remove redundant sites. All classification models we consider use positions on the gene to differentiate between the isolates. Positions where each isolate contains the same nucleotide are irrelevant for the constructed model and therefore can be removed. Furthermore, isolates 25 and 30 contain repetitions of 9 and 59 unknown nucleotide symbols N, respectively. As a final step in cleaning the data, we replace each symbol N in isolates 25 and 30 by the nucleotide that is present in the rest of the isolates. We believe that this nucleotide uncertainty is due to some error in the reading process which should not have an influence on the model. The final data set, after all preprocessing steps, contains 80 sequences of length 61, half labeled as being susceptible and the other half resistant. Figure 2.2 shows part of the preprocessed *C. albicans* data set. In Chapter 5 we discuss an encoding method to turn categorical data into numerical data. The *C. glabrata* data set, after applying similar preprocessing steps to the *C. albicans* data set, consists of 83 sequences. From these sequences, 28 are labeled as resistant and 55 are identified as susceptible. Each sequence contains 202 nucleotides denoted by the symbols as agreed by the IUPAC [7, 9].

Contrary to *C. albicans*, *C. glabrata* is a haploid yeast which means that each cell contains only one set of chromosomes. Since there is only one copy of each chromosome, there is no ambiguity between nucleotides, and therefore the *C. glabrata* data set contains none of the ambiguity symbols besides the nucleotides A,C,T and G. However, there are deletions present in the data set, one of the sequences has a deletion of nucleotide A, two sequences have deletions of G and several sequences have deletions of three adjacent nucleotides all at the same position. We encode these states respectively by the symbols $\mathrm{D_A}, \mathrm{D_G}$ and $\mathrm{D_3}$.

☑ 40. STA* 178 (S)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 41. STA* 183 (S)	C	G	G	T	A	C	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 42. STA* 194 (S)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 43. STA* 214 (S)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 44. Y105 (S)	C	G	G	T	A	M	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 45. Y107 (S)	C	G	G	T	A	C	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 46. 04BL1-97 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	C	A	T
☑ 47. 04BL2-162 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	C	A	T
☑ 48. 06BL1-6 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 49. 04BL1-80 (R)	C	G	G	T	A	C	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 50. 05BL1-2 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 51. 04BL2-145 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 52. 04BL2-147(R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T
☑ 53. 04BL1-61 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	C	A	T
☑ 54. 05BL2-174 (R)	C	G	G	T	A	A	A	G	G	G	G	T	T	A	T	T	T	A	T

Figure 2.2: Small part of the *C. albicans* data set in MEGA software

Chapter 3
Classification trees

Constructing classification trees is a machine learning technique that has been widely used in statistics and data mining as a model for classifying numerical and categorical data. Examples of numerical data are: melting temperatures of chemical elements, numbers of inhabitants per country and blood pressure measurements. Each data point has a numerical value that can be binary, integer or real. Categorical data is non-numerical, examples are: blood types (A, B, AB, O), board configurations at the end of a tic-tac-toe game and the political preference of dutch citizens (PvdA, Groenlinks, D66, etc.). Classification trees are, contrary to other classification methods such as neural networks, easy to interpret as they closely resemble human reasoning [19].

Also in bioinformatics and statistical genetics classification trees are popular as a tool to extract information from data. One of the reasons for this success is their capability to handle high dimensional data where there is only a limited number of samples, which is a very common situation in bioinformatics and statistical genetics [6]. Before explaining how classification trees are constructed and operate, we will first give a short introduction to machine learning. At the end of this chapter we demonstrate how to use classification trees for data analysis on genome data.

3.1 Machine learning

Machine learning can be defined as "a set of methods that automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty" [27, p. 1]. There are two broad categories in machine learning, supervised learning and unsupervised learning.

In supervised learning the decision maker starts off with a so called *training set*, each element of this set is a pair consisting of an input object and an output value. Typically the input object is a vector and the output value is a label or class [21]. Based on the training set, the algorithm "learns" a function that maps future input data to an output value. In the best case scenario, the supervised learning algorithm produces a perfect mapping that correctly labels new instances. Examples of supervised learning algorithms are: linear regression, support vector machines, logistic regression, k-nearest neighbour algorithm and decision trees [16].

Unsupervised learning deals with unlabeled data. The goal is to find interesting patterns within this data set by using algorithms. There are numerous applications of unsupervised learning algorithms including pattern recognition, web mining, social network analysis, market research, information retrieval and fraud detection [5]. What makes this type of learning challenging is the inability to evaluate the structure or pattern that is produced by the algorithm because of absence of labeled data. Examples of unsupervised learning algorithms are: neural networks, clustering algorithms and anomaly detection algorithms [16].

Machine learning techniques are very useful for analyzing the fast-growing volume of biological data. New information about underlying biological systems can be extracted from this data and turned into a model to obtain predictions for future data [20]. Figure 3.1 shows the interaction between machine learning methods and multiple topics within biology. Especially in the field of genomics various machine learning methods are being applied to obtain new knowledge. In the last decades progress in genetic technologies made it possible to sequence genomes from various organisms quickly and cost-effectively. This has led to an exponential growth in the number of available sequences. Processing these sequences to extract, for example, information about genetics, protein production and genetic pathology can be achieved by using machine learning techniques.

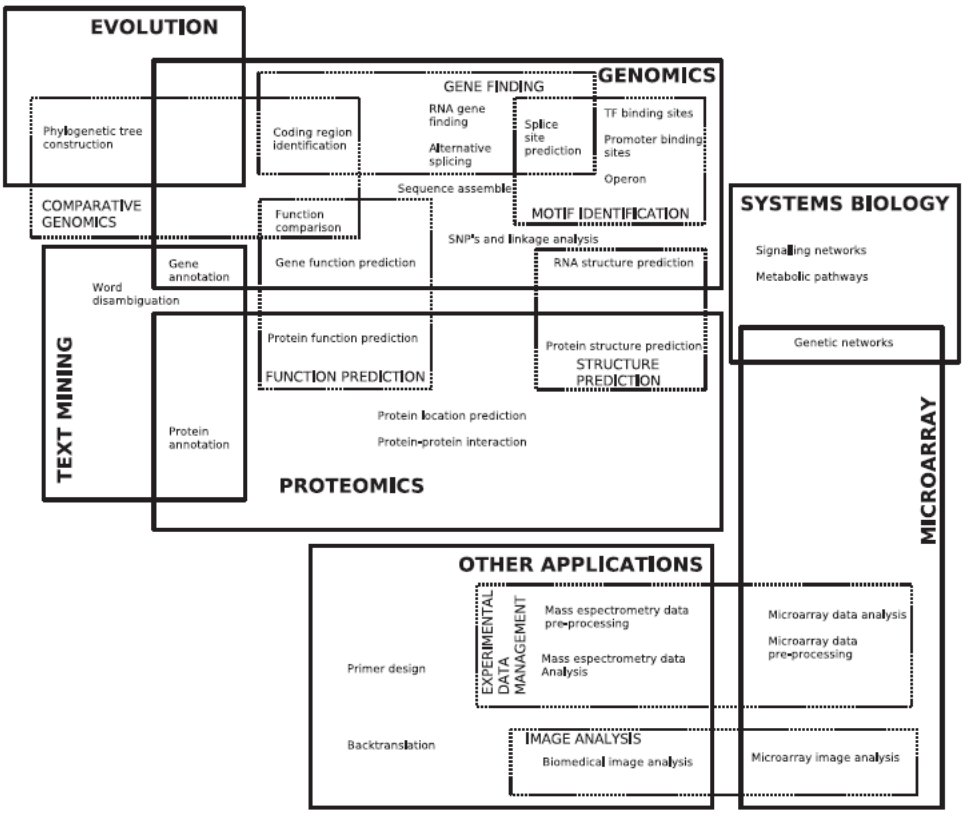


Figure 3.1: Topics in Biology where machine learning is applied. Adapted from *Machine learning in bioinformatics* by Larranaga, P. et al. 2006, Briefings in bioinformatics, 7(1), 86-112.

3.2 Classification trees

The objective of decision tree algorithms is to construct a tree model based on numerical or categorical training data that can then be used to classify new data. Decision trees are supervised machine learning methods where the training data is recursively partitioned at each step of the algorithm by choosing a feature, or a combination of multiple features, to split on. This process can be repeated until there is a perfect classification of the data or when a specific tree depth is reached. The final tree will consist of a root node and a number of branches that represent the order of splits applied for the classification.

When there is new data available the algorithm can classify this by following the correct path through the obtained classification tree.

Figure 3.2 shows a very simple classification tree of depth two where gender is classified based on the length and weight of a person. These two characteristics, height and length, are usually referred to as features. The top node is called the root node and the nodes where splits are applied are branch nodes. The labeled bottom nodes are called leaf nodes. Training data for this tree includes phenotypic information about the features weight and length together with the corresponding gender identity. The aim is to induce a classification tree from this data and use it as a model to classify future data, i.e. identify if a person is male or female based on his/her length and weight.

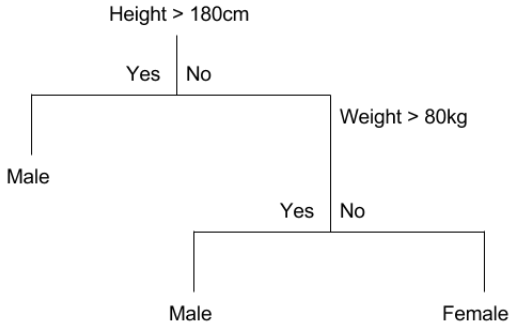


Figure 3.2: Simple classification tree of depth two. Adapted from *Classification And Regression Trees for Machine Learning* by Browniee, J. (2016).

The tree depicted in Figure 3.2 is binary, has a depth of two and classifies into two classes, which respectively means that at each split there are only two possible outcomes, yes or no; the tree terminates after applying two splits and there are two possible outcomes, male or female. Classification trees can have numerous different structures depending on the dimensionality of the data, the set of classes and the number of possible states at each split. Each of these different tree topologies will affect the performance of the model.

Before constructing a tree, the data set is split into training data and test data. Training data is used by the algorithm to construct the tree and for the evaluation of the performance labeled test data is available. Good performance of the model is generally obtained as a trade-off between the complexity of the decision tree, its ability to fit the training data and, most important, the ability to fit unseen data or test data.

The goodness of fit of the model is a measure for the discrepancy between the class determined by the algorithm and the actual label of each data point. A common used measure for this discrepancy is the misclassification error that is usually defined as the total number of data points in a leaf minus the number of points of the most common label in that leaf. The model with the least misclassification error, i.e. the model that fits the training data the best, will be called optimal.

To minimize the total misclassification error most tree constructing algorithms recursively solve local optimization problems to determine the best feature to select on at each split. This top down approach

is greedy in nature which can affect the global optimality of the solution. Widely used methods for constructing classification trees are CART, C4.5 and ID3. To explain the decision making process of these algorithms we will describe the method practised by CART.

3.3 CART

One of the most common algorithms for constructing classification trees is via classification and regression trees (CART). To measure the quality at each split CART uses the Gini impurity which provides an indication about the purity of the corresponding child nodes. A node is *pure* if all of its members belong to a single class, i.e. there is a homogeneous class distribution of data points in that leaf. If the class distribution of data points within a leaf node is heterogeneous, the node is considered *impure*. The formula to calculate the Gini impurity is given by:

$$G = \sum_{k \in C} p_k(1 - p_k).$$

where C is the set of classes and p_k the fraction of samples in that class. If there are only two classes denoted by class one and class zero, the Gini impurity becomes:

$$\begin{aligned} G &= \sum_{k \in \{1,0\}} p_k(1 - p_k) \\ &= p_1(1 - p_1) + p_0(1 - p_0) \\ &= p_1 - p_1^2 + p_0 - p_0^2 \\ &= 1 - p_0 - p_1^2 + p_0 - p_0^2 \\ &= 1 - (p_1^2 + p_0^2). \end{aligned}$$

Consider a leaf where each data point belongs to the same class r, then the fraction of data points p_r belonging to this class will be equal to one and the remaining fractions of points belonging to the other classes will all be equal to zero. Substitution of these fractions into the Gini impurity formula yields:

$$G = \sum_{k \in C} p_k(1 - p_k) = p_r(1 - p_r) + \sum_{k \in C, k \neq r} p_k(1 - p_k) = 0 + 0 = 0.$$

Clearly, pure leaves will have Gini impurity measures that approach zero. Classification algorithms like CART solve local optimization problems at each branch node where the sum of Gini impurity measures over all corresponding child nodes is minimized. Due to its local/greedy structure, and its failure to take all of the data into account at the same time, such an algorithm potentially fails to achieve a globally optimal solution.

3.3.1 Tree construction example CART

To understand the decision making process of CART, we will consider the following example: suppose we have a data set containing information about the weather conditions on several days and we wish to predict whether or not Peter will play a game of soccer on day fourteen. It is difficult to make a prediction by just looking at the data on the day of interest, therefore, we construct a model, based on data from previous days, that can help to make the prediction.

Day	**Outlook**	**Humidity**	**Wind**	**Play**
d1	sunny	high	weak	no
d2	sunny	high	strong	no
d3	overcast	high	weak	yes
d4	rainy	high	weak	yes
d5	rainy	normal	strong	no
d6	overcast	normal	strong	yes
d7	sunny	high	weak	no
d8	sunny	normal	weak	yes
d9	rainy	normal	weak	yes
d10	sunny	normal	strong	yes
d11	overcast	high	strong	yes
d12	overcast	normal	weak	yes
d13	rainy	high	strong	no
d14	rainy	normal	weak	?

Table 3.1: Labeled data set containing information about the weather on several days

There are three different features that can be used to split on: outlook, humidity and wind. Each feature has different outcomes, also referred to as states. In the first step of the algorithm CART compares the Gini impurities for each of the three features and selects the feature that results in the lowest Gini impurity. The Gini impurity of a feature is equal to the weighted average of the Gini impurities of the child nodes corresponding to the states of this feature.

First we will calculate the Gini impurity for the feature outlook by taking the weighted average of Gini impurities generated by the states sunny, overcast and rainy. It is defined by:

$$G_{\text{outlook}} = w_1 G_{\text{outlook,sunny}} + w_2 G_{\text{outlook,overcast}} + w_3 G_{\text{outlook,rainy}},$$

where w_1, w_2 and w_3 are the weights corresponding to the child nodes generated by respectively sunny, overcast and rainy. The weights are given by the fraction of points in each child node.

Figure 3.3 shows the subtree generated by the feature outlook. Each leaf is assigned the data points corresponding to the state of this feature, where black circles represent yes instances and white circles no instances. We can use this information to calculate for each state the Gini impurity measure.

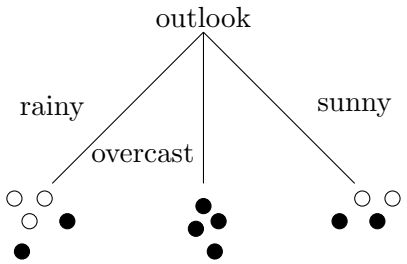


Figure 3.3: Subtree generated by the feature outlook

The Gini impurity for the leaf corresponding to sunny outlook is given by:

$$G_{\text{outlook,sunny}} = \sum_{k \in C} p_k(1 - p_k),$$

where $C = \{\text{yes}, \text{no}\}$, p_{yes} is the fraction of yes-instances in the leaf, and similar, p_{no} is the fraction of no-instances assigned to the leaf of interest:

$$p_{\text{yes}} = \frac{2}{4}$$
$$p_{\text{no}} = \frac{2}{4}.$$

The Gini impurity for the sunny outlook becomes:

$$\begin{aligned} G_{\text{outlook,sunny}} &= \sum_{k \in C} p_k(1 - p_k) \\ &= \frac{2}{4}(1 - \frac{2}{4}) + \frac{2}{4}(1 - \frac{2}{4}) \\ &= 0.5. \end{aligned}$$

The next step is to calculate the Gini impurities for the other states and take the weighted average:

$$\begin{aligned} G_{\text{outlook,sunny}} &= 0.5 \\ G_{\text{outlook,overcast}} &= 0 \\ G_{\text{outlook,rainy}} &= 0.48, \end{aligned}$$

$$\begin{aligned} G_{\text{outlook}} &= \frac{5}{13} * G_{\text{outlook,sunny}} + \frac{4}{13} * G_{\text{outlook,overcast}} + \frac{4}{13} * G_{\text{outlook,rainy}} \\ &= \frac{5}{14} * 0.5 + \frac{4}{14} * 0 + \frac{5}{14} * 0.48 \\ &= 0.35. \end{aligned}$$

CART compares the scores for all individual features and selects the one with the lowest value as split variable. This process will be repeated until the tree is complete or when some pre-sprecified stopping criterion is met. The classification tree obtained by using CART on the weather data set is depicted in Figure 3.4. This model can be used to predict whether or not Peter plays on day fourteen, a day with sunny outlook, normal humidity and weak wind, by following the corresponding path from root

node to the matching leaf. Based on this model the prediction would be positive.

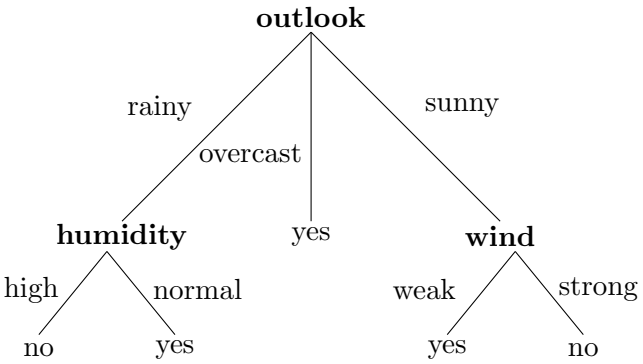


Figure 3.4: Classification tree for weather data

The example demonstrates that at each step of the algorithm CART selects the feature to split on by minimizing Gini impurities starting from the root node recursively to the leaf nodes. This top down approach where local minimization problems are solved does not necessarily yield optimal classification trees. In Section 4 we will introduce a different formulation for constructing classification trees by solving a global optimization problem based on mixed integer optimization that will generate optimal trees.

3.4 Multivariate classification trees

Up until now we have only discussed *univariate* classification trees that use single features to split on at each branch node. Trees that use multiple features to branch are known as *multivariate* trees. Instead of using one single variable, the tree constructing algorithm will define general hyperplanes to partition the data space at each branch node. Multivariate tree models can be constructed for numerical data sets or categorical data sets after encoding into numerical data.

Suppose we have a numerical, three features, data set containing labeled data from two different classes. This data can be represented in a 3-dimensional cube as depicted in Figure 3.5. Each circle represents a data point and the colour denotes the class. Multivariate classification trees recursively separate the data set into two partitions at each branch node by using general hyperplanes. In Figure 3.5 one single hyperplane separates the data set into two pure classes.

In the Euclidean p-dimension data space, $\mathbb{R}^p$ hyperplanes can be described by single linear equations of the form:

$$a_1 x_1 + \cdots + a_p x_p = b. \tag{3.1}$$

where all variables are real numbers. Partitioning of the data space via hyperplanes can be more effective than applying single feature splits. In Figure 3.6 one multivariate split and several univariate splits are applied on the same data set to partition the data. In the multivariate case only one split is needed to classify all data points correctly, while univariate trees need multiple splits. The

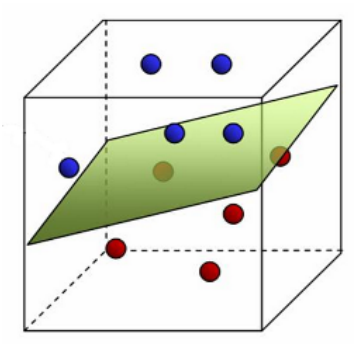


Figure 3.5: 3-dimensional binary class data together with a hyperplane separating the data space into two pure classes

corresponding classification trees will be of depth, respectively, one and two.

Figure 3.6: Data set with one multivariate split and three univariate splits showing the effectiveness of multivariate splits

3.5 Classification trees on genome data

Classification trees and other tree-based methods can be applied to a variety of informatics problems, such as high dimensional genome data analysis, where the number of samples is much smaller than the number of features [6]. There are roughly two major categories within this group of bioinformatics problems, classification and prediction problems and selection of important features [6]. The aim of this report is to classify *C. albicans* isolates and identify which mutations are possibly responsible for resistance, therefore, it can be viewed as mixture of both categories.

One of the objectives is to construct a classification tree model based on genome training data, usually in the form of sequences, to classify future genome data. Classification trees are applied in, e.g., sequence annotation, protein-function prediction, protein-protein interaction, classification between disease group and non-disease group and resistance classification, operating on both DNA level and amino acid level [6].

To illustrate how classification trees can operate as models for resistance classification we consider the following example. Suppose we are given three labeled sequences denoted by x_1, x_2 and x_3 as training data (3.2). The aim is to use this data to construct a classification tree model to help predict the unknown class of sequence x_4.

$$x_1 = \ldots \mathrm{ATT} \ldots (\mathrm{R}) \tag{3.2}$$

$$x_2 = \ldots \mathrm{ATC} \ldots (\mathrm{S})$$
$$x_3 = \ldots \mathrm{ATC} \ldots (\mathrm{S})$$
$$x_4 = \ldots \mathrm{GCC} \ldots (?)$$

The two susceptible sequences x_2 and x_3 differ from the resistant sequence x_1 at the third position. This feature can be used in the classification tree to split the data into two separated sets. Figure 3.7 shows a univariate classification tree based on the training data set (3.2). At the root node we apply a split: if at the third position the sequence contains a T it is placed in the left leaf; if the sequence contains a C on this position it branches to the right leaf. Each leaf is assigned the class that is most common among the sequences in that leaf. Hence, the right leaf is will be given the class susceptible and the left leaf resistant.

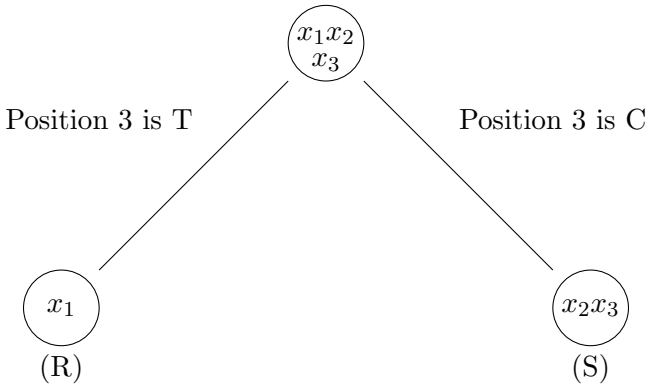


Figure 3.7: Classification tree for a genome data set

To help predict the class of sequences x_4 we can use the constructed tree model by following the right path from root to leaf. Every leaf is exactly assigned one class which prevents the classification from being ambivalent. According to the model from Figure 3.7, sequence x_4 is more likely to be susceptible.

A second objective is to identify important features that are involved in the expression of phenotypic traits. Univariate classification trees are easy interpretable, for each data point it is possible to extract the chain of decisions made by the tree that resulted in the class prediction for that data point. By analysing these paths from root to leaves, important features can be determined. In resistance classification for example, feature identification is crucial to understand which mutations are responsible for resistance. This information might be useful for the treatment of infections caused by resistance pathogens.

Concluding, classification trees are models that can be used to classify data and extract important features by recursive partitioning of the data into separated classes. Most tree constructing algorithms solve local optimization problems at each branch node, which usually does not yield optimal classification trees. In the next chapter we will present a method that solves the problem of finding optimal classification trees by using mixed integer optimization.

Chapter 4
Mixed integer optimization formulation

In this chapter we will formulate the problem of creating an optimal classification tree using mixed integer optimization. Unlike standard decision tree algorithms, this formulation matches the nature of the decision tree problem which involves a series of discrete decisions and discrete outcomes [2]. Bertsimas and Dunn (2017) claim that the construction of decision trees, as a series of discrete decisions, is best formulated as a mixed integer optimization problem.

Instead of minimizing local impurity measures at each branch node the MIO method applies global optimization of the misclassification error yielding optimal classification trees. Another advantage of the MIO based formulation is the possibility to model non-binary trees and add extensions to model for instance false positive and false negative errors, which can be useful in resistance classification where one type of error is preferred over the other.

First we present the MIO formulation for univariate and multivariate classification trees. We will roughly use the same formulation as presented in [2] for creating optimal classification trees, the only adjustments made are to fix minor typos that were present in the formulation presented in [2] and we use a new definition of epsilon suggested by Higler et al in [14]. The second part of this chapter will address the construction of classification tree models that distinguish false positive and false negative misclassification errors which was not considered by Bertsimas and Dunn.

4.1 Univariate optimal classification trees

We are given training data (x_i, y_i), for $i = 1, \ldots, n$, where each data point x_i contains p features, $x_i \in \mathbb{R}^p$ and label $y_i \in \{1, \ldots, K\}$ denoting the class to which the corresponding data point x_i belongs. Without loss of generality, we normalise the data points x_i such that, $\forall i, x_i \in [0, 1]^p$.

Given tree depth D, the corresponding tree of this depth has $T = 2^{D+1}-1$ nodes indexed by $t = 1, \ldots T$. The parent node of t is denoted by $p(t)$. We define $A_L(t)$ to be the set of all ancestors of node t whose left branch has been taken along the path from the root to node t. Similarly, $A_R(t)$ is the set of all ancestors whose right branch has been taken. The union of these two sets, $A_L(t) \cup A_R(t)$, is equal to the set of all ancestors of t denoted by $A(t)$.

We make a distinction between branch nodes and leaf nodes. Branch nodes are given by the set $\mathcal{T}_B = \{1, \ldots, \lfloor T/2 \rfloor\}$. Branch nodes apply splits of the form $a^T x < b$. If a point satisfies this split it will take the left branch, if not, it follows the right branch. Leaf nodes are denoted by the set $\mathcal{T}_B = \{\lfloor T/2 \rfloor, \ldots, T\}$, they are assigned a label to make a prediction for all points that end up in that leaf.

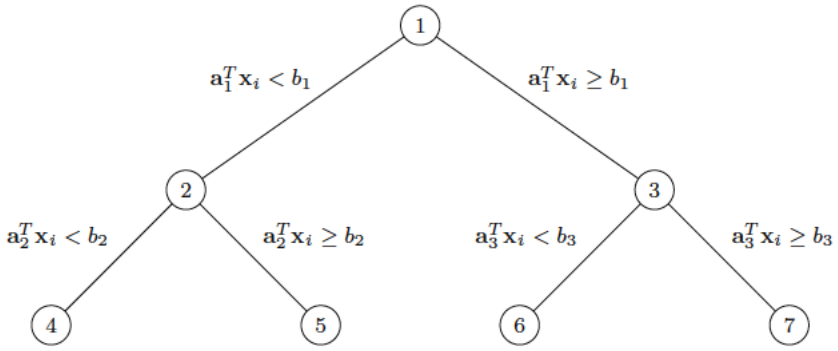


Figure 4.1: Tree of depth $D = 2$. Adapted from Optimal classification trees. Machine Learning. by Bertsimas, D., and Dunn, J. 2017, 106(7), 1039-1082.

4.1.1 MIO formulation for univariate trees

In the next part we will describe the MIO formulation in detail. All defined variables are presented at the end of this subsection in Table 8.1. For an overview of the constraints we refer to Table 8.2 in the Appendix. At the end of this subsection, several corrections are suggested and implemented in the original formulation by Bertsimas and Dunn (2017).

First we introduce variables $a_t \in \{0,1\}^p$ and $b_t \in \mathbb{R}$ for each branch node $t \in \mathcal{T}_B$, to keep track of the splits. Because we aim at generating a univariate classification tree, each split should only involve one single feature and therefore the elements of a_t need to sum up to one. To track which branch nodes apply splits we define indicator variables $d_t = \mathbb{1}\{\text{node } t \text{ applies a split}\}$. Variables a_t and b_t are both set to zero if a branch node does not apply a split. This forces all points to take the right branch. All of this can be modelled by the following constraints:

$$\sum_{j=1}^{p} a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B, \tag{4.1}$$

$$0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B. \tag{4.2}$$

To respect the structure of the tree, branch nodes should not apply a split if its parent does not apply a split. This is enforced by the constraint:

$$d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B. \tag{4.3}$$

To keep track of the points assigned to each leaf and to enforce a minimum number of points, $N_{\min}$, in each leaf we introduce indicator variables $z_{it} = \mathbb{1}\{x_i \text{ is in node } t\}$ and
$l_t = \mathbb{1}\{\text{leaf } t \text{ contains any points}\}$. Now for each leaf node we have:

$$z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L, \tag{4.4}$$

$$\sum_{i=1}^{n} z_{it} \geq l_t N_{\min}, \quad \forall t \in \mathcal{T}_L. \tag{4.5}$$

To force that each point is assigned to exactly one leaf the following constraint is added to the model:

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \dots n. \tag{4.6}$$

Every point that is assigned to a leaf followed a specific path from the root node to this particular leaf. To enforce the splits that are required for this path we apply two constraints:

$$a_m^T x_i < b_m + M_1(1 - z_{it}), \quad i = 1, \dots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{4.7}$$

$$a_m^T x_i \geq b_m - M_2(1 - z_{it}), \quad i = 1, \dots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \tag{4.8}$$

Constraint (4.7) enforces that, if point x_i is assigned to leaf t, then for each branching node m where the path from root node to leaf t took the left branch, $a_m^T x_i < b_m$ is satisfied. Similarly, constraint (4.8) enforces that for each branching node where the path took the right branch, $a_m^T x_i \geq b_m$ is satisfied.

Since strict inequalities can not be taken into account by MIO solvers, we need to introduce a small constant ϵ to the first inequality to make it non-strict:

$$a_m^T x_i + \epsilon \leq b_m + M_1(1 - z_{it}), \quad i = 1, \dots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t). \tag{4.9}$$

In theory this ϵ could be very small, however, taking ϵ too small can cause numerical instabilities in the MIO solver. On the other hand, if ϵ is chosen too big, it can affect the feasibility of valid solutions. To overcome this, ϵ is determined for each feature j separately by first sorting the values x_i^j of all data points for the jth feature and subsequently take the smallest difference of two adjacent values. Let

$$x_{(1)}^j \leq x_{(2)}^j \leq \dots \leq x_{(n)}^j$$

be the sorting of the values x_i^j of data points x_i for the jth feature. Then ϵ_j is defined for each feature as follows:

$$\epsilon_j = \min\{x_{(i+1)}^j - x_{(i)}^j | x_{(i+1)}^j \neq x_{(i)}^j, i = 1, \dots, n-1\}. \tag{4.10}$$

Note that in (4.10), $x_{(i)}^j$ represents the ith largest value of the jth feature amongst all data points which should not be confused with x_i^j, the value of data point x_i at feature j.

Instead of a scalar value, ϵ now represents a vector where each entry ϵ_j belongs to a corresponding feature j. Therefore, we need to add brackets to the equation to preserve validity

$$a_m^T(x_i + \epsilon) \leq b_m + M_1(1 - z_{it}), \quad i = 1, \dots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t). \tag{4.11}$$

Higler et al. [14] encounter a new problem is this formulation. By adding extra parentheses, both ϵ and x_i are multiplied by a_m^T. Observations show that modelling this constraint always lead to a solution where the objective function is equal to zero. Before, constraint (4.7) forced data points to take the right branch when no split was applied, with the new constraint (4.11) however, this is not enforced. The solver can simply set all variables a_m and b_m to zero and freely choose z_{it}, no splits will be applied and the solver can assign each data point arbitrarily [14].

To overcome this error Higler et al. [14] propose a new definition for ϵ where they define it as the minimum over all ϵ_j, i.e., $\epsilon = \min\{\epsilon_j | j = 1, \ldots, p\}$. Experiments demonstrate that this ϵ performs well in general. We will use their definition of ϵ throughout this report.

To specify the values for M_1 and M_2 we look at the largest possible value for $a_t^T x_i + \epsilon - b_t$. Since $a_t^T x_i \in [0, 1]$ we can set $M_1 = 1 + \epsilon$, the largest value for $b_t - a_t^T x_i$ is equal to one and hence M_2 is set to one. This results in the constraints (4.12) and (4.13) that we will use instead of constraints (4.7) - (4.11):

$$a_m^T x_i + \epsilon < b_m + (1 + \epsilon)(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{4.12}$$

$$a_m^T x_i \geq b_m - (1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \tag{4.13}$$

Now that we have defined the variables and constraints to model the tree structure we can determine the objective function. The goal is to minimize the the number of incorrect classifications. To model this, we first introduce class matrix Y given by:

$$Y_{ik} = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{otherwise} \end{cases}, \quad k = 1, \ldots, K, \quad i = 1, \ldots, n \tag{4.14}$$

N_{kt} is the number of points of label k in node t and N_t is the total number of points in leaf node t:

$$N_{kt} = \sum_{i=1}^{n} Y_{ik} z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{4.15}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L. \tag{4.16}$$

To keep track of the prediction at each leaf node we use binary variables $c_{kt} = \mathbb{1}\{c_t = k\}$, where c_t is a label given by $c_t = \text{argmax}_{K=1,\ldots,K}\{N_{kt}\}$, which is a most common label among all points in that leaf. To enforce that only one label is assigned to each leaf we add the constraint:

$$\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L. \tag{4.17}$$

Finally we define variables L_t to model the misclassification error at each leaf node. The variable L_t is equal to the total number of data points in leaf t, less the number of points labeled by the most common label in this leaf:

$$L_t = N_t - \max_{K=1,\ldots,K} \{N_{kt}\} = \min_{K=1,\ldots,K} \{N_t - N_{kt}\}, \tag{4.18}$$

which can be linearized by the following constraints

$$L_t \geq N_t - N_{kt} - M(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{4.19}$$

$$L_t \leq N_t - N_{kt} + M c_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L, \tag{4.20}$$

$$L_t \geq 0, \quad \forall t \in \mathcal{T}_L \tag{4.21}$$

where M should be large enough to make the constraints (4.19) and (4.20) automatically satisfied when $c_{kt} = 0$ and $c_{kt} = 1$ respectively. A sufficiently large number for M is n.

The objective is to minimize the total misclassification error. Therefore, we set the objective function as the sum of L_t over all leaf nodes t:

$$\min \sum_{t \in \mathcal{T}_L} L_t. \tag{4.22}$$

Combining all of this together yields the following MIO formulation for construction optimal classification trees:

$$\begin{aligned}
\min \ & \sum_{t \in \mathcal{T}_L} L_t && (4.23)\\
\text{s.t. } & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \leq N_t - N_{kt} + n c_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \geq 0, \quad \forall t \in \mathcal{T}_L,\\
& N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L,\\
& a_m^T x_i + \epsilon \leq b_m + M_1(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t)\\
& a_m^T x_i \geq b_m - M_2(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t)\\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n,\\
& z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{i=1}^{n} z_{it} \geq l_t N_{\min}, \quad \forall t \in \mathcal{T}_L\\
& \sum_{j=1}^{p} a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B,\\
& b_t \leq d_t, \quad \forall t \in \mathcal{T}_B,\\
& d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \backslash \{1\},\\
& z_{it}, l_t \in \{0, 1\}, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L,\\
& a_{jt}, d_t \in \{0, 1\}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B.
\end{aligned}$$

To get an overview of all constraints and variables that are introduced as part of the mixed integer optimization model for optimal classification trees we refer to Table 4.1 and Table 8.2 in the Appendix. The variables are categorised based on their type and function, the constraints are provided with a functional description.

4.1.2 Corrections on the original MIO formulation

We encountered a few obstacles in the MIO formulation presented by Bertismas and Dunn primarily caused by minor typos and a dubious implementation of ϵ in one of the constraints. In equations (4.4)-(4.5) and (4.7)-(4.8) they refer to branch nodes $\mathcal{T}_B$ instead of leaf nodes $\mathcal{T}_L$, in the present study this is corrected assuming it was a typo. Another more problematic difficulty has to do with the suggested definition and application of ϵ. Modelling the constraint

$$a_m^T(x_i + \epsilon) \leq b_m + M_1(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t)$$

where the vector entries are given by:

$$\epsilon_j = \min\{x^j_{(i+1)} - x^j_{(i)} | x^j_{(i+1)} \neq x^j_{(i)}, i = 1, \ldots, n-1\}$$

leads to false solutions as the objective value is set to zero for any data set. The MIO solver can allocate each data point arbitrarily and minimize the objective function by setting all variables a_m and b_m to zero [14]. In this study we use the definition of ϵ formulated by Higler et all. [14].

Variables		
Name	**Type**	**Function**
a_t	Binary	Split variable determining which feature to split on
b_t	Continuous	Split variable determining which branch direction to take
d_t	Binary	Indicator function to track which nodes apply splits
z_{it}	Binary	Indicator function to track the points assigned to each leaf
l_t	Binary	Indicator function to track if a leaf contains a point
Y_{ik}	$\{-1, 1\}$-Matrix	Label matrix
N_{kt}	Integer	Number of points of label k in node t
N_t	Integer	Total number of points in node t
c_t	$\{1, \ldots K\}$	Assigned label to each leaf given by the majority of the labels of all data points in that leaf
c_{kt}	Binary	Indicator function to track the prediction at each leaf
L_t	Integer	Misclassification loss in node t given by the total number of points in the node minus the number of points assigned the most common label

Table 4.1: Variables defined in the MIO formulation for univariate optimal classification trees

4.2 Multivariate optimal classification trees

An advantage of using MIO for tree construction is the simple modification from univariate trees to multivariate decision trees. Instead of using a single variable for each split, multivariate trees can split on several variables at a time. Only a few adjustments are required to convert the univariate algorithm into a multivariate tree algorithm with hyperplanes.

4.2.1 MIO formulation for multivariate trees

Instead of being restricted to split at single variables we can define hyperplane splits for each branch node. This means that variables a_t are not necessarily binary anymore, instead, $a_t \in [-1,1]^p$ and therefore we replace (4.1) by:

$$\sum_{j=1}^{p} |a_{jt}| \leq d_t, \quad \forall t \in \mathcal{T}_B,$$

to account for possibly negative elements. To linearize the absolute values we introduce auxiliary variables $\hat{a}_{jt}$ and add the constraints:

$$\begin{aligned}
&\sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, \quad \forall t \in \mathcal{T}_B, \\
&\hat{a}_{jt} \geq a_{jt}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B, \\
&\hat{a}_{jt} \geq -a_{jt}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B.
\end{aligned}$$

If a branch node t does not apply a split, variables a_t and b_t are set to zero. Since $a_t^T x_i \in [-1,1]$, we have:

$$-d_t \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B.$$

Constraints (4.7) and (4.8) are slightly modified to match the multivariate nature of the tree. Since the range of $(a_t^T x_i - b_t)$ is $[-2,2]$ instead of $[-1,1]$, we need to take $M = 2$, which ensures that the constraint is automatically satisfied when $z_{it} = 0$. Therefore, (4.7) and (4.8) are replaced by:

$$a_m^T x_i < b_m + 2(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{4.24}$$

$$a_m^T x_i \geq b_m - 2(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \tag{4.25}$$

Like in the univariate formulation we need to turn the strict inequality in constraint (4.24) into a non-strict inequality by adding a sufficiently small positive number μ

$$a_m^T x_i + \mu \leq b_m + (2 + \mu)(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t). \tag{4.26}$$

Since $(a_t^T x_i - b_t) \in [-2,2]$ and by adding the term $(2 + \mu)(1 - z_{it})$ on the right hand side of the equation, constraint (4.26) will automatically be satisfied when $z_{it} = 0$. It is not possible to choose

μ in a similar way as we chose ϵ in the univariate case, because there are infinitely many values for $a_t^T x_i$. Instead, μ is assigned the fixed number 0.005 as a compromise between stability of the solver and the feasible region size.

The final MIO model for construction multivariate classification trees is given by:

$$
\begin{aligned}
\min \ & \sum_{t \in \mathcal{T}_L} L_t && (4.27)\\
\text{s.t. } & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \leq N_t - N_{kt} + n c_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \geq 0, \quad \forall t \in \mathcal{T}_L,\\
& N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L,\\
& a_m^T x_i + \mu \leq b_m + (2 + \mu)(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t),\\
& a_m^T x_i \geq b_m - 2(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t),\\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n,\\
& z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, \quad \forall t \in \mathcal{T}_B,\\
& \hat{a}_{jt} \geq a_{jt}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B,\\
& \hat{a}_{jt} \geq -a_{jt}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B,\\
& -d_t \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B,\\
& d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \backslash \{1\},\\
& z_{it}, l_t \in \{0, 1\}, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L,\\
& d_t \in \{0, 1\}, \quad \forall t \in \mathcal{T}_B.
\end{aligned}
$$

4.3 Modelling false negative and false positive

In classification trees with binary target values, false positive and false negative errors are two types of errors whose union is the total misclassification error of the model. A false positive error in condition testing occurs when a sample is assumed to satisfy the tested condition while in reality it does not. When a sample does not satisfy the condition according to some classification algorithm, but it does in reality, it is called false negative.

Depending on the research area and the tested condition, avoiding one type of error may be more

important than avoiding the other. In fungal resistance testing for medication purposes, a false positive error i.e., classifying a fungal isolate as being resistant while in reality it is susceptible, is preferred over a false negative error. Despite unnecessary treatment with broad-spectrum antibiotics, specialists prefer this because at least the patient will recover from the fungal infection. False negative results can lead to a persevering infection because of ineffective treatment.

It is valuable to minimize the number of false negative errors in medical resistance classification. This can be done by penalizing each false negative misclassification by some constant valued parameter $\alpha \geq 1$. Based on several factors, including the medical impact of a false negative misclassification, parameter α is tuned to control the strength of the penalty.

In binary class classification there are three possible situations at each of the leaves: 1) the leaf is pure, it contains only data points from the same class, 2) there are only false negative allocations and 3) there are only false positive allocations. The optimal label to predict is the label of the majority of the data points in the leaf. Adding a sufficiently large penalty will affect the distribution of data points over the leaves, the total misclassification error might increase but the false negative error will decrease.

There are two classes in resistance classification, the susceptible class denoted by zero and the resistant class indexed by one. When a leaf t is assigned the susceptible label zero, all misallocated data points are false negative errors that should be penalized. The indicator function c_{kt} keeps track of the class prediction at each node t, therefore we wish to multiply the misclassification error L_t by α when $c_{0t} = 1$ and keep the error unchanged is the case where $c_{0t} = 0$. We can model this by replacing L_t in the original objective function (4.22) by the following term:

$$\big(c_{0t}(\alpha - 1) + 1\big)L_t.$$

The objective function becomes:

$$\min \sum_{t \in \mathcal{T}_L} \big(c_{0t}(\alpha - 1) + 1\big)L_t \tag{4.28}$$

Since c_{0t} is binary and L_t integer, the objective function (4.28) can be linearized by defining a new variable w_t equal to the product $c_{0t}L_t$ for each leaf t and adding the constraints:

$$w_t \leq L_t, \quad \forall t \in \mathcal{T}_L, \tag{4.29}$$

$$w_t \leq M c_{0t}, \quad \forall t \in \mathcal{T}_L, \tag{4.30}$$

$$w_t \in \mathbb{Z}, \quad \forall t \in \mathcal{T}_L. \tag{4.31}$$

M should be sufficiently large to make constraint 4.30 always satisfies when $c_{0t} = 1$. Since the misclassification error can be at most forty we set $M = 40$. The new objective function becomes:

$$\min \sum_{t \in \mathcal{T}_L} w_t(\alpha - 1) + L_t \tag{4.32}$$

We will call the MIO model with new objective function (4.32) and constraints (4.29)-(4.31) MIO-α.

Chapter 6 reports the in-sample and out-of-sample results obtained with this method.

Chapter 5
Non-binary classification trees

So far we have only considered binary classification tree models, univariate or multivariate, where at each branch node the data set is separated into two disjoint sets. In the case of categorical data where some of the features have more than two states, it can be beneficial and more natural to construct non-binary classification trees. Instead of branching in two directions, non-binary trees, for example, ternary, quaternary or quinary trees, branch in respectively three, four and five directions.

In this chapter we will discuss some of the advantages of non-binary trees over binary trees for different categorical data sets. Before we give a mixed integer program for constructing non-binary classification trees, we slightly modify the MIO problem (4.23) for univariate binary trees, to make it specific for the *C. albicans* and *C. glabrata* data sets. The result is a data specific and simplified model that can be used to explain the formulation for building non-binary classification tree models. After describing the simplified model, we present a novel mixed integer optimization based formulation for constructing ternary classification trees. Since the *C. glabrata* data set is binary at each feature, ternary trees and binary trees coincide, as will become clear later. Therefore, we will only focus on *C. albicans* in the ternary tree section. At the end of the chapter we show how to generalise the program for constructing ternary tree models to k-ary classification tree models.

5.1 Simplified univariate binary classification tree model for *C. albicans* and *C. glabrata*

After applying several preprocessing steps to remove redundant data, the *C. albicans* data set consists of 80 genome sequences having length 61. Since the MIO based algorithm only operates on numerical data, it is necessary to turn this categorical genome data into numerical data. A commonly practised encoding method uses *one-hot sequences*. A one-hot sequence is a sequence that contains a one at a single position, all others positions are equal to zero.

There are ten different symbols in the *C. albicans* data set that can be represented by unique one-hot sequences as follows:

$$\begin{aligned} A &= 1000000000, \\ C &= 0100000000, \\ T &= 0010000000, \\ &\vdots \\ K &= 0000000001. \end{aligned} \tag{5.1}$$

This one-hot sequence encoding enlarges the feature dimension of each data point by a tenfold. As a result, the *C. albicans* data set now contains 80 binary sequences of length 610 that is applicable as training data for constructing binary univariate classification trees. A slight modification of the

encoding method given in (5.1) can be used to turn categorical *C. glabrata* data into binary data. Since there are seven different symbols in this data set, we use a seven-bit one-hot sequence encoding.

To obtain the MIO formulation for data specific optimal classification trees, we can fix or even omit certain variables from the general MIO formulation (33). Since each data point is binary, $x_i \in \{0,1\}^p$, we can fix b_t for each branch node according to a predetermined topology. Because $a_t^T x_i \in \{0,1\}$, b_t can be set to zero for left branches and one for right branches, as depicted in Figure 5.1.

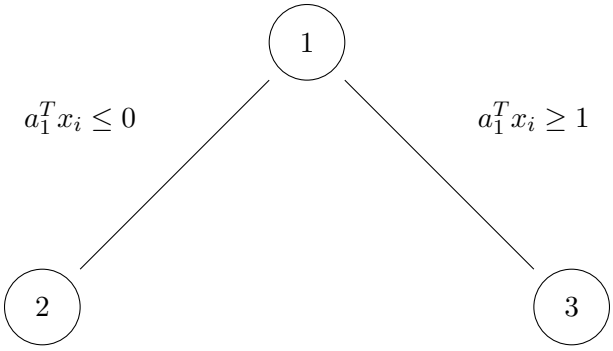


Figure 5.1: Binary tree topology

After fixing b_t for each branch node t, the constraint to model non-splits, $0 \leq b_t \leq d_t$, where $d_t = \mathbb{1}\{\text{node } t \text{ applies a split}\}$ becomes redundant. We can remove all variables d_t together with the corresponding constraints:

$$\begin{aligned} &0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B, \\ &d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B, \end{aligned}$$

and instead of

$$\sum_{j=1}^{p} a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B,$$

we will model the univariate character of the tree by the equality:

$$\sum_{j=1}^{p} a_{jt} = 1, \quad \forall t \in \mathcal{T}_B.$$

As a consequence, the algorithm can no longer decide not to apply a split, which means that pure leafs might be partitioned further without having any effect on the objective value. To prevent constructing unnecessarily complex trees, these branches should be pruned after obtaining the tree.

In the original formulation, for each data point assigned to a leaf, there are two constraints to enforce the splits that are required to obtain the correct path from the root node to that leaf:

$$\begin{aligned} &a_m^T x_i + \epsilon < b_m + (1+\epsilon)(1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \\ &a_m^T x_i \geq b_m - (1 - z_{it}), \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \end{aligned}$$

Since variables b_t are fixed and $a_t^T x_i \in \{0,1\}$, we can replace the previous constraints by;

$$a_m^T x_i - (1 - z_{it}) \leq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{5.2}$$

$$a_m^T x_i + (1 - z_{it}) \geq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t). \tag{5.3}$$

The resulting MIO formulation (5.4) to construct optimal data specific classification trees is less complex, there is no ϵ involved and we were able to remove various constraints and variables. The result is a more comprehensible formulation and faster operating algorithm on the *C. albicans* and *C. glabrata* data sets. Table 5.1 shows the time it takes to run the original MIO algorithm on *C. albicans* as presented in [2] and the data specific MIO algorithm as formulated in (5.4) for several train/test-distributions. For each distribution, the data specific MIO formulation operates faster because we were able to narrow down the number of variables and constraints in this formulation.

$$\begin{aligned}
\min \ & \sum_{t \in \mathcal{T}_L} L_t && (5.4)\\
\text{s.t. } & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \leq N_t - N_{kt} + n c_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& L_t \geq 0, \quad \forall t \in \mathcal{T}_L,\\
& N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,\\
& N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L,\\
& a_m^T x_i - (1 - z_{it}) \leq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t),\\
& a_m^T x_i + (1 - z_{it}) \geq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t)\\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n,\\
& z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L,\\
& \sum_{i=1}^{n} z_{it} \geq l_t N_{\min}, \quad \forall t \in \mathcal{T}_L\\
& \sum_{j=1}^{p} a_{jt} = 1, \quad \forall t \in \mathcal{T}_B,\\
& z_{it}, l_t \in \{0,1\}, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L,\\
& a_{jt} \in \{0,1\}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B
\end{aligned}$$

Method	Number of train sequences	Number of test sequences	Running time
Original	60	20	67.1 s
MIO formulation	50	30	39.6 s
	40	40	29.6 s
Data specific	60	20	14.7 s
MIO formulation	50	30	11.6 s
	40	40	8.2 s

Table 5.1: Running time using the data specific and original MIO formulation on different distributions of *C. albicans* data

5.2 Binary versus non-binary trees

Most classification tree constructing algorithms only support binary trees, however, in case of categorical data sets where features have multiple states, it might be desirable to use non-binary trees as a model to represent the data.

Consider a genome data set with four possible states, A,C,T and G, for each feature. At a split, partitioning of the data depends on the possible states of the selected feature. There are four different states, therefore, branching in four directions seems natural. Figure 5.2 shows a single split for genome data in a quaternary tree together with two equivalent binary trees. It takes multiple splits in a binary tree to represents the same single split in a quaternary tree.

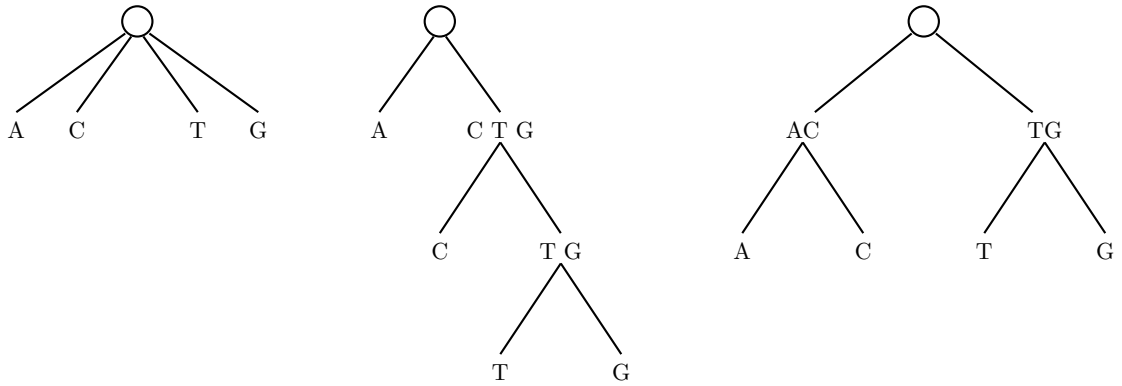


Figure 5.2: Quaternary tree and equivalent binary trees

Every k-ary tree has an equivalent binary tree, where each split in the first is represented in the latter by a binary subtree having depth at least $\log k$. Therefore, equivalent binary trees are much deeper than the corresponding k-ary tree, which can have an effect on the speed of the MIO based algorithm. Moreover, since each child node obtained by applying a split represents one of the possible k states of a feature, it is not necessary to use one-hot sequences encoding to turn the categorical data into binary data. This reduces the dimensionality of the data set significantly. A final advantage of non-binary trees is their interpretability, as can be seen in Figure 5.2, there are fewer obfuscating constructions

compared to binary trees.

5.3 Mixed integer optmization formulation for ternary classification trees

In this subsection we present the data specific MIO formulation for constructing univariate ternary optimal classification trees based on the *C. albicans* data set. Before we define the program, we explain how to model the tree topology and discuss an encoding method suitable for this model.

Since each feature/column in the *C. albicans* data set contains at most three different symbols, two main nucleotides and a third corresponding ambiguous nucleotide, it is natural to model the data using a ternary tree. Instead of using a ten-bit binary encoding, we can, column-wise, assign to each nucleotide a unique numerical value from the unit interval.

For binary data sets it was possible to model each split with inequalities $a_j^T x_i \leq 0$ and $a_j^T x_i \geq 1$, as shown in Figure 5.1. If the first equation holds for data point x_i, it will be placed in the left child node and if the second equation is satisfied, x_i will follow the right branch. Adding an extra value to the data set hinders the use of inequalities to define the splits, therefore, we use equalities instead. Figure 5.3 shows a ternary split defined by three equalities given by numerical values, $0, 0.5$ and 1. These numerical values and tree structure will be used for the *C. albicans* data specific ternary tree formulation explained in the next subsection.

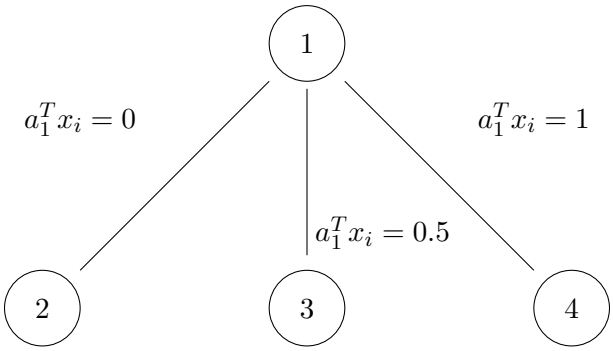


Figure 5.3: Ternary tree structure

5.3.1 MIO formulation for ternary trees

To model the ternary tree structure and obtain optimal classification trees, we need to replace several constraints and definitions from formulation (5.4). First of all, the number of nodes T for ternary trees is equal to $\frac{3^{D+1}-1}{2}$, where D is the depth of the tree. Moreover, the set of branch nodes $\mathcal{T}_B = \{1, \dots \lfloor T/3 \rfloor\}$ and the set of leaf nodes $\mathcal{T}_L = \{\lfloor T/3 \rfloor, \dots, T\}$. We also define $A_M(t)$ to be the set of all ancestors whose middle branch was taken along the path from the root to leaf node t.

To make sure each leaf assigned data point took the right path from root to leaf, respecting the tree structure given in Figure 5.3, we replace constraints (5.2) and (5.3) by the following six constraints:

$$a_m^T x_i - (1 - z_{it}) \leq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{5.5}$$
$$a_m^T x_i + (1 - z_{it}) \geq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{5.6}$$
$$a_m^T x_i - (1 - z_{it}) \leq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t), \tag{5.7}$$
$$a_m^T x_i + (1 - z_{it}) \geq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t), \tag{5.8}$$
$$a_m^T x_i - (1 - z_{it}) \leq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t), \tag{5.9}$$
$$a_m^T x_i + (1 - z_{it}) \geq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t). \tag{5.10}$$

Since $0 \leq a_m^T x_i \leq 1$, for $z_{it} = 0$ and $z_{it} = 1$, we have $a_m^T x_i - (1 - z_{it}) \leq 1$ and $a_m^T x_i + (1 - z_{it}) \geq 0$, and therefore, inequalities (5.6) and (5.7) are always satisfied. We can remove these constraints ending up with the following four constraints:

$$a_m^T x_i - (1 - z_{it}) \leq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t), \tag{5.11}$$
$$a_m^T x_i + (1 - z_{it}) \geq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t), \tag{5.12}$$
$$a_m^T x_i - (1 - z_{it}) \leq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t), \tag{5.13}$$
$$a_m^T x_i + (1 - z_{it}) \geq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t). \tag{5.14}$$

The final *C. abicans* data specific MIO formulation for optimal ternary univariate classification trees is presented below:

$$\min \sum_{t \in \mathcal{T}_L} L_t \tag{5.15}$$
$$\text{s.t. } L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,$$
$$L_t \leq N_t - N_{kt} + n c_{kt}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,$$
$$L_t \geq 0, \quad \forall t \in \mathcal{T}_L,$$
$$N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, \quad k = 1, \ldots, K, \quad \forall t \in \mathcal{T}_L,$$
$$N_t = \sum_{i=1}^{n} z_{it}, \quad \forall t \in \mathcal{T}_L,$$
$$\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L,$$
$$a_m^T x_i - (1 - z_{it}) \leq 0, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_L(t),$$
$$a_m^T x_i + (1 - z_{it}) \geq 1, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_R(t),$$

$$
\begin{aligned}
&a_m^T x_i - (1 - z_{it}) \leq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t), \\
&a_m^T x_i + (1 - z_{it}) \geq 0.5, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_M(t), \\
&\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n, \\
&z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_L, \\
&\sum_{i=1}^{n} z_{it} \geq l_t N_{\min}, \quad \forall t \in \mathcal{T}_L \\
&\sum_{j=1}^{p} a_{jt} = 1, \quad \forall t \in \mathcal{T}_B, \\
&z_{it}, l_t \in \{0, 1\}, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \\
&a_{jt} \in \{0, 1\}, \quad j = 1, \ldots, p, \quad \forall t \in \mathcal{T}_B
\end{aligned}
$$

5.3.2 Higher degree trees

The method for constructing data specific ternary optimal classification trees can be generalised to obtain k-ary trees, which can be useful for the classification of categorical data where each feature has k states.

To encode the data we use numerical values $\mu_j \in [0, 1]$ for $j = 1, \ldots, k$, where each state is uniquely encoded by one of the μ_j. The total number of nodes T in a k-ary tree is equal to $\sum_{i=1}^{D} k^i = \frac{k^{D+1}-1}{k-1}$, the set of branch nodes $\mathcal{T}_B = \{1, \ldots \lfloor T/k \rfloor\}$ and the set of leaf nodes $\mathcal{T}_L = \{\lfloor T/k \rfloor, \ldots, T\}$. The k branch directions at each branch node are defined by equations of the form $a_t^T x_i = \mu_j$, for $j = 1 \ldots k$. Figure 5.4 shows the structure of a depth one k-ary tree.

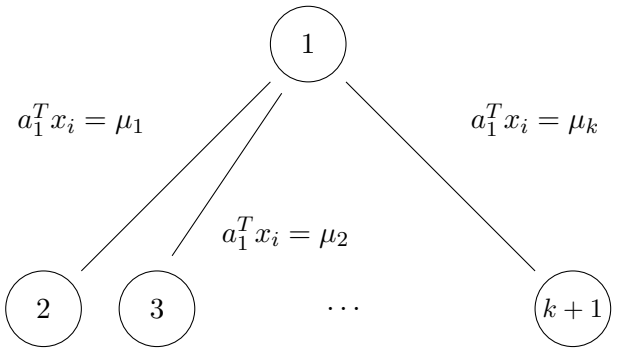


Figure 5.4: K-ary tree structure

For each equation $a_t^T x_i = \mu_j$ that defines a branch direction, we need to add two constraints:

$$
\begin{aligned}
&a_m^T x_i - (1 - z_{it}) \leq \mu_j, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_j(t), \\
&a_m^T x_i + (1 - z_{it}) \geq \mu_j, \quad i = 1, \ldots, n, \quad \forall t \in \mathcal{T}_L, \quad \forall m \in A_j(t),
\end{aligned}
$$

replacing constraints (5.12)-(5.14) from the ternary tree MIO program. In these constraints, $A_j(t)$ is the set of ancestors of leaf node t, where the j-th branch, indexing from left to right, was taken along the path from the root to node t.

Chapter 6
Results

This chapter contains the results obtained by applying several tree constructing algorithms on the *C. albicans* and *C. glabrata* data sets. To examine the performance of an algorithm we distinguish in-sample and out-of-sample accuracy. In-sample accuracy is the fraction of training sequences incorrectly classified by the model and out-of-sample accuracy is the fraction of misclassified test sequences. Note that the objective of MIO based classification trees, as given in (4.23), is to minimize the in-sample accuracy. We will compare the algorithms in terms of in-sample and out-of-sample accuracy, varying the number of test and training sequences for a range of tree depths. In all analyses we construct multiple trees and take the average result to minimize the effect of outliers.

First, we will focus on univariate binary classification trees, with and without false negative penalization, after this we consider multivariate binary classification trees. We will use a Linear Support Vector Machine (LSVM) algorithm [32] provided by the scikit-learn Python package [25] and the MIO based algorithm for univariate classification trees as benchmarks for multivariate trees. To extract mutations possibly related to resistance, we apply univariate MIO based binary and quaternary trees on the data and compare the obtained mutations to previously reported mutations. This analysis is done on both nucleotide and amino acid levels. Finally, non-binary trees are constructed for the two *Candida* data sets and compared to binary trees having the same or higher tree depth.

6.1 Tree depth selection for *Candida albicans* and *Candida glabrata*

One of the parameters that needs to be set before running any tree constructing algorithm is the tree depth. This parameter controls the complexity of the tree, as it bounds the number of splits applied in the tree constructing process. If a tree grows too deep, it might depend too much on irrelevant features from the training data, which can result in poor performance on test data. This is usually referred to as *overfitting* [19]. Obviously, in-sample accuracy results improve as the tree grows deeper until a perfect is tree is obtained.

A method for finding suitable tree depths is via loops and cross-validation. There are several steps in this method, the first step is to select a range of tree depths to use as a for loop. This selection should contain a wide range of different tree depths. Inside the loop, the data is split into training and test data, for example, 70% training data and 30% test data. We train multiple classification trees on the generated training data and measure their performance on test data. We keep the average out-of-sample misclassification error for each depth and compare for which depth the error is minimized.

We apply this method on the *C. albicans* data set, where, guided by practical limitations of the time, CART is used as a tree constructing algorithm for depths $D = 1, \ldots, 20$. For each depth, hundred trees are repeatedly constructed based on sixty randomly selected sequences of training data, the constructed trees are then tested by using the remaining twenty sequences as test data. The average

out-of-sample misclassification error is stored for each depth D. Figure 6.1 shows the average out-of-sample misclassification error for the *C. albicans* data set. We select the depth corresponding to the smallest out-of-sample error which turns out to be equal to two.

To determine a suitable tree depth for the *C. glabrata* data set, we apply the same method as described above. However, the differences in the out-of-sample error between most tree depths are very small. As can be seen in Figure 6.2, for $D \geq 8$, the performances are approximately the same. This might be attributed to the fact that there are some strong splits, generating pure leaves, present at the top levels of the tree. Each constructed tree model will split at these features resulting in approximately the same out-of-sample error. Because of computational time limitations induced by the MIO method, we will restrict to trees having $D \leq 3$.

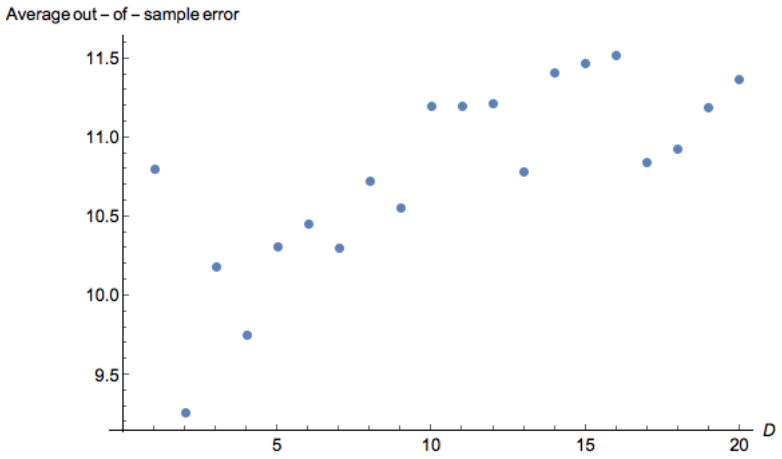


Figure 6.1: *C. albicans* average out-of-sample error for several tree depths D

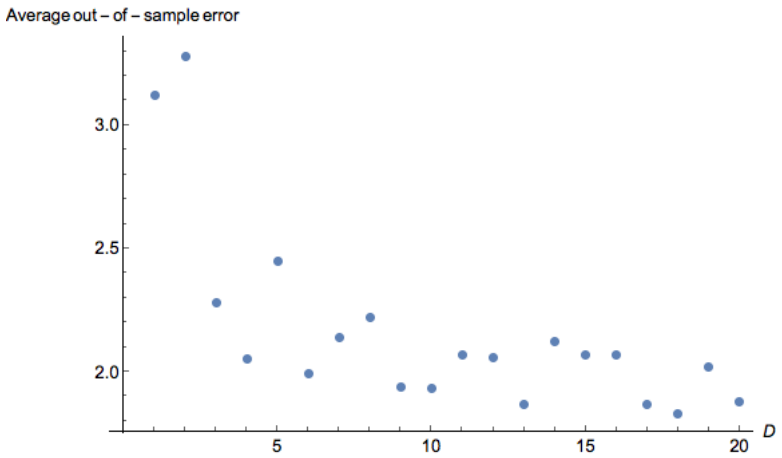


Figure 6.2: *C. glabrata* average out-of-sample error for several tree depths D

6.2 Univariate binary tree results

The MIO formulation for univariate binary trees, that was discussed extensively in Chapter 4, provides optimal classification trees. In this section we will compare binary univariate trees, obtained by applying the MIO formulation, to non-optimal trees constructed via CART. First we present several optimality results on the *C. albicans* and *C. glabrata* data sets, followed by comparative out-of-sample analysis using both methods. At the end of this section, we discuss novel results obtained by adding false negative error penalization to the MIO model.

6.2.1 Comparing MIO to CART on the full *C. albicans* and *C. glabrata* data sets

Figure 6.3 shows a binary univariate classification tree for $D = 2$ constructed by CART using the complete *C. albicans* data set as training data. The optimal label to predict at each leaf is the majority of the labels of all data points in that leaf. The leaves will be assigned, from left to right, respectively labels $1, 0, 0$ and 1, where label 0 corresponds to susceptible sequences and label 1 denotes the resistant class. Establishing the allocation of the labels enables us to calculate the misclassification error at each leaf node, which is given by the total number of samples in the node less the number of points of the established label. The total misclassification error L is equal to the sum of misclassification errors L_t over all leaves:

$$L = \sum_{t \in \mathcal{T}_L} L_t = (36 - 13) + (9 - 2) + (13 - 0) + (5 - 2) = 17.$$

Note that the feature numbers in the classification tree from Figure 6.3 do not correspond to real positions on the ERG11 gene, since binary preprocessing is necessary before applying CART and redundant columns have been removed. In section 6.4 we will extract the actual positions on ERG11 possibly related to resistance.

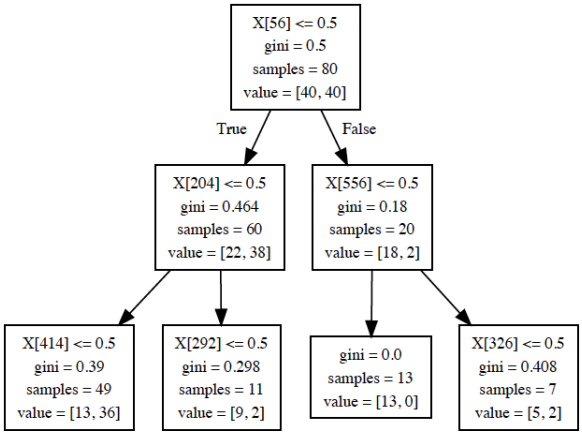


Figure 6.3: Classification tree of depth two obtained by CART on the *C. albicans* data set

The depth two classification tree model obtained by applying MIO on the *C. albicans* data set is presented in Figure 6.4. It turns out that the objective value is equal to sixteen, which is less than the

objective value obtained by CART. Apparently, it is better to apply the weaker split $X[204]$ before the strong split $X[56]$ to obtain a smaller misclassification error generated by these two splits. This emphasises the observation that using MIO allows to consider the possible impact of future splits in the tree and provides optimal classification trees by globally minimizing the misclassification error.

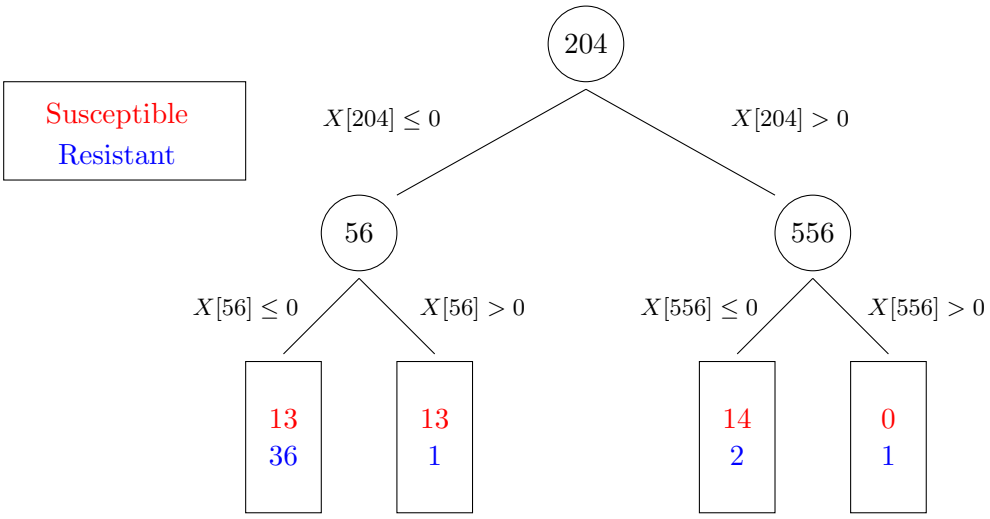


Figure 6.4: MIO classification tree of depth two on the *C. albicans* data set

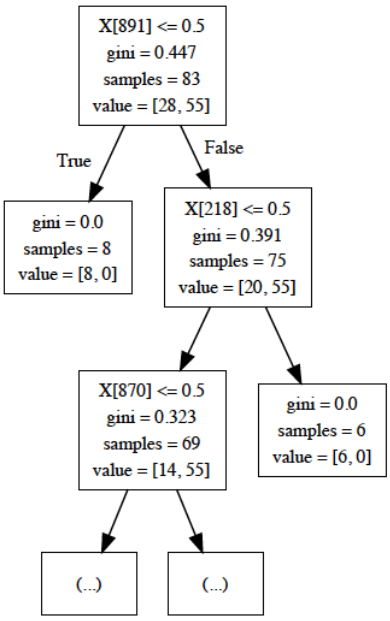


Figure 6.5: CART classification tree of depth two on the *C. glabrata* data set

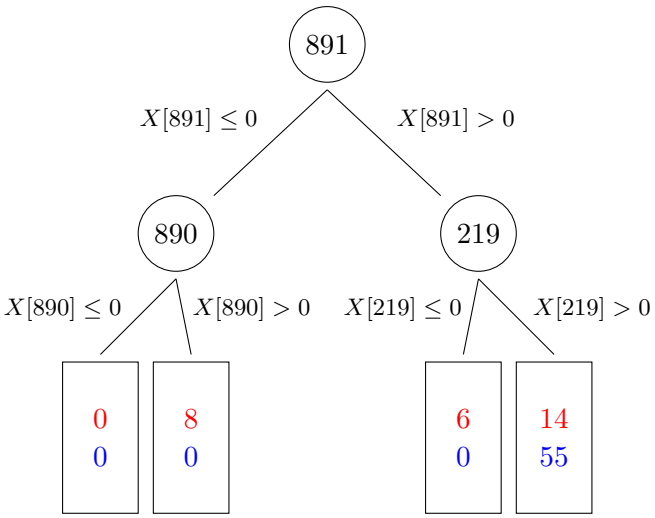


Figure 6.6: MIO classification tree of depth two on the *C. glabrata* data set

Classification trees for the *C. glabrata* data set are depicted in Figure 6.5 and Figure 6.6. The misclassification errors are equal for both trees and, since we used a seven-bit encoding, features 219 and 218 represent the same position on FKS1. The most left leaf in Figure 6.6 is empty, as a result of the removal of variables d_t and corresponding constraints in the data specific formulation 5.4, the algorithm can not decide to model none-splits. Therefore, the tree should be pruned afterwards by removing the unnecessary split that uses feature 890.

A major disadvantage of the approach proposed by Bertsimas and Dunn, is the difficulty of dealing with high dimensional data. Even though there has been an enormous increase in the speed of state-of-the-art MIO solvers, if the number of features is large, which is usually the case in genome data analysis, the high number of variables in the MIO formulation will be problematic. Furthermore, finding decision trees for large data sets containing thousands of points can only be done efficiently for depth up to four, exceeding this depth or data set size will significantly slow down the process of finding an optimal solution [2]. Table 6.1 shows the time it takes to construct depth two and depth three classification trees using MIO for the *C. albicans* data set, the difference in running time between both trees is significant.

Method	Depth	Running time
MIO	2	15 s
	3	3965 s

Table 6.1: Running time for constructing classification trees of depth two and three using MIO

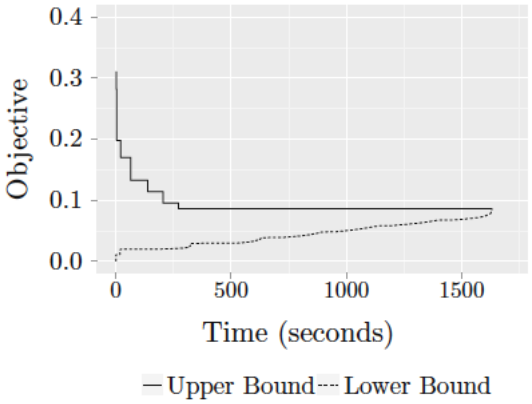


Figure 6.7: Time it takes to find and optimal solution for the UCI Wine data set

Proving optimality of the solution is often very time consuming. Figure 6.7 shows the evolution of the upper and lower bound, while solving the MIO formulation for finding an optimal depth two tree for the UCI Wine data set retrieved from [10]. After less than 300 seconds the upper bound remains unchanged, the remaining time, approximately 1200 seconds, is needed to prove optimality. One way to speed up the process is by terminating the algorithm once the solutions remains unchanged for a fixed period of time, another option is to inject a warm start solution to the MIO solver [2].

6.2.2 Comparing MIO to CART on test and training data

Optimal classification trees do not necessarily perform better on out-of-sample data comparing to non-optimal trees, which can possibly be attributed to overfitting. Bertsimas and Dunn, however, claim to have "strong evidence against the widely-held belief that optimal methods are more inclined to overfit the training set at the expense of out-of-sample accuracy" [2, p. 1061]. In this subsection we present the results on out-of-sample data for the *C. albicans* and *C. glabrata* data sets. For extensive results on other data sets we refer to [2].

Table 6.2 shows in-sample and out-of-sample results for several train/test-distributions from the *C. albicans* data set. For each distribution twenty trees are constructed and tested using both methods. Trees obtained by CART outperform MIO based classification trees for most distributions on out-of-sample accuracy. Obviously, since MIO trees are optimal, they will perform better on in-sample data

in comparison to CART trees. The same analyses are performed on the *C. glabrata* data set. The results in Table 6.3 indicate that, similar to the *C. albicans* data set, CART outperforms MIO based trees on out-of-sample data, but performs worse on in-sample data.

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
CART	70	10	78.5 %	73 %
D=2	60	20	78.9 %	70.8 %
	50	30	80.3 %	65.7 %
	40	40	82.4 %	68.4 %
MIO	70	10	81 %	71.5 %
D=2	60	20	81.6 %	68.2 %
	50	30	83.6 %	67.5 %
	40	40	84.8 %	65.4 %

Table 6.2: Accuracy results for univariate depth two trees on *C. albicans* data obtained by MIO and CART

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
CART	73	10	83.2 %	77 %
D=2	63	20	83.7 %	74.5 %
	53	30	85.2 %	74.5 %
	43	40	86 %	73.4 %
MIO	73	10	84 %	73 %
D=2	63	20	84.8 %	74.3 %
	53	30	86.7 %	72.5 %
	43	40	87.6 %	72.3 %

Table 6.3: Accuracy results for univariate depth two trees on *C. glabrata* data obtained by MIO and CART

It is remarkable that CART performs better on test data while MIO outperforms CART on training data. Intuitively it seems logical that, in the situation of comparing two equally deep trees on the same test and training data, the one that fits the training data the best also outperform the other on test data. However, this is not the case for *C. albicans* and *C. glabrata* data as can be seen in Tables 6.2 and 6.3, contradicting the claim that was made by Bertsimas and Dunn at the beginning of this subsection about the out-of-sample performance of optimal classification trees.

6.2.3 Univariate trees with false negative penalization

In many classification problems, especially medical classification, it is important to consider the effects of the different types of error. The UCI Machine Learning Repository, for example, contains multiple data sets e.g., cervical cancer (Risk Factors), Chronic—Kidney—Disease and Autistic Spectrum Disorder Screening Data for Children, that could benefit from error based penalization.

Figure 6.8 shows the false negative, false positive and total misclassification errors as functions of α for the *C. albicans* data set obtained by applying a univariate decision tree of depth two with false negative objective function as defined in Section 4.3. Clearly, the false negative error decreases when α, the penalty factor for false negative misclassifications, is increasing sufficiently. In the case where $\alpha = 2$, the algorithm decides to accept the penalty for the three misallocated data points, the distribution of points remains unchanged but the total misclassification error increases as a result of the extra weight of each false negative error. When further increasing α, the number of false negative errors decreases and the false positive errors increase until the system reaches equilibrium, where there are no false negative errors anymore.

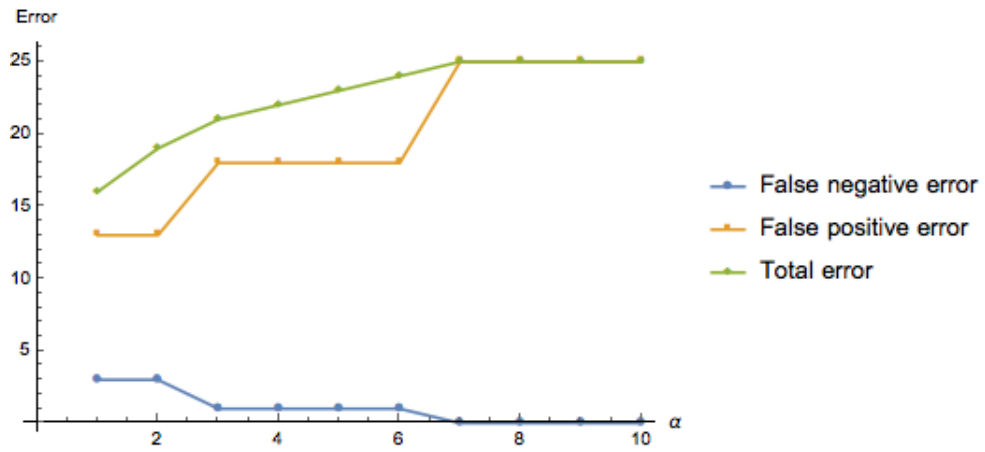


Figure 6.8: Misclassification errors for different values of penalty variable α

More important are the effects on out-of-sample data. It turns out that false negative penalization in the model construction process results in a significant decrease in out-of-sample false negative errors. As a consequence, the total out-of-sample misclassfication error increases, however, to a lesser extent. The out-of-sample results on the *C. albicans* data set are presented in Table 6.4, results for *C. glabrata* are given in Table 6.5. We compare univariate depth two MIO models without penalization to univariate depth two MIO models with penalty factor α, as presented in Section 4.3. The last column shows the out-of-sample false negative misclassification as percentage of the total misclassification error. As can be seen in both tables, the fraction of false negative misclassification decreases when the model uses a false negative penalizing objective function. For *C. albicans*, the fraction decreases 64 - 80% and for *C. glabrata*, 56 - 66%.

Features are selected by the α depended model in such a way that leaves with susceptible class assignment only contain training data points assigned the susceptible class, to prevent from penalization. Experiments show that trees obtained by the MIO-α model contain significantly more pure susceptible class assigned leaves. Decision paths towards these leaves are strongly biased towards susceptible sequences. The presence of more susceptible pure leaves might be an explanation for the out-of-sample false negative error decrease in MIO-α obtained classification trees.

Method	Number of train sequences	Number of test sequences	Out-of-sample accuracy	Percentage of false negative errors
MIO-α	60	20	66 %	11.8 %
$\alpha = 8$	50	30	66.7 %	8 %
	40	40	68.5 %	14 %
MIO	60	20	70 %	46.7 %
	50	30	69.3 %	41.3 %
	40	40	66.3 %	39.3 %

Table 6.4: Out-of-sample accuracy results and out-of-sample false negative misclassification as percentage of the total misclassification error for *C. albicans*

Method	Number of train sequences	Number of test sequences	Out-of-sample accuracy	Percentage of false negative errors
MIO-α	63	20	67.5 %	18.5 %
$\alpha = 8$	53	30	67.3 %	17.3 %
	43	40	66.3 %	13.3 %
MIO	63	20	73 %	42.6 %
	53	30	66.3 %	51.5 %
	43	40	68.3 %	31.5 %

Table 6.5: Out-of-sample accuracy results and out-of-sample false negative misclassification as percentage of the total misclassification error for *C. glabrata*

6.3 Multivariate binary tree results

Instead of restricting to single features at each split, we now consider splits using multiple features at once to construct MIO based multivariate binary decision trees (MIOM) for the *C. albicans* and *C. glabrata* data sets. As was explained in Section 3.4, each split in multivariate classification trees is generated by a general hyperplane separating the data space into two disjoint spaces. For both data sets it is possible to separate most of the resistant isolates from the susceptible ones by applying a single split. First we compare multivariate trees to an LSVM classification algorithm and the MIO algorithm for univariate classification trees on in-sample and out-of-sample performance. After this, we consider multivariate binary trees where the split generating hyperplanes are shifted up or down to generate better splits in the classification tree. The obtained trees are then being compared to multivariate trees without shifting the hyperplanes.

6.3.1 In-sample and out-of-sample analyses

In this subsection we present the out-of-sample results obtained by applying multivariate classification trees on the *C. albicans* and *C. glabrata* data sets. For more multivariate tree results on other data sets we refer to [2]. It is possible to split both data sets into two nearly perfect partitions by using one single split. Therefore, we will only consider depth one classification trees where a single split is applied to separate the data. In the MIO formulation for multivariate trees as defined in 4.27, constant

μ, to turn the strict inequality (4.24) into non-strict inequality (4.26), needs to be set before running the algorithm. Experiments show that setting $\mu = 0.0001$ will make the algorithm perform well on in-sample accuracy for various *C. albicans* training set sizes, in the next out-of-sample analysis we will therefore use this value for μ. For the *C. glabrata* data set, we set $\mu = 0.001$ to obtain good in-sample accuracy results.

Tables 6.6 and 6.7 show in-sample and out-of-sample MIOM results for *C. albicans* and *C. glabrata* data sets using LSVM and univariate MIO depth two trees as benchmarks. As can be seen in the tables multivariate trees (MIOM) perform very well on in-sample data. For most randomly generated training sets it is possible to separate the data space into two perfect classes after applying a single split. The in-sample accuracy for LSVM is comparable to MIOM. The high in-sample accuracy can be attributed to the large number of features of the data which increases the probability of finding a perfect split. On out-of-sample accuracy, on the other hand, MIOM and LSVM perform poorly. A possible reason for this poor performance is the so called *Curse of Dimensionality*, which is a phenomenon related to overfitting in high dimensional data analysis. It explains how increasing the number of features will decrease the density of data points in the data space, which makes it easier to find a separating hyperplane on the training data. However, in a lower dimensional space this classification hyperplane can represent a complex non-linear classification function that is very specific for the training data and performs poorly on test data. To learn a good function for out-of-sample classification, one needs enough data covering all regions in the space sufficiently well [33].

Comparing to univariate depth two classification trees (MIO), multivariate trees perform stronger on in-sample accuracy but worse on out-of-sample accuracy for each train/test-distribution for the *C. albicans* data set. In univariate classification trees, splits are generated by single features or, equivalent, axis aligned hyperplanes. In Section 3.4 it was explained and illustrated with Figure 3.6, that partitioning via general hyperplanes can be more effective than applying single feature splits. For *C. albicans* and *C. glabrata* it is possible to split the data into two nearly perfect partitions using one general hyperplane. As the results show it is not possible to obtain similar high-quality partitions by applying three axis aligned splits in depth two univariate trees. In terms of out-sample-accuracy, MIOM performs worse than MIO on the *C. albicans* data set which could be attributed to Curse of Dimensionality. For *C. glabrata* we see different results with regards to out-of-sample accuracy. Multivariate trees perform better than univariate trees for each distribution. Perhaps the data is linear separable by using a low dimensional hyperplane with little non-zero coefficients.

6.3.2 Shifted hyperplanes

The level of in-sample accuracy, and possibly out-of-sample accuracy, depends strongly on the value of μ in constraint (4.26), that needs to be fixed before running the algorithm. Table 6.8 shows the in-sample misclassification error for various values of μ for the *C. albicans* data set. The real in-sample error, obtained by manually checking the assignment of each training data point, fluctuates dramatically and differs significantly from the in-sample error outputted by the MIO algorithm. This difference is caused by numerical instability as a consequence of μ in equation (4.26). Data points, with distance less than μ to the hyperplane, end up on the other side of the hyperplane when manually

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
MIOM	70	10	98.9 %	63.5 %
	60	20	99.3 %	63 %
	50	30	99.7 %	64.8 %
LSVM	70	10	98.9 %	58.5 %
	60	20	99.4 %	61.3 %
	50	30	99.6 %	67.3 %
MIO	70	10	80.9 %	71 %
	60	20	82 %	67.5 %
	50	30	82.6 %	68.7 %

Table 6.6: MIOM and LSVM accuracy results for *C. albicans*

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
MIOM	73	10	95.1 %	76.5 %
	63	20	97.9 %	82.5 %
	53	30	96.4 %	79.2 %
LSVM	73	10	100 %	89 %
	63	20	100 %	88.8 %
	53	30	100 %	86 %
MIO	73	10	84.5 %	65.5 %
	63	20	85.4 %	73.3 %
	53	30	86.8 %	69.5 %

Table 6.7: MIOM and LSVM accuracy results for *C. glabrata*

checked, resulting in a different in-sample error compared to the MIO algorithmic output. The number of data points close to the hyperplane and the value of μ determine the real in-sample error explaining the fluctuation.

A solution for the discrepancy between the in-sample error as given by the algorithm and the manually checked error, is by shifting the split generating hyperplane slightly up or down, such that misclassified data points close to the hyperplane will end up in the correct partition. To shift a hyperplane we add a constant value $\gamma \in \mathbb{R}$ to the right-hand side of the equation for hyperplanes (3.1), resulting in the following formula for the shifted hyperplane:

$$a_1 x_1 + \cdots + a_p x_p = b + \gamma$$

To select a suitable value for parameter γ, we loop over a range of decimal numbers $1/10^i$, for $i = 1, \ldots 10$ and choose the value that minimizes the in-sample error. The results in Tables 6.9 and 6.10 show improvements of the in-sample accuracy while the out-of-sample accuracy remains approximately the same for shifted multivariate optimal classification trees (MIOM-shift). For *C. glabrata* it is possible to obtain perfect classification after shifting the hyperplane slightly up or down, which was also observed in Table 6.7 for LSVM on this data set.

μ	Real in-sample error	MIO output in-sample error	Shifted in-sample error	Shift
0.1	7	7	7	0
0.005	15	1	1	-0.001
0.001	5	1	1	-0.0001
0.0005	7	1	1	-0.0001
0.0001	1	1	1	0
$5e^{-5}$	5	1	1	$-1e^{-5}$
$1e^{-5}$	40	0	39	$-1e^{-5}$
$5e^{-6}$	8	0	1	$-1e^{-6}$
$1e^{-6}$	1	1	1	0
$5e^{-7}$	40	1	40	0
$1e^{-7}$	40	0	40	0

Table 6.8: In-sample accuracy results for various parameter values of μ on *C. albicans* data

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
MIOM	70	10	94.6 %	58 %
	60	20	95.7 %	58.7 %
	50	30	95.5 %	59.5 %
MIOM-shift	70	10	98.9 %	58.5 %
	60	20	99.7 %	59.8 %
	50	30	99.9 %	60.3 %

Table 6.9: Accuracy results for general hyperplane splits and shifted hyperplane splits on the *C. albicans* data set with $\mu = 0.0001$

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
MIOM	73	10	96.4 %	78.5 %
	63	20	99.2 %	82.2 %
	53	30	95.4 %	79.4 %
MIOM-shift	73	10	100 %	79.6 %
	63	20	100 %	83.7 %
	53	30	100 %	81.1 %

Table 6.10: Accuracy results for general hyperplane splits and shifted hyperplane splits on the *C. glabrata* data set with $\mu = 0.001$

6.4 Identification of important mutations

Apart from constructing the best tree to classify future *C. albicans* and *C. glabrata* isolates, a second objective is to identify the mutations that are possibly associated with resistance. We use the MIO

formulation for constructing optimal classification trees to extract the most important features that can distinguish between resistant and susceptible sequences. Figure 6.9 shows an optimal binary classification tree of depth three, the node numbers represent positions on the ERG11 gene and each leaf contains the number of assigned susceptible and resistant isolates. The symbols at each split represent the nucleotides determining the branch directions for the data points, when the nucleotide is underscored it is a mutation. To determine the mutations we use a reference ERG11 sequence obtained from the *C. albicans* wild-type SC5314.

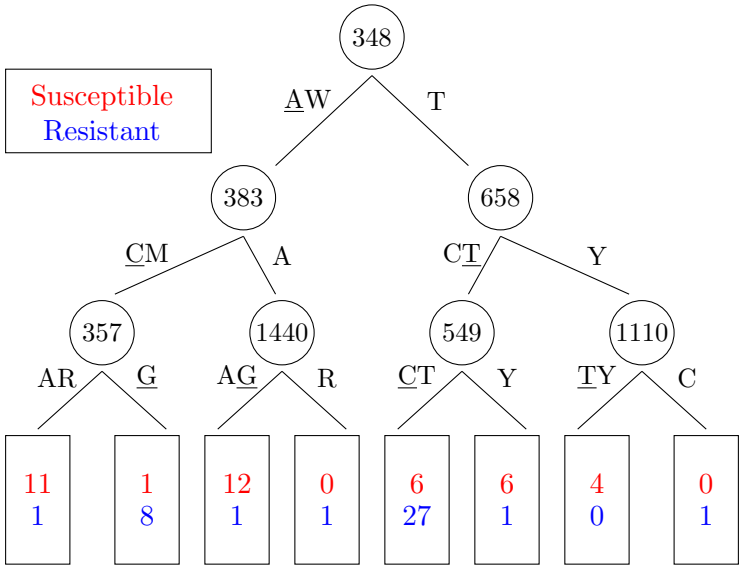


Figure 6.9: MIO depth two classification tree on *C. albicans* data set

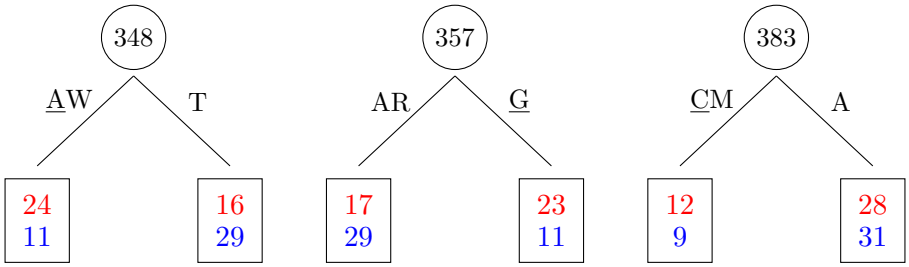


Figure 6.10: MIO splits using features 348, 357 and 383 on *C. albicans* data set

Leaves that contain relatively many resistant sequences are important for resistance related mutation identification. The second leaf, for example, contains eight resistant sequences and only one susceptible sequence. The features corresponding to this leaf can be extracted by traversing along the path from root to leaf, based on the model, simultaneous mutations at positions 348, 383 and 357 could potentially induce resistance. To examine the effect of simultaneous mutations on resistance we need to test the distinctive character of each feature individually, which is shown in Figure 6.10. As can be seen, individual splits poorly differentiate between resistant and susceptible sequences, which could be an indication for the occurrence of simultaneous mutations as resistance mechanism. Since there are multiple mechanisms that can cause resistance, it is not certain that these mutations are responsible, but the identified features can be used for further investigation.

An optimal tree of depth three for *C. glabrata* is depicted in Figure 6.11. Features that are possibly

related to resistance can be extracted by traversing the paths from root to leaves. Some of the nodes are highlighted to indicate which gene, FKS1 or FKS2, is considered at each split. Underscored nucleotides represent mutations and symbols without underscore represent non-mutated nucleotides. As shown in the figure, several combinations of mutations in one or both genes might be connected to resistance. Important combinations are: G1764A (FKS2), T1987C (FKS2) and G1764A (FKS2), T1885C (FKS1). Note that the fifth leaf in Figure 6.11 also contains relatively many resistant sequences, the decisions path to this leaf does not contain any mutation, which is remarkable. Since we are interested in finding resistant related mutations, we will not consider theses leaves.

The individual splits using features 1764 (FKS2), 1987 (FKS2) and 1885 (FKS1) are depicted in Figure 6.12. To examine the influence of simultaneous mutations we need to analyse the effect of individual mutations on resistance. As can be seen in Figure 6.12, individual splits at features 1987 (FKS2) and 1885 (FKS1) strongly distinguish between resistant and susceptible sequences, starting with a split at feature 1764 (FKS2) does not contribute to the distinctiveness. Therefore, the combinations we found are not necessarily related resistance.

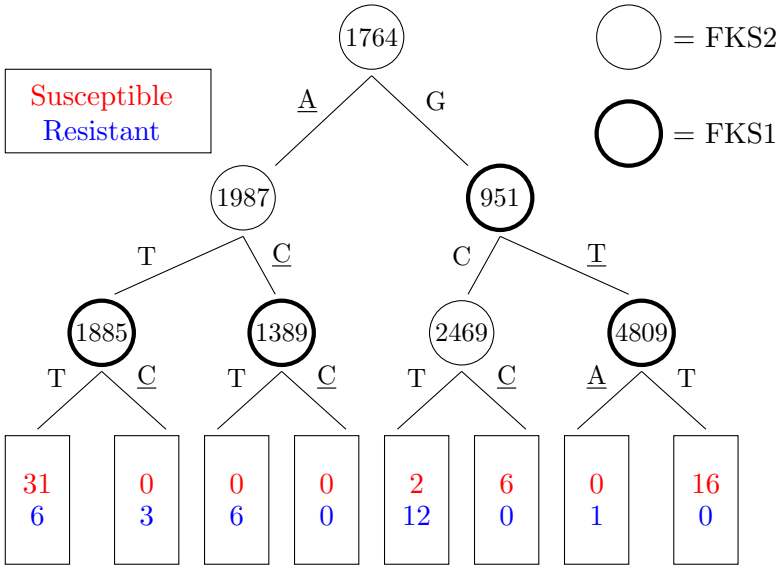


Figure 6.11: MIO depth two classification tree on *C. glabrata* data set

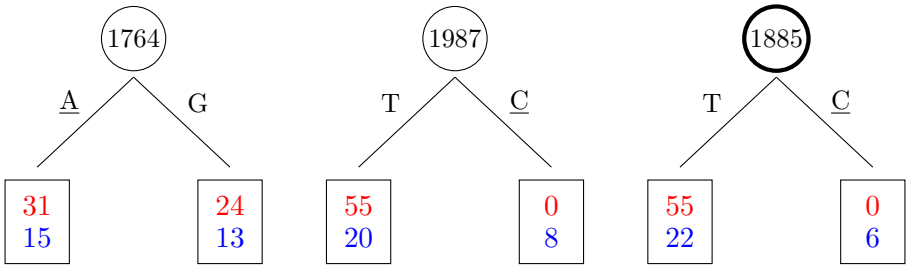


Figure 6.12: MIO splits using features 1764 (FKS2), 1987 (FKS2) and 1885 (FKS1) on *C. glabrata* data set

To study if there is an interdependent mechanism between mutations in FKS1 and FKS2 causing resistance, we construct several depth two classification trees on the complete *C. glabrata* data set.

We force the first split to use a position on FKS1, the second and third apply splits by using positions on FKS2. The distinctive character of each individual feature is examined by constructing the corresponding dept one classification tree.

The first leaf in tree (a) depicted in Figure 6.13 contains five resistant sequences and only one susceptible. The combination of mutations T2802C (FKS1) and G4320A (FKS2) correspond to this leaf. Individual mutations at positions 2802 (FKS1) and 4320 (FKS2) are only slightly differentiating between resistant and susceptible sequences as can be seen in the figure, which supports the theory of an interdependent mechanism between mutations in both genes. The second tree (b) contains two promising leaves with relatively many resistant sequences, the second and fourth, if we consider each split in the tree individually, it appears that simultaneous mutations do not contribute to resistance. For the remaining two trees (c) and (d) it is possible to identify important leaves, unfortunately, the corresponding decision paths contain non-mutations, which could point towards other resistance mechanisms, not related to mutations.

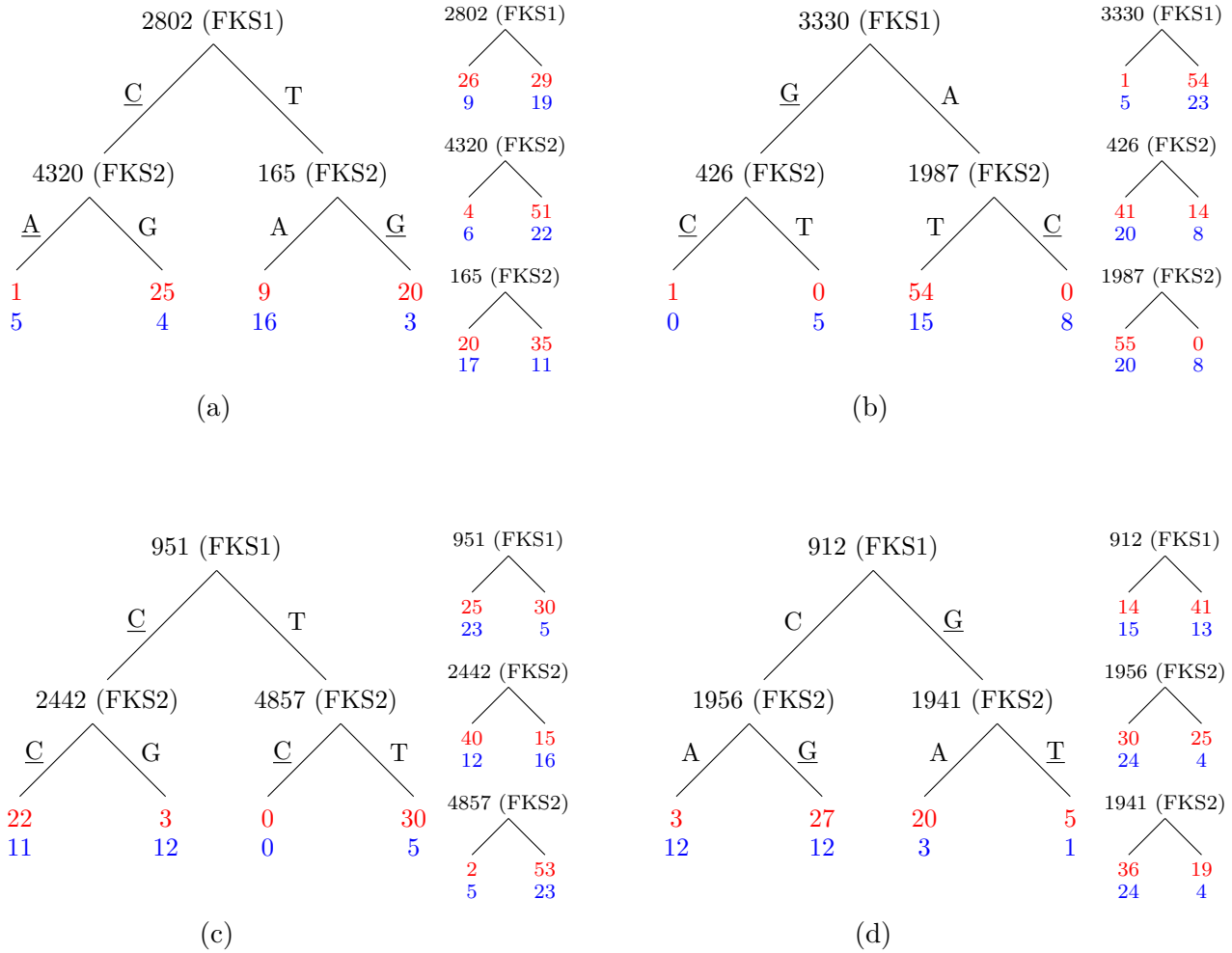


Figure 6.13: Four depth two classification trees to obtain combinations of mutations in FKS1 and FKS2 together with corresponding individual splits. The individual split results are important to verify the differentiating power of combinations of splits.

6.4.1 Resistance related amino acid substitutions

Mutations in ERG11 that cause amino acid substitutions and possibly changing the structure of the encoded protein, are important for resistance classification. In [35] there are 17 amino acid substitutions reported in the ERG11 gene of *C. albicans*, obtained from 33 isolates with reduced susceptibility to azole or full azole-susceptibility. They found the following substitution by clinically analyzing each isolate: A114S, D116E, K128T, Y132F, Y132H, K143R, Y257H, E266D, V437I, G448E and K143Q, Y205E, A255V, E260V, N435V, G472R, D502E. The first ten have been reported previously and, except for A114S and Y257H, are present in our data set. The second seven substitution are novel and absent in our data set.

Instead of clinical identification of amino acid substitution we construct MIO based optimal classification trees on the translated *C. albicans* data set, where nucleotides are translated to amino acids by using MEGA software. There are 20 proteinogenic amino acids encoded by triplets of nucleotides [13]. In the translation process, each triplet that contains one of the ambiguity symbols is translated by the symbol (?), since the ambiguity symbol represents two main nucleotides. In some cases, however, the ambiguity symbol does not bring ambiguity when it is translated as part from a triplet to amino acid. For example, the triplet GCY represents GCC or GCT, yet in both cases, the triplet is translated to the amino acid Alanine (A). To prevent extra ambiguity, we removed unnecessary (?) symbols and replaced them by the corresponding amino acid. Each sequence in the translated *C. albicans* data set exists of 514 amino acids, represented by a unique amino acid code according to the IUPAC system [7, 9]. Because columns in the data set contain at most four different symbols, we use quaternary trees, as depicted in Figure 6.14, to do the analysis. We retrieved the following 7 individual amino acid substitutions possibly related to resistance: Y132H, Y132F, K143E, K143R, S405F, V437I and G464S. Besides the effect of individual substitutions, classification trees enable to examine the effects

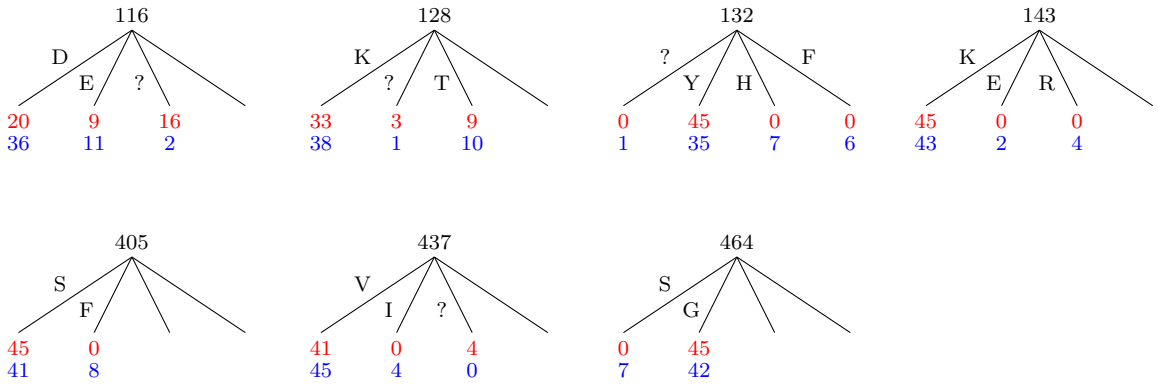


Figure 6.14: Single amino acid substitutions in ERG11

of simultaneous amino acid substitutions on resistance. By comparing paths in the ternary tree depicted in Figure 6.15 to single substitutions presented in Figure 6.14, we find that the combination of substitutions D116E and K128T could be an indication for resistance. Individual substitutions D116E and K128T differentiate poorly between susceptible and resistant sequences, however, as can be seen in Figure 6.15, simultaneous substitutions of D116E and K128T are more often present in resistant sequences.

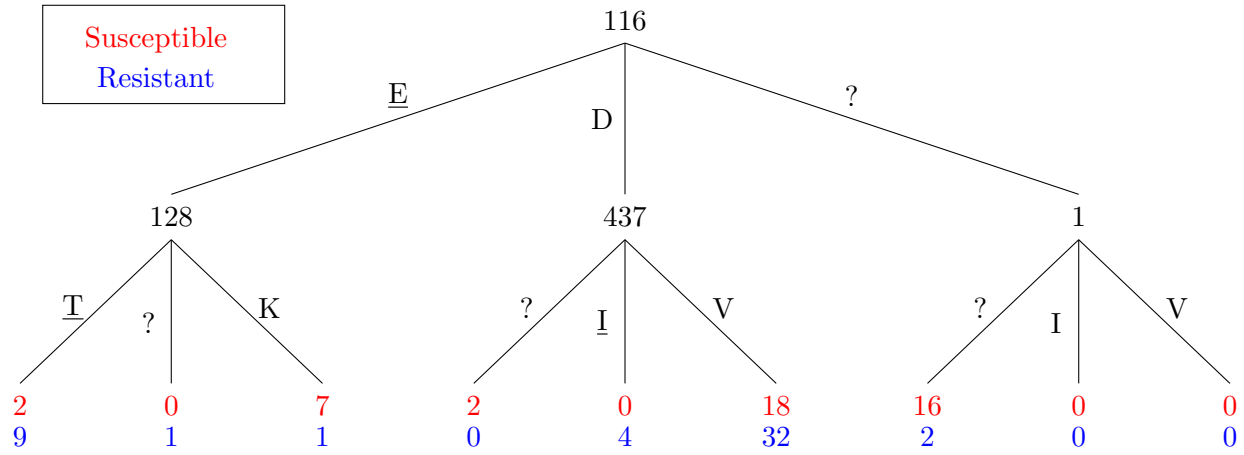


Figure 6.15: Depth two ternary tree for *C. albicans* data set on amino acid level

6.5 Ternary trees versus binary trees

In Chapter 5 we gave a formulation for constructing optimal ternary classification trees for the *C. albicans* data set. In some cases, ternary trees represent the data better than binary classification trees, especially when the data is ternary at each feature. Moreover, for ternary data sets, ternary classification tree will be easier to interpret compared to binary trees on the same data set.

In this section we compare ternary trees to binary classification trees on the *C. albicans* data set. Experimental results show that binary and ternary classification trees coincide for the *C. glabrata* data set, since only three features contain three different nucleotides in this data set. Table 6.11 shows the obtained in-sample and out-of-sample accuracy results for depth two MIO ternary, MIO binary and binary CART trees. Ternary trees perform at least as good as equally deep binary trees on in-sample data, a proof can be found at the end of this section.

In terms of out-of-sample performance, ternary MIO trees perform better than binary MIO trees for each train/test-distribution. Binary CART trees outperform ternary MIO trees for one distribution, for the other distributions, ternary trees perform stronger. These results indicate that ternary depth two trees might fit the data slightly better than binary trees. To further examine this we compare ternary depth two trees to binary CART trees of depths 2-4. As can be seen in Table 6.12, ternary depth two trees outperform binary CART trees of depth three and four in terms of out-sample-accuracy. Especially the results between (Ternary, D=2) and (CART, D=4) are interesting. The in-sample results show that binary depth four trees are more powerful than ternary depth two trees. Nevertheless, ternary depth two trees have better out-of-sample accuracy.

Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy	Average running time
Ternary MIO	70	10	83.6 %	72 %	12s
	60	20	84.7 %	68 %	10s
	50	30	84 %	71.7 %	5s
Binary MIO	70	10	81 %	71 %	27s
	60	20	83 %	66.5 %	15s
	50	30	81.6 %	69 %	10s
Binary CART	70	10	79 %	71 %	0s
	60	20	79.7 %	72 %	0s
	50	30	78.2 %	68.7 %	0s

Table 6.11: Depth two ternary and binary tree accuracy results

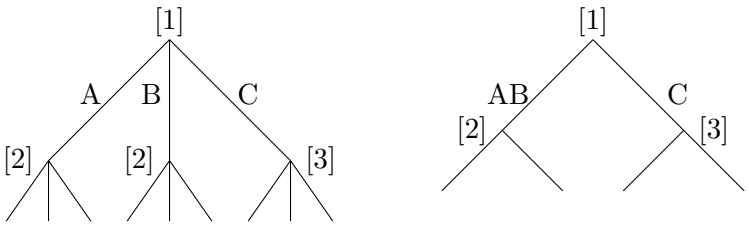
Method	Number of train sequences	Number of test sequences	In-sample accuracy	Out-of-sample accuracy
Ternary D=2	70	10	83.4 %	72 %
	60	20	83.7 %	70.8 %
	50	30	84.2 %	70.8 %
CART D=2	70	10	78.9 %	71.5 %
	60	20	78.3 %	71.3 %
	50	30	78.4 %	70.3 %
CART D=3	70	10	83.2 %	66 %
	60	20	83 %	67 %
	50	30	82.7 %	67.3 %
CART D=4	70	10	85.8 %	67.5 %
	60	20	85.8 %	67.3 %
	50	30	86.2 %	68.3 %

Table 6.12: Comparing ternary depth two tree to binary depth four tree

Theorem 1. *Ternary trees perform at least as good as equally deep binary trees in terms of in-sample missclassification error for binary class classification problems with ternary data.*

We will prove this by showing that for each binary tree of depth d we can always construct a ternary tree of depth d, by using the same features at each branch node, that has equal or lower misclassification error. Figure 6.16 shows the construction process of a ternary tree given a binary tree. The right-hand side of the figure shows a depth two binary tree and on the left-hand side the constructed ternary tree using the same features [1]-[3] for each split is depicted.

Suppose we are given a binary tree T_{Bin} and equivalent ternary tree T_{Ter}, constructed by splitting on the same features. Each leaf $t_{\text{Bin}} \in T_{\text{Bin}}$ is equivalent to the union of k leaves $\bigcup_{j=1}^{k} t_{\text{Ter}}^{j}$ for $k \geq 1$ and $t_{\text{Ter}}^{j} \in T_{\text{Ter}}$, $j = 1, \ldots, k$, where equivalent means that both contain the same data points. We will prove that each misclassification error in $\bigcup_{j=1}^{k} t_{\text{Ter}}^{j}$ will result in a misclassification error in the equivalent leaf t_{Bin}.



Let p^j_{Ter} be the number of data points in t^j_{Ter} labeled class p and q^j_{Ter} the points assigned class q. The misclassification error for $\bigcup_{j=1}^k t^j_{\text{Ter}}$ is given by:

$$\begin{aligned} L(\bigcup_{j=1}^{k} t^j_{\text{Ter}}) &= \sum_{j=1}^{k} L(t^j_{\text{Ter}}) \\ &= \sum_{j=1}^{k} \left(p^j_{\text{Ter}} + q^j_{\text{Ter}} - \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\}\right) \\ &= \sum_{j=1}^{k} \left(p^j_{\text{Ter}} + q^j_{\text{Ter}}\right) - \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\}, \end{aligned}$$

where we assume this error to be at least one. The misclassification error for t_{Bin} is given by:

$$\begin{aligned} L(t_{\text{Bin}}) &= \sum_{j=1}^{k} p^j_{\text{Ter}} + \sum_{j=1}^{k} q^j_{\text{Ter}} - \max\{\sum_{j=1}^{k} p^j_{\text{Ter}}, \sum_{j=1}^{k} q^j_{\text{Ter}}\} \\ &= \sum_{j=1}^{k} \left(p^j_{\text{Ter}} + q^j_{\text{Ter}}\right) - \max\{\sum_{j=1}^{k} p^j_{\text{Ter}}, \sum_{j=1}^{k} q^j_{\text{Ter}}\}. \end{aligned}$$

The aim is to prove $L(\bigcup_{j=1}^k t^j_{\text{Ter}}) \leq L(t_{\text{Bin}})$ which is equivalent to proving that:

$$\max\{\sum_{j=1}^{k} p^j_{\text{Ter}}, \sum_{j=1}^{k} q^j_{\text{Ter}}\} \leq \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\}.$$

Since

$$\sum_{j=1}^{k} p^j_{\text{Ter}} \leq \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\} \quad \text{and} \quad \sum_{j=1}^{k} q^j_{\text{Ter}} \leq \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\},$$

we have

$$\max\{\sum_{j=1}^{k} p^j_{\text{Ter}}, \sum_{j=1}^{k} q^j_{\text{Ter}}\} \leq \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\}$$

and hence

$$L(\bigcup_{j=1}^{k} t^j_{\text{Ter}}) = \sum_{j=1}^{k} \left(p^j_{\text{Ter}} + q^j_{\text{Ter}}\right) - \sum_{j=1}^{k} \max\{p^j_{\text{Ter}}, q^j_{\text{Ter}}\}$$

$$\leq \sum_{j=1}^{k} \left(p^j_{\text{Ter}} + q^j_{\text{Ter}}\right) - \max\{\sum_{j=1}^{k} p^j_{\text{Ter}}, \sum_{j=1}^{k} q^j_{\text{Ter}}\} = L(t_{\text{Bin}}).$$

To show that the opposite does not always hold, we give a counter example. Suppose we have six two-dimensional labeled data points as shown in Figure 6.17. Black circles represent one class and white circles the other. We can perfectly partition the data by using a ternary tree that splits at feature 1 first and uses feature 2 for the following three splits. By using this ternary tree, each data point will be placed in one of the leaves solely, resulting in a misclassification error equal to zero.

If we apply binary trees, using the same split features, there are three situations possible after applying the first split: 1) points with A and B at feature 1 are partitioned together, 2) points with A and C at feature 1 are partitioned together 3) points with B and C at feature 1 are partitioned together. In all cases, as can be seen in the figure, the partitioning is problematic at the second level of the tree. For each situation, one of the partitions contains conflicting data points, i.e. data points containing the same letter at feature 2 but labeled a different class. It is not possible to differentiate between these points resulting in a non-zero misclassification error. Changing the order of features applied at each split will not improve the misclassification error. Which proves the theorem.

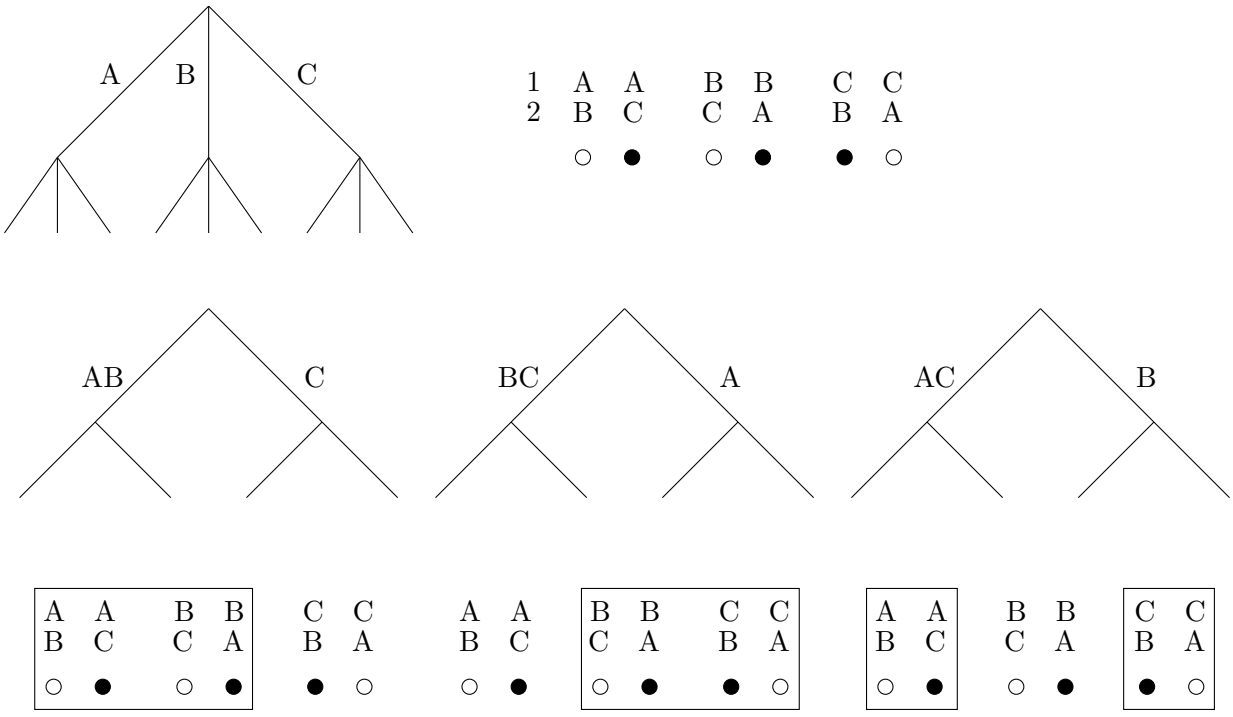


Figure 6.17: Ternary and binary tree split on the same feature.

6.6 Conclusion

The objective of this thesis was to examine the use of classification tree models on the *C. albicans* and *C. glabrata* data sets for classification and resistance related mutation/substitution identification. We focused on the MIO formulation for constructing univariate (4.23) and multivariate (4.27) optimal classification trees as presented by Bertsimas and Dunn in [2]. We added several extensions to this formulation to construct tree models that take into account false negative errors and we modelled non-binary classification trees. As benchmarks for the MIO based classification tree models we used state-of-the-art CART and LSVM algorithms.

We found that CART, in most cases, outperforms the MIO approach for univariate binary classification trees in terms of out-of-sample accuracy, which is remarkable since MIO provides optimal trees and, as a result, performs better on in-sample data. It seems more intuitive that an equally deep tree with a better fit on training data performs stronger on test data as well. This is not the case for *C. albicans* and *C. glabrata*. Perhaps, optimal trees tend to overfit the training data in data-scenarios where there is little data available. It would be interesting to further analyse these results.

By altering the objective function (4.32) and adding linearizing constraints (4.29)-(4.31) to the original MIO formulation (4.23), we were able to take into account false negative errors in the classification process, which was not done before. When α, the false negative penalty parameter, is sufficiently large, the fraction of false negative errors decreased 64-80% for *C. albicans* and 56-66% for *C. glabrata*. This serious decrease in the number of false negative misclassifications can have a significant impact in medical resistance classification or other classification problems where one type of error is preferred over the other. The effect of a penalty factor in the objective function on out-of-sample accuracy is little. For *C. albicans*, the out-of-sample accuracy decreased at most 14%, for *C. glabrata*, the mean out-of-sample accuracy decreased at most 8%. In some cases, the out-of-sample accuracy increased slightly for both data sets.

Multivariate trees turned out to be very powerful in terms of in-sample accuracy. For both data sets it was possible to separate most of the resistant isolates from the susceptible ones by applying a single split, which could be attributed to the high dimensionality of the data. As the number of features in the data increases, the probability of finding a hyperplane that perfectly partitions the data also increases. The multivariate in-sample accuracy results correspond to the results obtained by using LSVM, which gave 100% in-sample accuracy for *C. glabrata* and 98.9-99.6% for *C. albicans*. On out-of-sample data, multivariate trees performed poorly for the *C. albicans* data set, only 63.5-64.8% got assigned the correct class. For the *C. glabrata* data set, however, multivariate trees provided a good model. On average 76.5-82.5% of *C. glabrata* isolates got assigned the correct class using multivariate trees compared to 65.5-73.3% using univariate depth two trees. The best results on out-of-sample and in-sample accuracy for *C. glabrata* were obtained by using a LSVM algorithm. On average, 86-89% of training sequences got assigned the correct class by the this model.

For the *C. albicans* data set we constructed ternary classification trees and compared the obtained trees to binary trees of depths 2-4. Running times for MIO constructed ternary trees turned out to be slightly better than running times for MIO based binary trees, which might be due to a reduction in

the number of features for this method. Promising were the ternary depth two tree results compared to binary depth three and four trees. Ternary trees are more powerful on out-of-sample data, while, binary depth three and four trees perform better in terms of in-sample accuracy. Compared to binary depth two trees, ternary depth two trees had similar out-of-sample accuracy results. On average approximately 68-72% of *C. albicans* test sequences got assigned the correct class using ternary or binary depth two trees.

Finally, the second objective was to identify individual and combinations of mutations/substitutions that could be an indication for resistance in *C. albicans* and *C. glabrata*. The individual amino acid substitutions in *C. albicans* that we found were reported before in [35]. An interesting combination of substitutions was discovered between D116E and K128T, that was present in nine resistant *C. albicans* isolates and only two susceptible isolates. Individual substitutions D116E and K128T poorly differentiated between susceptible and resistant isolates, indicating a possible interdependent resistance mechanism between both substitutions. For *C. glabrata* we examined the presence of interdepent mutations in FKS1 and FKS2 that could induce resistance in this species. We found a resistant related combination T2802C (FKS1) and G4320A (FKS2) that could indicate the presence of a resistance inducing mechanism of simultaneous mutation in FKS1 and FKS1. However, in our analysis, the effect of individual mutations on resistance was generally stronger than the effect of combinations.

Concluding, the use of univariate binary and non-binary classification trees to classify *C. albicans* yeast isolates as being resistant or susceptible is very promising. Despite the small number of training data, we were able to construct tree models of reasonable quality. Adding more data to the model will further increase the predictive power in the future. For *C. glabrata*, the use of LSVM as a predictive model for drug resistance is recommended. Both out-of-sample and in-sample accuracy results are outstanding and the implementation of this algorithm is straightforward. A disadvantage of LSVM models is that they are relatively difficult to interpret compared to univariate classification trees.

Using classification trees to identify mutations/substitutions in *C. albicans* and *C. glabrata* responsible for drug resistance or any other phenotypic trait was effective. Especially when combinations of mutations possibly play a crucial role in the expression of a phenotypic trait, univariate classification trees offer opportunities, as they are easy to interpret. We were able to identify some important combinations present in relatively many resistant *C. albicans* and *C. glabrata* isolates. Further investigation should be done to possibly find more resistant related combinations.

Chapter 7
Discussion

In this section we will reflect on some of the difficulties we encountered during the experimental phases as part of this thesis. We will also discuss several applications of our findings and state possibilities and methods for further investigation.

To be able to find good models that represent the data and predict with high accuracy it is important to have enough data. We constructed classification models for *C. albicans* and *C. glabrata* using small labeled data sets. Experiments showed percentage fluctuations in terms of out-of-sample and in-sample accuracy for different generated training and test sets. The quality of the model turned out to be highly dependent on the distribution of test and training data, which impeded accurate comparative method analysis. Especially in the field of medicine, an increase or decrease in the predictive power of a model can have a significant impact. Therefore, it would be of great value if there are more sequences available to make the model more robust.

Another difficulty we encountered in this thesis was caused by limited computing power. We could not run the algorithm for depth four or deeper binary and non-binary trees, due to time limitations. Ternary depth two trees showed good accuracy results on the *C. albicans* data set, it would be interesting to see if ternary depth three trees perform even better. There are several ways to improve the running time of solving the MIO model for classification trees. By implementing these methods in the future, it might be possible to analyse the power of deeper ternary trees on the *C. albicans* data set and other data sets.

In many comparative analyses between equally deep MIO optimal classification trees and CART, the performance of CART in terms of out-of-sample accuracy was stronger than the performance of MIO obtained trees. This seems counterintuitive, since MIO trees are optimal on training data, we would expect them to outperform non-optimal trees on test data as well. It would be interesting to further analyse this and find possible reasons explaining this phenomenon.

In Section 6.4 we used classification trees to identify resistant related mutations and amino acid substitutions in *C. albicans* and *C. glabrata*. We were especially interested in mechanisms of simultaneous mutations that could indicate resistance. In multiple cases for the *C. glabrata* data set we experienced how strong individual mutations overpowered the effect of combinations of mutations on resistance. In a future study, powerful features could be removed from the data to see if there are simultaneous mutations that occur more often in resistant sequences than in susceptible ones. In some cases we found promising leaves containing relatively many resistant sequences without the occurrence of mutations on the corresponding decision paths. This could indicate the presence of some other resistance inducing mechanism that needs further investigation.

Finnaly, the presence of uncertainty in the data gave rise to an interesting optimization problem of finding a completion of the data that generates a tree with smallest misclassification error. In the case of genome data, we wish to find a strategy to optimally replace ambiguity symbols by one of the main nucleotides to obtain an optimal performing model.

Chapter 8
Appendix

8.1 Tables

Variables		
Name	**Type**	**Function**
a_t	Binary	Split variable determining which feature to split on
b_t	Continuous	Split variable determining which branch direction to take
d_t	Binary	Indicator function to track which nodes apply splits
z_{it}	Binary	Indicator function to track the points assigned to each leaf
l_t	Binary	Indicator function to track if a leaf contains a point
Y_{ik}	$\{-1, 1\}$-Matrix	Label matrix
N_{kt}	Integer	Number of points of label k in node t
N_t	Integer	Total number of points in node t
c_t	$\{1, \dots K\}$	Assigned label to each leaf given by the majority of the labels of all data points in that leaf
c_{kt}	Binary	Indicator function to track the prediction at each leaf
L_t	Integer	Misclassification loss in node t given by the total number of points in the node minus the number of points assigned the most common label

Table 8.1: Variables from the MIO formulation

Constraints	
Constraint	**Description**
$a_t x_i < b_t$ and $a_t x_i \geq b_t$	Split applied at branch node
$\sum_{j=1}^{p} a_{jt} = d_t$	Enforces splits to only involve a single variable
$0 \leq b_t \leq d_t$	If branch node does not apply split d_t, b_t and a_t are set 0
$d_t \leq d_{p(t)}$	Restrict branch node from applying a split if its parent does not also apply a split.
$z_{it} \leq l_t$	To keep track of the points assigned to each leaf
$\sum_{i=1}^{n} z_{it} \geq l_t N_{\min}$	To enforce a minimum number of points at each leaf
$\sum_{t \in \mathcal{T}_L} z_{it} = 1$	Forces each point to be assigned to exactly one leaf
$a_m^T x_i + \epsilon < b_m + (1+\epsilon)(1 - z_{it})$ $a_m^T x_i \geq b_m - (1 - z_{it})$	To enforce the splits that are required for path from root to assigned leaf t
$Y_{ik} = \begin{cases} +1, & \text{if } y_i = k \\ -1, & \text{otherwise,} \end{cases}$	Label matrix
$N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}$	Number of points of label k in node t
$N_t = \sum_{i=1}^{n} z_{it}$	Total number of points in node t
$c_t = \text{argmax}_{K=1,\ldots,K} \{N_{kt}\}$	Optimal label prediction
$\sum_{k=1}^{K} c_{kt} = l_t$	Forces a single class prediction at each node
$L_t = N_t - \max_{K=1,\ldots,K} \{N_{kt}\}$	Misclassification error per leaf t
$L_t \geq N_t - N_{kt} - M(1 - c_{kt})$ $L_t \leq N_t - N_{kt} + n c_{kt}$ $L_t \geq 0$	Linearization of the above described misclassification error.
$\min \sum_{t \in \mathcal{T}_L} L_t$	The objective is to minimize the total misclassification error.

Table 8.2: Constraints from the MIO formulation

Bibliography

[1] Beerenwinkel, N., Schmidt, B., Walter, H., Kaiser, R., Lengauer, T., Hoffmann, D., Korn, K. and Selbig, J. (2002). Diversity and complexity of HIV-1 drug resistance: a bioinformatics approach to predicting phenotype from genotype. *Proceedings of the National Academy of Sciences*, 99(12), 8271-8276.

[2] Bertsimas, D. and Dunn, J. (2017). Optimal classification trees. *Machine Learning*, 106(7), 1039-1082.

[3] Bongomin, F., Gago, S., Oladele, R. O. and Denning, D. W. (2017). Global and Multi-National Prevalence of Fungal Diseases-Estimate Precision. *Journal of Fungi*, 3(4), 57-86.

[4] Breiman, L., Friedman, J., Stone, C.J. and Olshen, R.A. (1984). *Classification and Regression Trees.* Chapman and Hall.

[5] Celebi, M. E. and Aydin, K. (Eds.). (2016). *Unsupervised Learning Algorithms.* Springer, Cham.

[6] Chen, X., Wang, M., and Zhang, H. (2011). The use of classification trees for bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1), 55-63.

[7] Comm, I. I. (1970). Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20), 4022-4027.

[8] Conforti, M., Cornuejols, G., Zambelli, G. (2014). *Integer Programming* New York, NY: Springer Publishing Company.

[9] Cornish-Bowden, A. (1985). Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic acids research*, 13(9), 3021.

[10] Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[11] Feijen, W. (2018). *Optimal Classification Trees* (Master Thesis) VU, School of Business and Economics.

[12] Firat, M., Crognier, G., Gabor, A.F., Zhang, Y., Hurkens, C.A.J. (2018). *Constructing classification trees using column generation.* arXiv:1810.06684 [cs.LG]

[13] Fromm, H.J. and Hargrove, M. (2012). *Essentials of Biochemistry* Springer, Cham.

[14] Higler, S., Schmidt, J., Schneider, S., Shingjergji, K. and Stallman, M. (2018). *Following Bertsimas: Machine Learning through a Modern Optimization Lens* (Master project). University of Maastricht.

[15] Hyafil, L. and Rivest, R.L. (!976). Constriction optimal binary decision trees is NP-complete. *Information processing letter*, 5(1), 15-17.

[16] John Lu, Z. Q. (2010). The elements of statistical learning: data mining, inference, and prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 173(3), 693-694.

[17] Kadlec, K., Wendlandt, S., Feler, A. T., and Schwarz, S. (2015). Methods for the detection of antimicrobial resistance and the characterization of Staphylococcus aureus isolates from food-producing animals and food of animal origin. In *Antimicrobial Resistance and Food Safety* (207-232). San Diego, CA: Elsevier.

[18] Keprta, S. (1996). Non-binary classification trees. *Statistics and Computing*, 6(3), 231-243.

[19] Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4), 261-283.

[20] Larranaga, P., Calvo, B., Santana, R., Bielza, C., Galdiano, J., Inza, I., Lozano, J., Armaanzas, R., Santaf, G., Prez, A. and Robles, V. (2006). Machine learning in bioinformatics. *Briefings in bioinformatics*, 7(1), 86-112.

[21] Mehryar M., Afshin R., Ameet T. (2012). *Foundations of Machine Learning.* Cambridge, MA: The MIT Press.

[22] Murray, R. J., Lewis, F. I., Miller, M. D., and Brown, A. J. L. (2008). Genetic basis of variation in tenofovir drug susceptibility in HIV-1. *Aids*, 22(10), 1113-1123.

[23] Ohta, S., Kurebayashi, R. and Kobayashi, K. (2008). Minimizing False Positives of a Decision Tree Classifier for Intrusion Detection on the Internet. *Journal of Network and Systems Management.*, 16(4), 399-419.

[24] Pham, C. D., Iqbal, N., Bolden, C. B., Kuykendall, R. J., Harrison, L. H., Farley, M. M., Lockhart, S. R. (2014). Role of FKS Mutations in Candida glabrata: MIC Values, Echinocandin Resistance, and Multidrug Resistance. *Antimicrobial Agents and Chemotherapy*, 58(8), 4690 - 4696.

[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... and Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), 2825-2830.

[26] Quinlan, J. R. (1993). C4.5: Programs for Machine Learning. *Machine Learning*, 16(3), 235-240.

[27] Robert, C. (2014). *Machine learning, a probabilistic perspective.* Cambridge, MA: The MIT Press.

[28] Rodrigues, C. F., Rodrigues, M. E., Silva, S., and Henriques, M. (2017). Candida glabrata Biofilms: How Far Have We Come?. *Journal of fungi.* 3(1), 11-41.

[29] Rutgers Biomedical and Health Sciences. (2013, December 23). Attacking fungal infection, one of world's major killers. *Science Daily.* Retrieved January 2, 2019 from www.sciencedaily.com/releases/2013/12/131223181303.htm

[30] Sanglard D. (2017). Mechanisms of Drug Resistance in Candida albicans. In *Candida albicans: Cellular and Molecular Biology* (287-311). Springer, Cham.

[31] Sasso, M., Roger, C., and Lachaud, L. (2017). Rapid emergence of FKS mutations in Candida glabrata isolates in a peritoneal candidiasis. *Medical Mycology Case Reports*, 16, 2830.

[32] Suykens, Johan A. K., et al. (2002). Support Vector Machine. In *Least Squares Support Vector Machines* (29-38). World Scientific Publishing Co Pte Ltd, 2002. ProQuest Ebook Central.

[33] Theodoridis, S. (2015). LEARNING IN PARAMETRICMODELING: BASIC CONCEPTSAND DIRECTIONS. In Machine learning: a Bayesian and optimization perspective. Academic Press.

[34] Uppuluri P., Khan A., Edwards J.E. (2017). Current Trends in Candidiasis. In *Candida albicans: Cellular and Molecular Biology* (5-23). Springer, Cham.

[35] Xiang, M. J., Liu, J. Y., Ni, P. H., Wang, S., Shi, C., Wei, B., Ni, Y.X. and Ge, H. L. (2013). Erg11 mutations associated with azole resistance in clinical isolates of Candida albicans. *FEMS yeast research*, 13(4), 386-393.

[36] Yapar, N. (2014). Epidemiology and risk factors for invasive candidiasis. *Therapeutics and clinical risk management*, 10, 95-105.