CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Automatic intron detection in fungal genomes using machine learning

Bc. Denis Baručić
Open Informatics, Data Science

May 2019
Supervisor: doc. Ing. Jiří Kléma, Ph.D

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Baručić** Jméno: **Denis** Osobní číslo: **434816**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Datové vědy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Automatická detekce intronů v genomech hub pomocí metod strojového učení

Název diplomové práce anglicky:

Automatic intron detection in fungal genomes using machine learning

Pokyny pro vypracování:

1. Seznamte se se strukturou DNA eukaryot.
2. Implementujte parser gff a fasta souborů z Joint Genome Institute dodaných vedoucím práce pro extrakci DNA segmentů.
3. Introny a exony extrahujte, reportujte základní statistiky, srovnejte introny a exony i jednotlivé genomy a jejich třídy mezi sebou.
4. Vypracujte rešerši stávajících nepravděpodobnostních algoritmů strojového učení pro automatickou segmentaci genomu.
5. Vybraný algoritmus implementujte v úpravě vhodné pro detekci intronů hub.
6. Nad daty dodanými vedoucím práce vytvořte modely pro jednotlivé genomy, pokuste se modely zobecnit pro příbuzné čeledě, rody, apod.
7. Vyhodnoťte dosažené výsledky.

Seznam doporučené literatury:

Sonnenburg, Sören, et al. 'Accurate splice site prediction using support vector machines.' BMC bioinformatics. Vol. 8. No. 10. BioMed Central, 2007.
Sonnenburg, Sören, Gunnar Rätsch, and Bernhard Schölkopf. 'Large scale genomic sequence SVM classifiers.' Proceedings of the 22nd international conference on Machine learning. ACM, 2005.
Huang, J., et al. 'An approach of encoding for prediction of splice sites using SVM.' Biochimie 88.7 (2006): 923-929.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Kléma, Ph.D., Intelligent Data Analysis FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **23.01.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **20.09.2020**

doc. Ing. Jiří Kléma, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání Podpis studenta

Acknowledgement / Declaration

I would like to express my sincere gratitude to my supervisor doc. Ing. Jiří Kléma, Ph.D. for giving me the opportunity to work on this project, for his valuable guidance, and for always taking his time to discuss my thesis and other topics.

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 24. 5. 2019

.....

Abstrakt / Abstract

Tato práce se zabývá problémem detekce intronů v DNA hub. Správná detekce intronů je stěžejní pro rozpoznání druhu na základě neznámé DNA sekvence. Introny však nelze identifikovat pomocí sady jednoduchých pravidel. Z toho důvodu jsme navrhli řešení, které využívá tři modely strojového učení. Dva z těchto modelů se snaží detekovat začátky, respektive konce intronů. Introny pak vznikají párováním detekovaných začátků s konci a jsou dále filtrovány pomocí třetího modelu, jehož účelem je zpřesnění výsledků.

Pracovali jsme s daty 940 hub patřících do různých taxonů. Parametry modelů ale byly určeny empiricky na základě testování na pouze jednom druhu, což ponechává prostor pro zlepšení, neboť lze očekávat, že takové parametry nebudou optimální pro ostatní druhy.

Přestože byly použity suboptimální parametry, pomocí navrženého řešení lze najít až 90 % všech intronů konkrétních druhů.

Klíčová slova: detekce intronů; genom hub; support vector machine (metoda podpůrných vektorů)

Překlad titulu: Automatická detekce intronů v genomech hub pomocí metod strojového učení

This thesis tackles the problem of intron detection in fungal DNA. An accurate intron detection is crucial when recognizing a species from a given unfamiliar DNA sequence. However, the introns cannot be identified using a set of simple rules. Therefore, we designed a solution consisting of three machine learning models. Two of them attempt to detect starts and ends of the intron sequences, respectively. The introns are then composed by pairing the detected starts and ends and further filtered by the third model, which is supposed to refine the results.

We worked with data of 940 different fungi belonging to many taxons. However, parameters of the models were determined empirically using a grid search only on a single species, which leaves a space for future improvement as one can expect the parameters to be suboptimal for other species.

Despite using suboptimal parameters, the designed solution is able to detect up to more than 90% of all introns of certain species.

Keywords: intron detection; fungal genome; support vector machine

Contents /

Tables / Figures

Listings /

Chapter 1
Introduction

A genome of a eukaryotic organism contains parts called genes which consist of subsequences of two types: introns and exons. Both the introns and exons are difficult to determine given a sample of DNA due to multiple reasons, one of which may be their relatively small size when compared to the whole genome. Another reason is that the subsequences respect almost no rules, especially the exons. Some regularities hold among the majority of introns; however, the regularities are not very distinctive too.

The goal of this thesis is to develop a solution able to detect introns within fungal DNA. Since the intron detection is so complex, it is very challenging (perhaps even impossible) to come up with a traditional algorithm that would be capable of performing the task. We are also provided with a database of already known, annotated sequences available. Altogether it seems like a suitable problem for machine learning.

We thus decomposed the problem into a series of three classification tasks:

1. classification of intron starts,
2. classification of intron ends,
3. classification of whole introns given their starts and ends according to the previous tasks,

each of which is performed by a trained support vector machine.

Using the presented approach, we were able to detect up to more than 90% of all introns in individual organisms.

1.1 Motivation

The motivation for the thesis arises in the field of environmental microbiology which is a field of science that deals with microorganisms present in environments such as soils, forest litter, and others.

Environmental microbiologists have a wide range of interests, and one of them is to know what fungi occur in the soil at given locations. An approach that is used for this purpose is based on DNA comparison and could be summarized as follows:

1. the biologists gather soil samples in a forest,
2. then, DNA sequences present in the samples are extracted,
3. the sequences are compared with a database of DNA of known fungi, if there is an approximate match, they know what fungus of taxon the sequence belonged to.

This approach, however, does not perform very well. The main problem lies in the fact that introns bring variability into the DNA of organisms (i.e., the introns of two related fungi could differ significantly). Therefore, even though the DNA database contains fungi that are closely related to the fungi present within the gathered samples of soil, there are very few matches. Another complication is that although the total number of all fungi is expected to be up to 1,800,000, only around 90,000 are currently known and described, and just a small fraction of that is sequenced.

The proposed solution is to determine the fungi based on homology that is preserved in the exome of relatives. This approach, of course, requires to extract exons from the fungal genomes. Since introns are generally easier to identify than exons, we chose to identify the introns in the first place and to utilize the fact that any intron is always located between two exons which can thus be extracted afterward.

1.2 Text structure

After this introduction, we briefly introduce terminology and concepts from genetics and biology that we use throughout the thesis. We also present the theoretical foundations that are behind the machine learning methods we used to implement the solution, and several metrics to evaluate trained models.

The next chapter contains related work. We mention a broad spectrum of different approaches not only from a field of machine learning.

In the fourth chapter, we describe the provided data and also give some statistical properties of the data which reveal relevant findings we will employ in our implementation.

Based on the findings presented in the previous chapter, we design our solution in Chapter 5. The solution consists of multiple parts each of which we touch and outline in the chapter.

In Chapter 6, we describe the software we used to handle the provided data.

Chapters 7 and 8 deal with an implementation of the splice site classification and the intron classification. It includes an overview of the parameter selection and discussion on the model ability to generalize.

Chapter 9 shows results of the whole classification process, which was designed in Chapter 5, when applied on multiple fungi.

Finally, Chapter 10 summarizes the thesis and proposes ideas for future work.

Chapter 2
Theoretical background

In this chapter, we introduce several concepts coming from biology or machine learning that we use in the thesis. The biological part pays the main attention to regularities that may help to detect introns. The machine learning part focuses on support vector machines that will be our primary detection method. As we deal with biological sequences, we will use kernel functions as an effective counterpart to explicit feature extraction.

2.1 DNA basics

The following paragraphs regarding the basics of DNA are based on [1–3].

Every living organism has DNA within its cells. DNA, an acronym for *deoxyribonucleic acid*, denotes a complex molecule which, among other purposes, contains all the information necessary to build and maintain an organism.

DNA is composed of a series of smaller molecules called *nucleotides*. Every nucleotide contains a *nitrogenous base*, a *deoxyribose sugar molecule*, and a *phosphate group*. There are four different DNA nucleotides named after the nitrogenous bases they contain:

- adenine (A),
- thymine (T),
- guanine (G),
- and cytosine (C).

The deoxyribose molecule contains five carbon atoms arranged in the shape of a ring. We refer to each of these atoms by a number followed by the prime symbol. The area surrounding the $5'$ carbon is known as the $5'$ end of the nucleotide, and at the opposite side of the deoxyribose ring, there is the $3'$ end - the area around the $3'$ carbon. When the nucleotides are bonded together, the $5'$ end of a single nucleotide attaches to the $3'$ end of the adjacent nucleotide. Consequently, a group of bonded nucleotides forms a strand.

A strand of DNA can bond with another strand of DNA resulting in a double-stranded DNA. The bond is base-to-base: bases from one strand are bonded with complementary bases from the other strand — see Figure 2.1. The complementary bases are given according to Table 2.1.

nucleotide base	complementary base
A	T
T	A
G	C
C	G

Table 2.1. Pairs of nucleotides that form bonds between DNA strands.

The two strands of a double-stranded DNA run *anti-parallel* to each other. One runs in $3' \to 5'$ direction while the other in the opposite ($5' \to 3'$) direction.

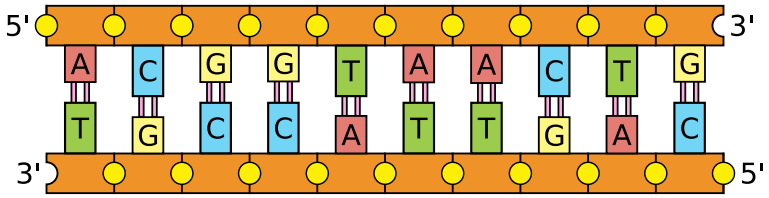

Figure 2.1. An illustration of double-stranded DNA. Based on [4].

2.2 Genes and protein synthesis

DNA includes, among others, sections called *genes*. *A gene is the basic physical and functional unit of heredity. ... Some genes act as instructions to make molecules called proteins. However, many genes do not code for proteins* [5].

In general, there are two categories of genes: eukaryotic and prokaryotic [6]. Because fungi are eukaryotic organisms [7], we will consider only eukaryotic genes in the following lines. Be aware that there are differences between eukaryotic and prokaryotic genes.

There are three major steps of protein synthesis for eukaryotic genes (see Figure 2.2):

- transcription,
- post-processing,
- and translation [8].

During transcription, a particular enzyme called *RNA polymerase* breaks the bonds in a double-stranded DNA and attaches to one of the strands. The enzyme then begins moving down the strand in the $3' \rightarrow 5'$ direction, and as it does so, it pairs together complementary bases creating a new strand of *precursor messenger ribonucleic acid* (pre-mRNA) that is organized in the $5' \rightarrow 3'$ direction. RNA is a molecule composed of a series of the same nucleotides as in DNA except thymine (T) is replaced with uracil (U) [2].

In the post-processing step, the pre-mRNA molecule is processed to form mature mRNA [2]. A necessary step called *splicing* takes part during the process. The step involves removal of certain (noncoding) sequences referred to as intervening sequences, or *introns*. The remaining sequences of a gene, called *exons*, get connected and together they form the final mRNA [9]. The process of splicing is especially relevant for the thesis, so it is further described in the next section.

The information contained in the final, mature mRNA is used to direct the creation of a new protein molecule during the translation step. The site where the translation begins and ends is marked by a start and stop codon[1], respectively. Start codon is usually ATG, and stop codon one of TAG, TAA, or TGA [10].

2.2.1 RNA splicing

A gene is a series of alternating introns and exons, whereas every gene must start and end with exons. As mentioned before, the introns are removed during the splicing process [11].

Splicing requires introns to contain a branch point, and two different splice sites (boundaries between the intron and its adjacent exons):

- the exon-intron boundary, known as the donor site ($5'$ end of the intron),
- and the intron-exon boundary, called the acceptor site ($3'$ end of the intron) [12].

[1] A *codon* is a nucleotide triplet that is a part of the genetic code and codes for a specific amino acid.

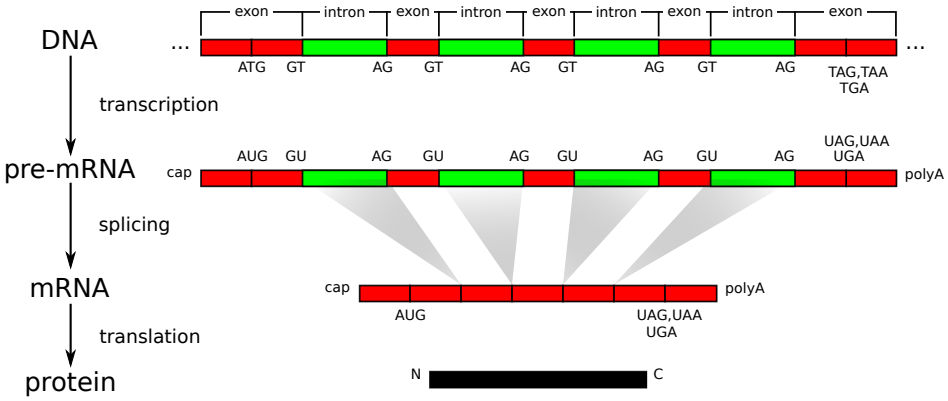

Figure 2.2. The process of protein synthesis. The image is taken (and redrawn) from [8].

The branch point is a single nucleotide (typically adenine) which is involved in the splicing.

The vast majority of introns has the dimer GT as the donor site and AG as the acceptor site (this fact is known as `GT-AG` rule [6]). Therefore, the dimers GT and AG can be used to identify potential donor and acceptor sites, respectively. The `GT-AG` pair, however, occurs very frequently making it not sufficient to solely search for the `GT-AG` pairs when trying to identify the true splice sites and, consequently, an intron.

Another regularity is that introns often contain a polypyrimidine tract, which is a region with frequent occurrence of `T`s. This region is usually close to the acceptor site [12].

2.3 Support Vector Machines

Support Vector Machine (SVM) is a supervised learning model which belongs to the family of linear classifiers and currently is one of the most effective classification algorithms. The following description is based on [13, Chapter 4] and [14].

2.3.1 Linear classification

For an input space $\mathcal{X} \subseteq \mathbb{R}^D, D \geq 1$ and a target space $\mathcal{Y} = \{-1, +1\}$, a linear classifier learns a hypothesis (or a prediction rule) $f\colon \mathcal{X} \to \mathcal{Y}$ in the form of

$$f(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{x}, \mathbf{w} \rangle + b\right),$$

where $\mathbf{x} \in \mathcal{X}$ is an input sample, $\mathbf{w} \in \mathbb{R}^D$ is a weight vector, and $b \in \mathbb{R}$ is a bias. A linear classifier thus labels all points falling on one side of the hyperplane $\langle \mathbf{x}, \mathbf{w} \rangle + b = 0$ positively, and the rest negatively.

Given a training sample $S_m = \{(\mathbf{x}^i, y^i) \mid 1 \leq i \leq m\} \subseteq (\mathcal{X} \times \mathcal{Y})^m$, SVMs find the *maximum-margin hyperplane* which is the hyperplane with maximal distance (or margin) to the closest examples. As a consequence, the hyperplane separates the classes represented by their examples.

2.3.2 Primal optimization problem

Had we dealt with linearly separable data, we would always find a hyperplane with parameters $\mathbf{w}, b$ that does not pass through any sample point such that

$$\min_{(\mathbf{x},y) \in S_m} |\langle \mathbf{x}, \mathbf{w} \rangle + b| = 1.$$

Then for linearly separable data, it holds

$$|\langle \mathbf{x}^i, \mathbf{w} \rangle + b| \geq 1, \quad i = 1, \ldots, m,$$

and hence

$$y^i \left(\langle \mathbf{x}^i, \mathbf{w} \rangle + b \right) \geq 1, \quad i = 1, \ldots, m.$$

However, we generally deal with data that is not linearly separable, i.e., there is a sample $(\mathbf{x}^i, y^i) \in S_m$ such that

$$y^i \left(\langle \mathbf{x}^i, \mathbf{w} \rangle + b \right) \ngeq 1.$$

Therefore we introduce slack variables $\xi_i \geq 0$ for each sample $(\mathbf{x}^i, y^i) \in S_m$, such that

$$y^i \left(\langle \mathbf{x}^i, \mathbf{w} \rangle + b \right) \geq 1 - \xi_i.$$

The slack variables ξ_i allow a violation of the desired inequality. Since we wish the violation to be as small as possible, we seek a hyperplane that minimizes $\sum_{i=1}^m \xi_i$. Another objective is to find a hyperplane with a large margin. That is equivalent to minimizing the norm of the weight vector $\|\mathbf{w}\|$, because the lower the norm gets, the greater the margin will be. However, these two objectives can potentially be conflicting as a larger margin can lead to more violations of the inequality.

The following optimization problem captures the trade-off between the margin-maximization and the slack penalty minimization.

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}} \ & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i \\ \text{s.t.} \quad & y^i \left(\langle \mathbf{x}^i, \mathbf{w} \rangle + b \right) \geq 1 - \xi_i, \quad i = 1, \ldots, m, \\ & \xi_i \geq 0, \qquad\quad i = 1, \ldots, m, \end{aligned}$$

where $C > 0$ is a regularization constant. Higher C means higher norm $\|\mathbf{w}\|$ and thus a smaller margin.

2.3.3 Dual optimization problem

Let us introduce Lagrange variables $\alpha_i \geq 0, 1 \leq i \leq m$, and $\beta_i \geq 0, 1 \leq i \leq m$. Then we define the Lagrangian for the primal problem by

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i \left[y^i \left(\langle \mathbf{x}^i, \mathbf{w} \rangle + b \right) - 1 + \xi_i \right] - \sum_{i=1}^m \beta_i \xi_i.$$

Obtaining the Karush–Kuhn–Tucker conditions and plugging them into the Lagrangian leads to the dual formulation which is a convex quadratic program that reads

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y^i y^j \langle \mathbf{x}^i, \mathbf{x}^j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^m \alpha_i y^i = 0, \quad i = 1, ..., m, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, ..., m. \end{aligned}$$

The primal variable $\mathbf{w}$ can be expressed with respect to the dual variables

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y^i \mathbf{x}^i = \sum_{i \in \mathcal{I}_{SV}} \alpha_i y^i \mathbf{x}^i,$$

where $\mathcal{I}_{SV} = \{i \mid \alpha_i > 0\}$. Sample vectors $\mathbf{x}^i, i \in \mathcal{I}_{SV}$ are called *support vectors* as they *support* the hyperplane. The bias b can be obtained from any support vector $\mathbf{x}^i, i \in \mathcal{I}_{SV}$ by

$$b = y^i - \langle \mathbf{x}^i, \mathbf{w} \rangle.$$

Expressing the weight vector $\mathbf{w}$ in terms of the dual variables $\boldsymbol{\alpha}$ then leads to the hypothesis

$$f(\mathbf{x}) = \operatorname{sgn}\left(\left\langle \mathbf{x}, \sum_{i \in \mathcal{I}_{SV}} \alpha_i y^i \mathbf{x}^i \right\rangle + b \right) = \operatorname{sgn}\left(\sum_{i \in \mathcal{I}_{SV}} \alpha_i y^i \langle \mathbf{x}, \mathbf{x}^i \rangle + b \right).$$

2.3.4 Kernel trick

As we can observe, the training samples occur merely in the form of dot product in both the hypothesis and the dual problem. Let us define a new function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Such a function is called a *kernel* (or a *kernel function*) over $\mathcal{X}$.

We can now set

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle,$$

which allows us to rewrite the dual optimization problem to

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^i y^j k(\mathbf{x}, \mathbf{x}') \\ \text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i y^i = 0, \quad i = 1, ..., m, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, ..., m, \end{aligned}$$

and also the hypothesis

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i \in \mathcal{I}_{SV}} \alpha_i y^i k(\mathbf{x}, \mathbf{x}^i) + b \right).$$

We can define more complex kernel functions (than mere dot product). That way it is possible to influence the shape of the decision boundary and even make it non-linear. In consequence, we can classify data which are not linearly separable by employing a non-linear kernel. The next section introduces several kernels.

2.4 Kernel functions

There is a broad spectrum of different kernels, each of which is suitable for different data (or task). We shall define some of the most popular ones for data consisting of real-valued vectors, and, furthermore, a few so-called *string kernels*.

2.4.1 For real-valued data

Kernels in this section assume the input samples to be numeric vectors, i.e., $\mathcal{X} \subseteq R^D$, where D is dimension of the samples.

The *linear kernel* is the most basic kernel. It is defined as the dot product of its inputs

$$k^{\text{lin}}(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle.$$

Given its simplicity, it is fast to compute, but it is suited for linearly separable data only.

The *polynomial kernel* of degree d is defined as:

$$k^{\text{poly}}(\mathbf{x}, \mathbf{x}') = \left(\langle \mathbf{x}, \mathbf{x}' \rangle + r\right)^d.$$

The value of the parameter r affects the influence of higher-degree versus lower-degree monomials. However, values of 0 or 1 are often chosen. When $r = 0$ we call the kernel homogenous. The value of the degree d controls the flexibility of the decision boundary [8].

The *radial basis function kernel* (sometimes also called the gaussian kernel) with the definition

$$k^{\text{rbf}}(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$$

corresponds to the dot product of the input vectors in infinite dimensional space [14]. The parameter γ has a similar effect on the flexibility of the decision boundary as the degree d in the polynomial kernel [8].

2.4.2 String kernels

A string kernel is such a kernel that operates on strings, i.e., the input space $\mathcal{X}$ contains character sequences of finite length, although, the lengths of the sequences may vary in general.

The *spectrum kernel* utilizes an explicit feature mapping to transform the input vectors:

$$k_l^{\text{spect}}(\mathbf{x}, \mathbf{x}') = \langle \Psi_l(\mathbf{x}), \Psi_l(\mathbf{x}') \rangle,$$

where $\mathbf{x}, \mathbf{x}'$ are sequences over an alphabet Σ, and Ψ_l maps a sequence $\mathbf{x}$ into a $|\Sigma|^l$ dimensional feature space. Each dimension corresponds to the number of occurrences of a string $s \in \Sigma^l$ in $\mathbf{x}$ [8]. The parameter l is sometimes called the order of the kernel.

The *weighted degree kernel* assumes sequences of fixed length L and employs positional information:

$$k_d^{\text{wd}}(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \beta_k \sum_{i=1}^{L-k+1} \text{I}(\mathbf{x}_{[\text{i:i+k}]} = \mathbf{x}'_{[\text{i:i+k}]}),$$

where $\mathbf{x}_{[i:i+k]}$ is the substring of length k of $\mathbf{x}$ from position i. The major parameter, the degree d, sets the maximal length of compared subsequences. Another parameter is a vector of weights β_k. A suggested value for the weights is $\beta_k = 2\frac{d-k+1}{d^2+d}$ [15]. Such setting assigns lower weights for longer subsequences since they contain shorter subsequences that already contributed to the kernel output.

The *weighted degree kernel with shifts* [15] extends the weighted degree kernel by allowing some positional flexibility of matching substrings [8]:

$$k_d^{\text{wds}} = \sum_{k=1}^{d} \beta_k \sum_{i=1}^{L-k+1} \sum_{s=0}^{S} \delta_s \mu_{k,i,s,\mathbf{x},\mathbf{x}'},$$

where $\mu_{k,i,s,\mathbf{x},\mathbf{x}'} = \text{I}(\mathbf{x}_{[\text{i+s:i+d+s}]} = \mathbf{x}'_{[\text{i:i+d}]}) + \text{I}(\mathbf{x}_{[\text{i:i+d}]} = \mathbf{x}'_{[\text{i+s:i+d+s}]})$.

2.5 Evaluation metrics for model performance

When dealing with binary classification, we can divide the outcome of the classification to four classes with respect to the truth:

- *true positives* – positive examples classified as positive,
- *true negatives* – negative examples classified as negative,
- *false positives* – negative examples classified as positive,
- *false negatives* – positive examples classified as negative.

The number of examples in each class is denoted as TP, TN, FP, and FN, respectively.

2.5.1 Accuracy

Accuracy is defined as the ratio of correctly classified samples

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

A similar metric is *classification error* Error $= 1 -$ Accuracy.

Accuracy (or classification error) is not, however, a suitable evaluation method when dealing with very imbalanced classes [16]. Assume a dataset with examples of which 99% are negatives and only 1% positives. A trivial classifier that predicts always false achieves 99% accuracy (or 1% classification error).

2.5.2 Receiver Operating Characteristics curve

Receiver Operating Characteristics (ROC) captures the trade-off between *True Positive Rate* $\text{TPR} = \frac{\text{TP}}{\text{TP+FN}}$ and *False Positive Rate* $\text{FPR} = \frac{\text{FP}}{\text{FP+TN}}$. We often depict the trade-off as a curve (see Figure 2.3), and to sum it up into a single number, we then use the area under the curve (AUC). A notable property of the ROC curve is that it is not influenced by the class priors [16].

When given a classifier for which TPR $= 1.00$, then the classifier successfully detected all positive examples. However, if FPR $= 1.00$, the classifier designated all negative examples as positive as well.

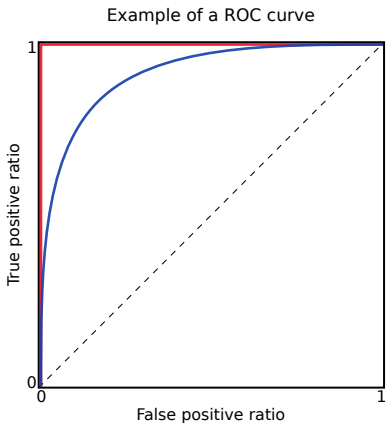

Figure 2.3. An example of a ROC curve. The dashed line represents a classifier that chooses between the positive and negative class randomly (with the same probability for both the classes). The AUC for such a classifier is 0.5. A better classifier (with AUC $= 0.89$) is represented by the blue curve. The red broken line shows a course for an optimal classifier with respect to the ROC (AUC $= 1.00$).

2.5.3 Precision and Recall

Precision-Recall (PR) curve shows the relationship between *precision* $\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{FP}+\mathrm{TP}}$ and *recall* $\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$. Note that the recall here is the very same value as the TPR that was discussed before. This evaluation metrics is especially useful when we are concerned in the positive class only [16]. We can compute AUC similarly as in the before case. Unlike ROC, PR reflects the class priors.

If a classifier achieves both the precision $\mathrm{P} = 1.00$ and recall $\mathrm{R} = 1.00$, then the classifier detects all positive examples and, furthermore, there are no false positives (i.e., every positive prediction is correct).

Chapter 3
Related work

When talking about intron detection, there are two related tasks to handle. One of them is splice site detection. In this particular task, we try to determine positions of donor and acceptor splice sites in a sequence of DNA (or RNA) nucleotides. The other task is to decide whether the DNA subsequence between a pair of complementary splice sites is an intron.

In the following sections, approaches that employ both traditional algorithms and machine learning are cited. In general, there is a variety of different machine learning methods that can be used in the problems mentioned above. We, based on the thesis assignment, constrain the broad spectrum of possible ML methods by removing the whole class of probabilistic methods from our consideration. Therefore, only non-probabilistic machine learning methods are stated.

3.1 Splice site classification

In all of the following works, a neighborhood (with a given size) around splice sites is assumed. The neighborhood determines subsequences that are used for prediction and is sometimes called a *window*. Furthermore, only a small subset of all possible *donor-acceptor* pairs is considered – often only the most frequent pair `GT-AG` – because the more pairs are considered, the more difficult the prediction gets.

Splice site prediction plays a crucial role in gene finding systems, an example of such a system is Genie [17]. In Genie, they use feed-forward neural networks to predict donor and acceptor splice sites. A similar approach was also described formerly [18]. To use neural networks on string data (such as DNA sequences), one must decide on the input encoding method. An orthonormal encoding method[1] is frequently used [18, 17, 19], but other encodings are used too. For example, there is a complementary encoding method[2] which is inspired by the complementary relationship between different nucleotides [20].

Apart from the artificial neural networks, support vector machines (SVM) are used and prove to be well suited for this task. Some approaches employ an explicit feature mapping. In [21], they use three types of features: *positional information* (presence or absence of a nucleotide), *compositional information* (presence or absence of a polymer), and *coding potential* (presence or absence of codons in reading frames). A similar approach is used by [22] where each sequence is considered as a sequence of pairs of nucleotides, and each pair is encoded as a 16-dimensional binary vector[3]. Another approach is to examine mono-nucleotide[4] and pair-wise nucleotide encoding (each pair is assigned a number between 1 and 16) combined with frequency differences between true and false sites [6]. In [23], they use a numerical encoding based on differences in error matrices for true, and false splice sites. Furthermore, they compared the performance

[1] For example, $A \mapsto 0001, T \mapsto 0010, G \mapsto 0100, C \mapsto 1000$.
[2] For example, $A \mapsto 1, T \mapsto -1, C \mapsto 2, G \mapsto -2$.
[3] There are 4 different nucleotides and, therefore, $4^2 = 16$ different pairs of nucleotides.
[4] Maps A, T, C, G to $1, 2, 3, 4$ respectively.

of neural networks, support vector machines and random forests for the task of donor splice site prediction using the proposed encoding.

When using *kernel functions*, an SVM can handle DNA sequences directly without an explicit feature mapping. The *spectrum kernel*, *weighted degree kernel* (WD), and *weighted degree kernel with shifts* (WDS) are, among other kernels, applied in computational biology [8]. The locality improved kernel, WD and WDS kernels are used for the splice site prediction very often [11, 21, 24]. In addition to SVM, further statistical information (such as intron and exon length statistics) can be used to improve prediction accuracy [11]. Efficient algorithms for computing the mentioned kernels, and also for fast SVM training, were proposed [25]. In [26], they compare two methods of multiclass SVM (where classes are *donor splice site*, *acceptor splice site*, or *none*), and evaluate ensembles of multiclass SVMs using bagging.

A completely different way to solve the splice site prediction problem is realized by an algorithm called BRAIN which infers a DNF formula from the training examples such that it describes the splicing rule [27]. An advantage of this approach is that the resulting formula (or the splicing rule) can be easily interpreted. A discriminant analysis can be used to predict the splice sites too [28]. Another approach is to employ a hypernetwork architecture to find DNA splice sites. *The hypernetwork architecture is a biologically inspired information processing system composed of networks of molecules forming cells, and a number of cells forming a tissue or organism* [29].

3.2 Intron classification

Many statistical analyses regarding introns (e.g., intron lengths) for various organisms have been performed. Following works studied different, specific organisms, and thus the results do not necessarily apply globally to all organisms.

A relation between the number of lengths that are multiple of three, and the number that are multiple of three plus one (or two) was studied. The numbers should be similar but it turned out that, based on introns from 29 eukaryotic species, skew in intron length distributions is common [30].

For a plant, it was shown that the length of introns and exons vary depending on the number of introns per gene [31]. For fungi, however, the introns tend to be short [32]. A correlation between the number of introns and variations in intron (and exon) lengths was discovered [33]; furthermore, a correlation between the lengths of neighboring introns was observed [34].

There were multiple approaches to intron prediction as a sequence classification problem. A set of deterministic rules to detect an intron was introduced [35]. Utilization of signal processing methods to analyze DNA sequences, and to further predict introns and exons was proposed [36]. In [37], they extract statistical information from DNA sequences using a wavelet-based time-series approach. The information is then used to construct feature vectors, which are further used to train an SVM with an aim to classify DNA sequences (exons or introns).

Support vector machines have been successfully used for sequence classification tasks, e.g., [38–40]. There has been an effort to interpret support vector machines with the aim to extract biologically relevant knowledge, for example, using Support Vector Multiple Kernel Learning [41]. SVMs were used to build a system which recognizes classes of proteins [42].

There were other methods for sequence classification based on, for example, sequence comparison [43], or compression-based induction [44].

Chapter 4
Data analysis

In this chapter, we describe the data we have been provided with and, furthermore, we present several statistical properties of the data. These properties are important in order to propose a reasonable solution. Later, we motivate our solution based on the statistical properties stated in this chapter.

4.1 The data

We are given DNA sequences and annotations describing individual genes and their positions in the DNA for 940 different organisms. The data is acquired from the genome portal of the Department of Energy Joint Genome Institute[1] (JGI) which is a web application that provides access to JGI genomic databases and allows searching or exploring genomes.

In biology, a species belongs to a genus which belongs to a family and so on. These are so-called taxonomic ranks and for the needs of thesis we consider[2] the following five of them (ordered from the most general to the most specific):

1. phylum [8],
2. class [45],
3. order [123],
4. family [306],
5. and genus [567].

The numbers in the brackets state the number of taxons for each of the taxonomic ranks.

4.1.1 Sequences

The DNA sequences are stored in a simple, yet convenient, text-based format called FASTA[3]. The FASTA files are assigned to the respective organisms using their filenames which always read as `<Organism>_AssemblyScaffolds.fasta`.

Each sequence in the FASTA format is described by a so-called *description line* which starts with the symbol `>` and precedes the particular DNA sequence (see Listing 4.1). In our data, the description lines are used to denote different continuous subsequences of the whole genome called *scaffolds*. Since a scaffold is a continuous DNA sequence, it always contain DNA of a single organism only. All text lines are usually shorter than some fixed value (it is 70 characters in our case).

The sequences should consist of A, C, T, or G only, but the character N—which, according to the nucleid acid notation[4], stands for "any base"—occurs in the data too.

[1] `genome.jgi.doe.gov`
[2] There is a taxonomic rank above phylum too, but it is not important for us since all the fungi we work with are contained in a single group on that level.
[3] `ncbi.nlm.nih.gov/BLAST/fasta.shtml`
[4] https://www.bioinformatics.org/sms/iupac.html

```
>scaffold_1
GATAGGCGCCATAGCCCTCCATTGTGGGTGTTAGAACAAGGGCAATTCCTGCCACCTATACTGGCAGCCT
ATGAGTGGCCCAGATTAGCTTAGTATGATTACATAATGCTCCCTATAACACTGGCTGAGAAACAAADATG
ATTCCTCAGCGTTCGTCTCTATCATTTGGGGATAATAGAATTAGAA
```

Listing 4.1. An example of a FASTA format. The first line is the description line; it states the name of the following scaffold. The scaffold consists of a sequence of 185 characters each of which denotes an individual nucleotide. The sequence is spread over three lines so that the number of characters per line does not exceed 70.

The occurrences might arise when creating the FASTA files from inaccurately sequenced DNA.

The overall size of the FASTA files is 38 GB, while the sizes of the individual FASTA files range from 2.2 MB to 325 MB and the number of bases distributed within scaffolds per file (proportionally) ranges from 10^6 to 10^8 bases.

4.1.2 Annotations

The gene annotations are distributed as GFF[1] files. The files are linked with the organisms using their filenames in a similar way as the FASTA files.

A GFF file consists of lines each describing a particular subsequence using nine columns separated by a tab (see Listing 4.2 for an example):

1. scaffold name,
2. source,
3. feature type,
4. start position (inclusive),
5. end position (inclusive),
6. score,
7. strand,
8. frame,
9. and a semicolon-separated list of additional attributes.

```
scfd_1   JGI  exon       919   1103 . - .   name "gm1"
scfd_1   JGI  CDS        919   1103 . - 2   name "gm1"
scfd_1   JGI  stop_codon 919   921  . - 0   name "gm1"
scfd_1   JGI  exon       1187  1322 . - .   name "gm1"
```

Listing 4.2. An example of a GFF file. This snippet, according to the first column, describes subsequences from a scaffold called `scfd_1`. All the subsequences belong to a gene named `gm1` which is located on a negative strand.

The scaffold name is the name stated in the description line of the scaffold (see Listing 4.1). The feature type determines the meaning of the subsequence and can be one of the following:

- exon,
- CDS (i.e., coding sequence),
- start codon (`start_codon`),
- or stop codon (`stop_codon`).

[1] `www.ensembl.org/info/website/upload/gff.html`

Note that one of the feature types is *exon*, which is not be confused with exon mentioned in Section 2.2. Exon as a feature type includes, in addition to coding sequences, also non-coding exon sequences. However, exon, as introduced in Section 2.2, means a coding sequence only and is denoted as *CDS* in the notion of the feature types. There is not any feature type for introns so their positions cannot be extracted directly, but it is possible to derive them based on the positions of other feature types. Section 4.1.4 deals with this problem.

Start and stop codon refers to the meaning explained in Section 2.2.

Start and end position assign the subsequence location within the scaffold (both the positions are inclusive).

Strand can be either `+` or `-` and determines DNA strand which contains the subsequence.

The list of attributes may contain *any* key-value pair. There is although a common key `name` with an associated value that states the name of a gene which includes the given subsequence. This key-value pair is present in case of every organism. However, for one organism called *Volvo1* the pair has a different semantic; it contains values such as *conserved hypothetical protein* instead of gene names.

We will omit a description of the fields source, score, and frame as they are irrelevant for the thesis.

4.1.3 Subsequence extraction

A tuple (N, S, P_s, P_e), where N is the name of a scaffold, S denotes a strand, and P_s, P_e are start and end position, is essentially sufficient information to locate a subsequence in a FASTA file. In the case of positive strand, the extraction is fairly straightforward.

1. Find a sequence of a scaffold named N,
2. return the sequence between positions P_s and P_e inclusively while skipping line breaks which are not considered as a part of the sequence.

For subsequences located on negative strand, the extraction involves two additional steps:

1. reverse the extracted subsequence,
2. rewrite each character of the extracted subsequence to its complement (see Table 2.1).

See Listing 4.3 for an example of both a positive and negative strand extraction which followed the instruction described above.

In general, sequences in the FASTA files represent the positive strand of DNA. Therefore, when working with negative strand, one needs to reverse the sequences a rewrite them according to the complementarity.

4.1.4 Getting introns from the annotations

To find introns using specific methods of machine learning, we need to label intron subsequences. To do that, we need to know their positions, but, as aforementioned, the positions of introns cannot be extracted from the GFF files directly. This section describes an algorithm which allows us to get the positions for a given gene.

We assume that it is possible to obtain all non-intron subsequences for the gene, which can be done simply by grouping the subsequences by the gene name which is always among the additional attributes. Also, we reckon that any gene starts and ends with a non-intron sequence. This is trivially satisfied as a gene always starts and ends with an exon.

```
>scaffold_1
ACATAGACATACTTGATGGATGAGAATGGATCAACTAGAACTGCTCAGAGCGTAGGCATACGCATCCGCA
TCAGACGTCAGCACGCATACTCGCATCCCTGACCATCACGCATCAGCTACTCGATCATGCAGCATCTTGC
AGACGCCTACATATCCCAATGCACGATGCGATCTGAAATAC
>(scaffold_1, +, 68, 75)
GCATCAGA
>(scaffold_1, -, 68, 75)
TCTGATGC
```

Listing 4.3. An example of subsequence extraction. The listing uses the FASTA format. The scaffold named `scaffold_1` contains a source sequence of 181 characters. The other two scaffolds represent results of two extractions. Names of those two scaffolds are tuples which express origin of the extractions as described in the beginning of this section.

Given the gene, let us denote the number of the non-intron subsequences as m. We can then represent the non-intron subsequences as pairs $(s_i, e_i), 1 \leq i \leq m$, where s_i and e_i are start and end positions of i-th subsequence, respectively. Without any loss of generality, we assume $s_i \leq e_i$ for $1 \leq i \leq m$, and $s_i \leq s_{i+1}$ for $1 \leq i < m$. The first assumption can be satisfied by swapping s_i and e_i for any i that violates the assumption, and the second one by employing a simple ordering on the subsequences. Then we can define the algorithm as seen in Listing 4.4.

```
FUNCTION ExtractIntrons(s, e)
  start := min(s)
  end   := max(e)

  last    := start
  introns := []

  FOR i IN 1..m DO
     IF s[i] > last + 1 THEN
        append (last+1, s[i]-1) to introns
     END IF
     last := max(last, e[i])
  END FOR

  RETURN introns
END FUNCTION
```

Listing 4.4. Pseudocode of an algorithm that extracts intron positions for a gene. The arguments `s` and `e` are, respectively, lists of start and stop positions of non-intron subsequences as defined in the text. The variable `m` refers to the number of non-intron sequences as introduced in the text as well. The algorithm result is a list of pairs of start and end positions for all introns in the given gene.

In the beginning, the algorithm finds the minimal start position, and the maximal end position. Then, after making exactly m iterations to find the intron positions, it returns the result. The algorithm thus has a linear time complexity $O(m)$.

4.2 Statistical analysis of the data

Statistical analysis of the data might lead to interesting findings. Therefore in this section, we examine the DNA sequences present in the FASTA files. We remove the

organism *Volvo1* from the analysis as the GFF file of this organism does not respect the discussed `name` attribute rule (see Section 4.1.2) and thus it is not possible to determine its gene positions.

4.2.1 Splice site pairs

As mentioned before in the Section 2.2.1, the most common pair of splice sites is GT (the donor site) and AG (the acceptor site). This statement holds in the case of our data too. Over 97% of all the introns start with GT and end with AG. Apart from that, a wide range of 426[1] different pairs of splice sites occurs in the data.

Figure 4.1 displays the relative frequency of four most frequent splice site pairs. Every other pair has its relative frequency lower than 0.1%.

In the figure, we can observe that the most common acceptor site AG forms a pair with another dimer (different than GT) too, the pair GC—AG is even the second most common splice site pair. In fact, both the donor site GT and the acceptor site AG pair with all 16 possible dimers (consisting of A, C, T, or G); although such pairs are not frequent.

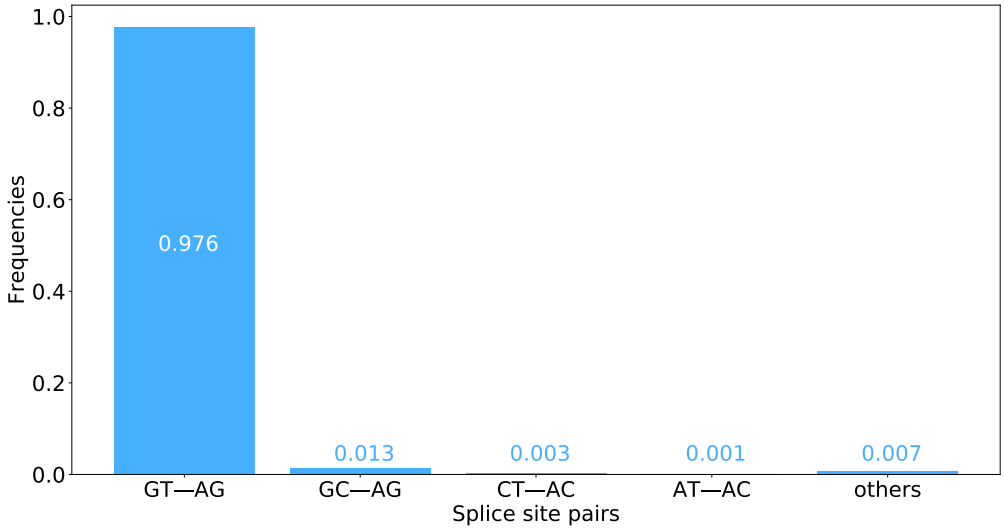

Figure 4.1. The relative frequency of four most frequent splice site pairs.

4.2.2 Nucleotide bases within introns

If we divide all the DNA sequences into two disjoint groups of intron and non-intron sequences, we can compare nucleotide distributions for both the groups. It turns out that there are distinct differences—see Figure 4.2.

Despite not being a nucleotide base, we also included frequencies for the character N to get a complete view of the distributions. However, ignoring the frequency of N, we see that the distribution for the non-intron sequences is *almost* uniform. On the contrary, the intron sequences have a strong presence of thymine (T), when compared to the non-intron sequences, which could be explained by the polypyrimidine tracts being present in the introns (see 2.2.1). Another, yet not that significant, difference between the two distributions is in the guanine (G) frequency which is greater in case of the non-intron sequences.

[1] The number includes splice sites containing N too. Otherwise there would be at most $4^2 \cdot 4^2 = 256$ possible splice site pairs.

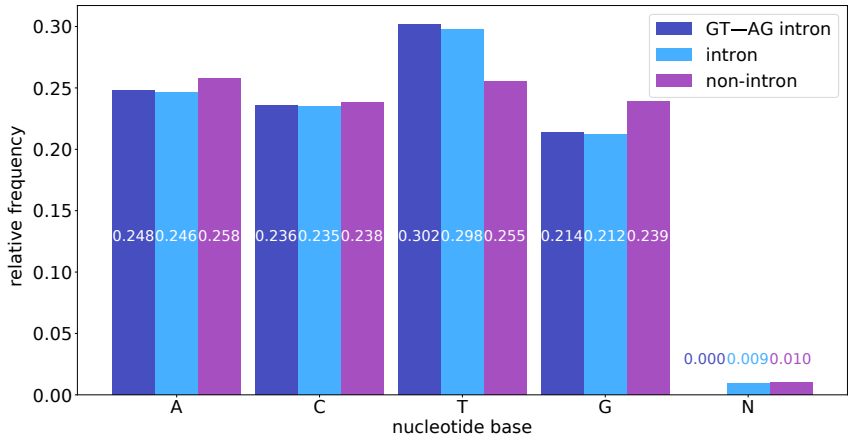

Figure 4.2. Relative frequencies of different nucleotide bases (or characters) within selected introns, within all introns, and outside introns. The selected introns are such introns that start and end with GT and AG, respectively, and do not contain any N.

Figure 4.2 also depicts a distribution for a group of introns that start with GT, end with AG, and consist purely of A, C, T, or G. This group can be viewed as a group of *typical* (in sense of splice sites) introns that are well sequenced (so they contain exact nucleotides). These typical introns are in the majority (see Figure 4.1]) and thus the nucleotide distribution of these introns does not deviate from the distribution of all introns very much.

4.2.3 Intron lengths

As mentioned in the related work, it has been reported that introns of some fungal organisms tend to be short [32]. Table 4.1 and histogram in Figure 4.3 imply the same for our data.

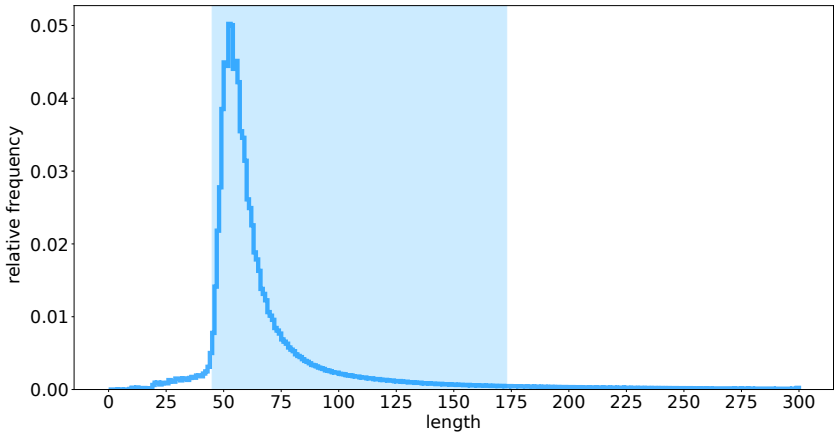

Figure 4.3. Intron lengths histogram. The blue area displays range between the 5th, which is 45, and the 95th percentile, which is 173. Thus, 90% of all intron lengths are between 45 and 173. Although it is missing in the plot, the 99th percentile is 476.

There are multiple organisms with single-base introns, e.g., *Abobi1*[1]. The intron having the maximal length belongs to *Gymlu1* organism and is located in a gene named

[1] `https://genome.jgi.doe.gov/cgi-bin/dispGeneModel?db=Abobi1&id=471248`

statistic	value
minimum	1
average	81
median	58
maximum	49062

Table 4.1. Basic statistics of intron lengths (see a related histogram in Fig. 4.3).

`gw1.144.11.1`[1]. Given all the intron lengths, the 99th percentile is 476 which allows us to ignore very long introns and still cover 99% of the domain.

For comparison, the average intron size for the human genome is more than 5000 bases [45].

It is useful to examine lengths of the individual phyla (see Table 4.2). Phyla *Chytridiomycota* and *Zoopagomycota* have abnormally long introns; on the other hand, *Cryptomycota* and *Microsporidia* have the opposite.

phylum	number of species	median intron length
Ascomycota	521	62.5
Basidiomycota	317	63.0
Blastocladiomycota	4	75.0
Chytridiomycota	22	184.0
Cryptomycota	2	28.0
Microsporidia	8	31.0
Mucoromycota	50	74.5
Zoopagomycota	16	204.5

Table 4.2. Median lengths of introns in different phyla.

4.2.4 Portion of introns in the data

Introns represent just a small portion of the entire DNA (also partly because they are short as concluded in the previous section): only 3.7% of all nucleotides belong to introns. A visualization of the portion is shown in Figure 4.4.

We have stated that GT and AG are the most common donor and acceptor splice sites, respectively. However, given the DNA sequences we have, *only about 0.82% of all GTs and 0.72% of all AGs are the actual splice sites.* The difference of 0.1% between the portions arises mainly from the fact that there are about 1.14 times more AGs than GTs in the DNA data.

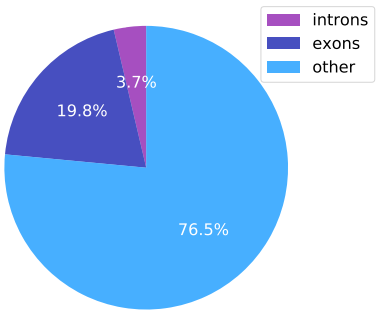

Figure 4.4. Portion of DNA that belongs to introns, exons, or neither of them.

[1] `https://genome.jgi.doe.gov/cgi-bin/dispGeneModel?db=Gymlu1&id=125403`

4.2.5 Intron distribution on strands

Introns can occur on both the positive and negative strand of DNA. In the provided data, the distribution between the strands is approximately uniform:

- 50.074% of introns lie on the positive strand,
- and 49.926% on the negative strand.

4.2.6 Summary

Let us summarize the important statistical properties that were discovered.

- The vast majority of all splice site pairs respect the GT-AG rule,
- there are statistical differences between intron and non-intron sequences on the level of nucleotide distributions,
- the introns are generally short (e.g., when compared to the human genome),
- a typical intron is about 60 bases long,
- however, the lengths of typical introns of different phyla vary (e.g., *Zoopagomycota* has very long introns compared to the others),
- more than 75% of DNA[1] is represented by non-coding sequences (i.e., less than a quarter of the DNA are genes),
- and finally, the number of introns on the positive strand is (almost) the same as on the negative strand.

[1] within the provided data

Chapter 5
Classification process

Given a long sequence of fungal DNA, the ultimate goal of the thesis is to find introns contained in the DNA. In other words, we need to localize start and end positions of the introns. As mentioned before, each intron starts and ends with the donor and acceptor splice site, respectively. Although there are many possible pairs of donor and acceptor splice sites, we will consider only introns starting with GT and ending with AG. By doing so, we substantially reduce the task complexity while still covering most of the domain, as over 97% of introns stick to the GT—AG rule (see Section 4.2.1). If we also considered the minor splice sites, it would necessarily lead to many false detections because only a negligible fraction of dimers that form these minor splice sites are splice sites.

A "naive" algorithm solving the task could be defined as follows.

```
1  FUNCTION find-introns-naive(sequence):
2      FOR i IN 0..length(sequence) DO
3          IF sequence[i, i + 1] = 'GT' THEN
4              FOR j in (i + 2)..(i + neighborhood):
5                  IF sequence[j, j+1] = 'AG' THEN
6                      is_intron := classify(span(sequence, i, j+1))
7                      IF is_intron = 1 THEN
8                          output sequence[i, j+i]
9                      END IF
10                 END IF
11             END FOR
12         END IF
13     END FOR
14 END FUNCTION
```

Listing 5.1. A naive approach to the intron retrieval. The algorithm takes a sequence, finds all GTs in the sequence, and for each of them, it searches some forward neighborhood for AGs. If an AG is found, the sequence between the GT and the AG is considered as a candidate intron sequence and therefore is passed to classification (function `span` returns the sequence extended with some neighborhood). Based on the classification result, the algorithm then decides whether the sequence is an intron.

The algorithm would be inefficient as it produces a candidate intron for any GT and AG that are within their neighborhood. The number of such candidates depends on the size of the neighborhood (which should be based on the results presented in Section 4.2.3) however it could get very large. Moreover, the vast majority of the candidates would be negative examples.

A better approach is to classify each GT and AG independently beforehand, and afterward produce the intron candidates by pairing the positively classified splice site candidates only.

One could argue that it is unnecessary to process a GT if there is no AG in its neighborhood and vice versa. However, this is extremely unlikely to happen, assuming a

reasonable neighborhood size (Figure 5.1 displays a graph of a function which represents the probability). Consequently, this approach produces proportionally fewer intron candidates when compared to the naive algorithm.

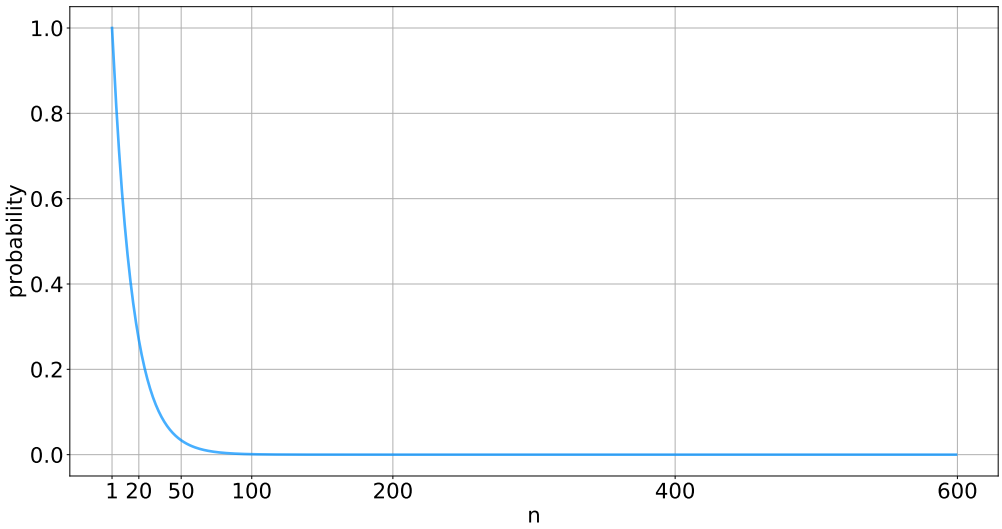

Figure 5.1. The conditional probability that there is no AG in a neighborhood of size n after a given GT. For simplicity, we assume that the nucleotides are distributed uniformly and independently. We should consider intron lengths up to a few hundreds of bases (see Figure 4.3), and, as we can observe, for $n \geq 100$ the probability is very close to zero. See Appendix B for computation of the probability.

5.1 The framework

Based on the previous findings, we can now set out a framework (or a general process) which we will use to find introns in a DNA sequence.

Figure 5.2 contains a holistic view of the classification process. There is an input, an output, and three different, consecutive tasks. Each part is introduced and described in the following sections.

Figure 5.2. Schema of the classification process. The classification process input is a FASTA file containing DNA sequences. In the first step, every splice site candidate within the DNA sequence is classified using a trained model. Then in the second step, intron candidates are formed based on positively classified splice site candidates. The third and final task is to classify the intron candidates, and to produce a FASTA file consisting of positively classified intron candidates.

5.1.1 Input

The process has a FASTA file as its only input. The FASTA file contains DNA sequences in which introns are about to be detected.

Generally, the FASTA files always contain sequences coming from only a single strand—suppose the positive one—but given those positive strand sequences, it is always possible to get sequences from the negative strand too, as the strands are com-

plementary. However, there are additional steps involved in order to get the negative strand sequences (we explained the steps in Section 4.1.3).

To keep the classification process simple, we completely abandon the concept of strands. Only sequences read directly from the input FASTA file are considered (i.e., there is no reversing nor rewriting of the sequences). Of course, this way we would implicitly lose all potential introns located on the negative strand, which is about half of all introns in a genome (see Section 4.2.5). Since this is not acceptable, we come up with a simple solution.

We suggest extending the input FASTA file by introducing new scaffolds which are complementary to the scaffolds from the original input. Although it causes the input to double its size, it allows us to keep the whole process straightforward and simple while not giving up half of the potential introns.

Furthermore, we can name the newly introduced scaffolds based on names of the scaffolds from which they originate, e.g., by adding a prefix to the original names. Such a naming convention binds together two different scaffolds that, however, represent complementary strands of the same sequence. See Listing 5.2 for pseudocode of this procedure.

```
1  FUNCTION extend-input(scaffolds):
2      FOR i IN 0..length(scaffolds) DO
3          output (name(scaffolds[i]), sequence(scaffolds[i]))
4
5          new_name := "neg_" + name(scaffolds[i])
6          compl_seq := complement(reverse(sequence(scaffolds[i])))
7          output (new_name, compl_seq)
8      END FOR
9  END FUNCTION
```

Listing 5.2. Pseudocode of a function that extends an input with complementary sequences. The function takes a list of all original scaffolds. It then outputs each of the original scaffolds and their respective complementary scaffolds which are named the same as the originals but with a prefix `neg_`.

5.2 Task: Classify splice sites

The purpose of this task is to classify all splice site candidates that are contained within the input FASTA file.

It follows from the related work that methods of statistical machine learning are frequently used for splice site classification. Many approaches use artificial neural networks or support vector machines, in particular.

Artificial neural networks can learn very complex decision functions, but they tend to have many parameters to tune. Moreover, ANNs do not require a convex cost function, but they do not guarantee to find the global optimum. On the contrary, SVMs always find the global optimum and have only a few parameters in exchange for a simple decision function which we can however influence using a specific kernel.

Among the related work, the best results have been achieved by a support vector machine [24]. And therefore we too shall follow this approach.

The best performing SVM employed weighted degree and weighted degree with shits kernels, both of which are string kernels (see Section 2.4.2). These kernels seem like a natural choice as they should be able to utilize the intron characteristics such as the presence (and location) of polypyrimidine tracts or the branch point (see Section 2.2.1).

Both the kernels have the same definition except there is a possible subsequence shifting involved in WDS kernel which makes it more general than WD. Because of that, WDS kernel may yield better results [24] in exchange for longer computation time due to additional iterations implied by the subsequence shifting.

Since we work with large datasets, execution speed is an essential criterion. We, thus, will use the WD kernel.

5.2.1 Classification problem

Our goal is to classify both donor and acceptor splice site candidates. The approach we have chosen above, however, leads to considering the donors and acceptors independently. Splitting the classification of the splice sites into two independent tasks also allows us to run the tasks *in parallel* which is very convenient in terms of execution time.

The classification tasks for both the donors and acceptors are essentially the same and thus have the same definition.

For each splice site candidate, we extract a sequence of length N such that the splice site is contained within the sequence. The length N and location of the splice sites within the sequence are given by parameters which are explained in Section 5.2.2. The sequences are considered input examples for a classifier.

Given M sequences, the classifier then assigns each sequence to a positive or negative class, meaning that the sequence contains a true or false splice site, respectively. Therefore we deal with a binary classification problem with the input space

$$\mathcal{X} = \{A, C, T, G\}^N$$

and the target space

$$\mathcal{Y} = \{-1, +1\}.$$

SVM is a supervised learning algorithm which means that it requires to know the true classes for each sequence during its training. This is not a problem since the true classes follow from the sequence annotation (see Section 4.1.2).

5.2.2 Parameters

Since we use the WD kernel, two parameters arise directly:

- a degree d,
- and weights β_k (both are explained in Section 2.4.2),

and, because the WD kernel operates over fixed-length sequences, there are other two parameters:

- the sequence length,
- the splice site position within the sequences.

The value of the degree d has to be determined empirically, however in the related work, values around $d = 20$ were used. Regarding the weights β_k, we will use the values $\beta_k = 2\frac{d-k+1}{d(d+1)}$ as suggested in [15].

We call the sequences, containing the splice sites, *windows* around the splice sites. The windows are parameterized by the number of bases before and after the splice site. Let us denote these two numbers W_L and W_R, respectively.

It follows from the definition of the numbers W_L, W_R that they determine both the window size (or the sequence length)

$$N = W_l + W_R + 2$$

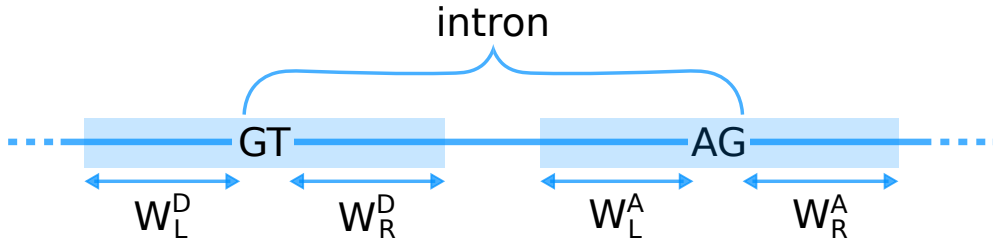

Figure 5.3. Splice site windows. The blue baseline represents a continuous DNA sequence which contains an intron that starts with a GT donor and ends with an AG acceptor. The blue bar behind the donor GT depicts the window around the donor. The analogous applies for the blue bar behind the acceptor AG. The arrows depict the value of W_L^D, W_R^D in case of donors, or W_L^A, W_R^A in case of acceptors. Note that the part of the window defined by W_R^D (or W_L^A) always interferes with an intron (assuming true splice site).

and the position of splice sites within the window. See Figure 5.3 for a graphical explanation.

5.3 Task: Generate pairs

The previous task, the splice site classification, classifies every splice site candidate that is present within the input FASTA. From now on, we consider only the positively classified donors and acceptors and use them to create a set of intron candidates.

As mentioned before, an intron starts with a donor and ends with an acceptor. We therefore iterate over each (positively classified) donor and try to find one or possibly more (positively classified) acceptors that are located within a reasonable distance D from the donor.

5.3.1 Distance between splice sites

The distance D between splice site pairs directly determines the length of produced intron candidates

$$N = D + 2$$

and therefore, also the length of introns we can detect. The value of D should be based on the length of real introns which we analyzed in Section 4.2.3.

Let us denote a lower bound D_L and an upper bound D_U, and let them constrain the distance

$$D_L \leq D \leq D_U.$$

By setting these bounds, we control the length of intron candidates. It must hold $D_L \geq 2$, as the minimal possible intron candidate is `GTAG` with length of $N = 4$ where the distance between its splice sites is $D = 2$. Suitable values of the bounds are discussed in Section 5.4.

5.4 Task: Classify intron

The part of the process we have introduced so far produces a set of intron candidates, i.e., sequences which, we believe, could be introns based on their splice sites. However, it turns out that many of the candidates are false positive examples. By employing the final step of the process (see Figure 5.2), we aim to reduce the number of false positives while maintaining the number of true positives as much as possible.

```
FUNCTION make-pairs(donors, acceptors, d_min, d_max):
    FOR i IN 0..length(donors) DO
        pos := position(donors[i])
        to_pair := filter(acceptors, pos + d_min, pos + d_max)
        FOR j in 0..length(to_pair) DO
            output (donors[i], to_pair[j])
        END FOR
    END FOR
END FUNCTION
```

Listing 5.3. Pseudocode for pair generation. The function takes positively classified donors and acceptors, and the minimal and maximal distance between two complementary splice sites. Function `filter` returns acceptors located between given positions.

The problem we deal with in this task is similar to the problem we try to solve in the splice site classification. The current task is also a binary classification problem. However, there is an essential difference in the input space: unlike the splice site classification, here we deal with sequences of variable lengths. It is therefore impossible to use the same approach as in the splice site classification because the weighted degree kernel requires fixed-sized sequences. It is also useful to use a different feature representation (i.e., kernel) in order to bring different kind of information to the process.

We touched the topic of intron classification in the related work (Section 3.2), but none of the mentioned approaches is very suitable given our setup. We bring up a simple yet effective solution.

Once again, we will use a support vector machine but with the spectrum kernel in this case. We introduced the spectrum kernel in Section 2.4.2. It "compares" two sequences over an alphabet Σ based on the number of occurrences of subsequences Σ^l within each sequence. The subsequences length l is a parameter of the kernel, and the alphabet $\Sigma = \{A, C, T, G\}$ in case of DNA sequences.

The motivation of our approach consists of two parts:

- SVMs solve binary classification problems exceptionally well,
- there are differences between the nucleotide distribution of intron and non-intron sequences (see Figure 4.2).

When the kernel parameter $l = 1$, the kernel compares the sequences by the number of occurrences of each nucleotide. Hence, this value of l directly utilizes the differences in the nucleotide distributions to determine intron and non-intron sequences (see Figure 4.2).

However, greater values of l would induce feature space of higher dimensionality which could be beneficial for the classification as it could capture sequential regularities. Thus we will perform experiments to choose the best parameter l.

This approach relies on the statistical differences in l-mer[1] distributions of intron and non-intron sequences. The longer the sequences are, the stronger and more robust differences can be expected.

The length of sequences can be controlled by the previously introduced bounds D_L, D_U. Let us thus set the lower bound $D_L = 28$ (which implies intron lengths $N \geq 30$), and the upper bound $D_U = 598$ (which implies $N \leq 600$). Given the provided data, approximately 98% of true introns fit in this range. Should the classification

[1] l-mer is a nucleotide subsequence of length l.

target more typical introns, the bounds can always be tightened closer to the median length (see Section 4.2.3).

5.4.1 Output

After the intron classification is done, we will get a set of supposedly true introns. It is the final result of the whole classification process, and therefore we need to store the output in a file. Since the output is, as well as the process input, a set of DNA sequences, we will stick to the FASTA file format.

It is also important for the output to contain origin (scaffold name along with start and end positions) of the introns. It is however possible to store this kind information in the description lines (see Section 4.1.1). Let us introduce a new convention for the description lines of introns for this purpose: `>SCAFFOLD START END`, where `SCAFFOLD` is the name of the scaffold that contains the given intron, and `START` with `END` declare the start and end positions. See Listing 5.4 for an example of the introduced syntax.

```
1 >scaffold_1 1930 1971
2 GTACTAGCTGCTATAGCTAGTGGTAGAACCATGTTTGACAAG
```

Listing 5.4. An example of a description line of an intron. The intron is located at scaffold named `scaffold_1`, starts at position 1930 and ends at 1971.

Note that the output does not contain any information about strands. It is so because of the simplification introduced in Section 5.1.1. According to our suggestion, the information about the strand of origin is in the scaffold name.

5.5 Fungi heterogeneity and classification models

We expect that a single model (either for splice sites or introns) will not be sufficient for all fungi. But we naturally try to minimize the number of models as more models means more executions of the classification pipeline.

To minimize the number of models means to find the most general taxonomic rank (see Section 4.1) for which a trained model performs (almost) as well as in case of more specific ranks.

Chapter 6
Data processing tools

This chapter describes two major tools we used to process the data.

6.1 Extraction from FASTA

The solution consists of two tasks, the classification of splice site and introns, where we employ support vector machines. In Section 5.2.2, we defined that the data for the splice sites classification will consist of fixed-size windows (or sequences) around the splice site candidates. For the intron classification, the data is a set of sequences of variable lengths (see Section 5.4).

The datasets for both the tasks consist of subsequences of scaffolds stored within the FASTA files we have. Therefore we need a tool that would allow us to extract an arbitrary subsequence from a given scaffold located in a given FASTA file.

We have implemented such a tool and called it `extract-fasta`. It uses only the command line interface (CLI) to communicate via a simple protocol.

The program expects the name of a FASTA file as its only argument. When running, it reads the whole FASTA file, parses all scaffolds from the file and keeps them in a map. It follows that the whole FASTA file is loaded into RAM which is questionable. However, the maximal size of a FASTA file within the data is 325MB, and that is—given the typical memory size being over 8GB nowadays—relatively insignificant.

Once the FASTA file is loaded, the program reads line by line from the standard input where the user enters locations of the subsequences they wish to extract. The input is expected to respect the following form: `scaffold strand start end\n`. For example, `scaffold_1 + 217 302` causes extraction of the subsequence starting at position 217 and ending at position 302 on the positive strand of scaffold *scaffold_1*.

For each line of the input, the program writes the extracted sequence to the standard output. The sequences are outputted in the FASTA format meaning that a description line precedes each extracted sequence. The description line contains the origin of the extracted sequence which is the same as the input line that invoked the sequence extraction.

Note that the program accepts strands. If the strand is `-`, the program automatically reverses and rewrites the given sequence so that the outputted sequence is the correct, complementary one.

We have to consider the strands now, despite not considering them during the classification, because about half of genes are located on the negative strand (see Section 4.2.5). So if we only worked with the positive strand, we would lose half of the data available for training of the models.

Since the program uses the standard input and output, we can use features of the UNIX command line such as pipes or I/O redirection (see an example of typical usage of the program in Listing 6.1).

Because the scaffolds are stored in a map, the complexity to get a scaffold given its name is constant. Then, in case of the positive strand, the program returns a substring

```
./find-introns fung.gff | extract-fasta fung.fasta > fung-introns.fasta
```

Listing 6.1. An example of a typical usage of `extract-fasta` program. The example contains two programs, `find-introns` and `extract-fasta`, which are chained together using the pipe (suppose that the output of `find-introns` respects the syntax of the input of `extract-fasta`). The final result is then redirected to a new file called `fung-introns.fasta`. The idea of this whole pipeline is that program `find-introns` extracts positions of introns from file `fung.gff`, then passes them to `extract-fasta` which outputs the introns into a new FASTA file.

of the scaffold, which has linear complexity in the length of the extracted sequence. In case of the negative strand, there is also the additional "complementarity" procedure which has, however, linear complexity too. Overall, an extraction of a subsequence of length n has a linear complexity $O(n)$.

6.2 GFF processing

The GFF files contain sequence annotations (see Section 4.1.2). We need to be able to parse the annotations in order to determine intron positions which is crucial to train and test the SVM classifiers.

We have implemented a Python script named `process-gff` which goes through a GFF file, parses its content and outputs intron positions.

The GFF files do not contain positions of introns directly (see Section 4.1.4) so we follow the algorithm described in Listing 4.4 to extract the positions.

The algorithm requires us to group all non-intron sequences by the genes to which they belong. We can do this by using a dictionary (where keys are gene names and values are lists of non-intron sequences) and a single iteration over all lines each of which we would parse to get the gene name, and then store into the dictionary.

Consequently, the processing tool has a linear time complexity $O(n)$, where n is the number of lines in a given GFF.

Chapter 7
Splice site classification

In this chapter, we describe the implementation of the splice site classification. In particular, we touch the topics of parameter selection, or generalization capability (concerning the taxonomic ranks of fungi).

However, at the beginning of the chapter, we discuss the selection of a machine learning library that we coincidentally used for both the splice site and intron classification. We also include a code snippet which we used to perform the splice site classification tasks.

7.1 Machine learning library

As stated in Section 5.2 and 5.4, we have two types of classifications both of which we decided to deal with using SVMs. Since support vector machines are extensively used, there are libraries providing efficient and optimized implementations of SVMs.

We were considering two different libraries:

- Scikit-learn,
- and Shogun.

Scikit-learn [46] is a machine learning library featuring many well-established algorithms apart from SVM. This library has many users, large community, a great online documentation, and the provided Python API is very convenient to use.

However, Scikit-learn implements only a few kernels; moreover, none of them is a sequence kernel. Although there is a way of implementing a custom kernel function, it is not viable for large data because the kernel function must return the Gram matrix of size $n \times n$ given n training samples. Imagine having 200,000 training samples. That would imply a matrix of $4 \cdot 10^{10}$ elements. Assuming each element is a 4-byte number, we would need 149GB of RAM. This leads us to Shogun.

Shogun [47] provides a smaller set of algorithms when compared to Scikit-learn, but it implements SVM too. In general, the available documentation and provided Python API is worse than in the case of Scikit-learn. However, what makes Shogun a great library is the vast range of implemented kernel functions—both the weighted degree and spectrum kernel are present. The kernel implementations are also focused on large-scale applications, so there is not an issue with the capacity of RAM.

Based on the advantages and disadvantages stated before, we conclusively decide to use the SVM implementation from Shogun instead of Scikit-learn for both the splice site and intron classification.

7.1.1 Script

We used a Python script to train and evaluate splice site models. See Listing 7.1 for a code snippet. The code is very straightforward.

1. read the input dataset,
2. split the dataset into train and test data,

3. create feature objects (which are instances of classes from Shogun) for train and test sequences,
4. analogically, create label objects for train and test labels,
5. instantiate a kernel with a given degree d,
6. create a new SVM instance with given the parameter C, the kernel, and the true train labels,
7. train the SVM,
8. classify sequences in the test dataset (and perhaps in the training too),
9. evaluate the model.

```
import shogun as sg
import numpy as np

data = read_data(data_filename, window=(W_L, W_R))
train, test = split_data(data, test_size)

train_features = sg.StringCharFeatures(train.sequence.tolist(),
                                       sg.RAWBYTE)
test_features = sg.StringCharFeatures(test.sequence.tolist(),
                                      sg.RAWBYTE)

train_labels = sg.BinaryLabels(np.array(train.label))
test_labels = sg.BinaryLabels(np.array(test.label))

kernel = sg.WeightedDegreeStringKernel(train_features,
                                       train_features,
                                       degree)

svm = sg.LibSVM(C, kernel, train_labels)
svm.train()

test_predict = svm.apply_binary(test_features)
// compute evaluation metrics...
```

Listing 7.1. A snippet of code used for training and evaluation of a splice site model.

7.2 Parameter selection

In Section 5.2.2, we described parameters that arise from using the WD kernel:

- a degree d,
- weights β_k,
- and parameters W_L, W_R for the window size.

There is also another parameter C, which is the regularization constant of the SVM (see Section 2.3.2), that allows us to control the decision boundary of the SVM.

We have already set the weights to be $\beta_k = 2\frac{d-k+1}{d(d+1)}$ in Section 5.2.2. Coincidentally, this is also the default setup for the WD kernel in Shogun so we can omit the weights from further considerations about the parameters.

There are DNA sequences of 940 different fungi, each of which belongs to one of 567 different genera, 406 families, 123 orders, 45 classes, and 8 phyla. We already mentioned in Section 5.5 that a single universal model will not be sufficient for all the

fungi. However, to get a basic understanding of how different parameters influence the model performance, we will start with a single, arbitrary fungus—we picked *Armost1*[1] from phylum *Basidiomycota*.

7.2.1 Dealing with unbalanced datasets

In Section 4.2.4, we stated that less than 1% of all splice site candidates are true splice sites. Therefore, uniform sampling of the splice site candidates from the data necessarily leads to a dataset with extremely unbalanced class priors where more than 99% of all examples are negative.

Training a SVM using such a dataset is challenging. Since the goal of the SVM is to minimize the number of misclassifications (i.e., maximize its accuracy), it tends to classify all examples as negative and achieve more than 99% accuracy.

Generally, there are two approaches to tackle this issue:

- subsample the negative examples to make the difference in the class priors less significant,
- or set different regularization constants C^-, C^+ for the positive and negative class.

The subsampling approach modifies the original class priors directly. The second approach does not affect the class priors but it allows us to set $C^+ > C^-$, i.e. greater penalty for misclassification of positive examples than negative. Setting $C^+ = 2C^-$ is analogous to oversampling the positive class by a factor of 2.

We have tried both the approaches. The results are in the following sections. In all the experiments, we did not use any sequences (i.e., windows around splice sites) containing N (see Section 4.1.1).

7.2.2 Subsampling the negative class

We have performed a grid search through 81 combinations of parameters

- $C \in \{0.1, 1, 10\}$,
- $d \in \{15, 20, 25\}$,
- $W_L \in \{60, 80, 100\}$,
- and $W_R \in \{60, 80, 100\}$.

Within the sequenced DNA of Armost1, only about 2.7% of all splice site candidates are true splice sites. Therefore, the class priors are $P(+) = 0.027$ and $P(-) = 1 - P(+) = 0.973$. To alleviate the imbalance, we decided to subsample the negative class and keep only 10% of the negative examples. Consequently, the class priors changed to $P(+) = 0.27$ and $P(-) = 0.73$.

The donor dataset consisted of 413 042 examples: 94 886 positives and 318 156 negatives. The acceptor dataset contained 441 773 examples: 96 697 positives and 345 076 negatives. For both the donors and acceptors, we used 65% of the data for training and 35% for testing.

Many configurations led to high accuracy and recall for both the donors and acceptors. Precision is high too, but it is computed on a dataset with modified class priors, and therefore the precision is too optimistic. To get a meaningful estimate, we need to consider the original class priors and adjust the precision [16]. The adjusted precision is significantly lower than the plain precision. We will thus focus on the first 10 configurations with the highest adjusted precision (as the other metrics are generally high for any configuration we tested).

[1] `https://genome.jgi.doe.gov/Armost1/Armost1.home.html`

See Table 7.1 for results for the *donor* splice site classification. All ten configurations have the regularization constant $C = 1$. We can observe that higher order d does not necessarily imply better performance. Another observation is that the window size W_R tends to be rather high; 8 of the 10 best configurations have $W_R \geq 80$.

C	d	W_L	W_R	Acc [%]	Pr [%]	Pr_{adj} [%]	Rec [%]
1	15	100	80	95.502	91.518	44.735	88.633
1	15	60	100	95.461	91.446	44.505	88.522
1	25	100	60	95.454	91.417	44.415	88.522
1	20	80	80	95.428	91.374	44.279	88.449
1	20	60	60	95.460	91.253	43.903	88.744
1	25	60	80	95.444	91.241	43.868	88.678
1	25	80	100	95.424	91.228	43.828	88.597
1	20	80	100	95.385	91.221	43.804	88.419
1	25	60	100	95.358	91.174	43.662	88.344
1	25	100	80	95.354	91.158	43.611	88.347

Table 7.1. The best results for donor model achieved with subsampling of the negative examples. *Acc* is accuracy, *Pr* is precision, and *Rec* is recall.

Table 7.2 presents 10 best configurations for the *acceptor* splice site classification. The regularization constant is $C = 1$ in all the configurations, which is what we observed in the donors too. Higher degree d can improve performance (compare the second and the third configuration), but it can also cause performance degradation if it is too high (see the second and the sixth configuration). The window preference is the opposite to the donors; here the W_L tends to be greater.

C	d	W_L	W_R	Acc [%]	Pr [%]	Pr_{adj} [%]	Rec [%]
1	20	100	100	94.181	89.777	39.714	82.848
1	20	80	60	94.329	89.705	39.527	83.696
1	15	80	60	94.296	89.625	39.323	83.622
1	25	100	80	94.174	89.593	39.241	83.028
1	25	60	100	94.130	89.500	39.003	82.907
1	25	80	60	94.249	89.498	38.999	83.524
1	20	80	80	94.204	89.488	38.973	83.306
1	20	60	60	94.312	89.483	38.962	83.870
1	15	100	60	94.195	89.466	38.917	83.288
1	25	100	60	94.135	89.462	38.908	82.981

Table 7.2. The best results for acceptor model achieved with subsampling of the negative examples. *Acc* is accuracy, *Pr* is precision, and *Rec* is recall.

Both the donor and acceptor classification is a difficult task since only a small fraction of GT or AG are true splice site. A trade-off between precision and recall emerges from this fact. Either we discover only a few introns, which results in bad recall, or we produce many false positives and therefore achieve low (adjusted) precision. *The class priors affect the number of positively classified examples*: the more substantial subsampling of the negative class is performed, the more positives are classified.

Compared to the acceptors, we accomplish better results for the donors, perhaps due to AG being more frequent than GT in general (see Section 4.2.4).

Notice that accuracy is over 94% which, in connection with rather low precision, means that the models perform very well in the classification of the negative examples that are in the majority.

Another interesting observation is, that the window sizes W_L, W_R tend to be longer on the side of splice sites where an intron is: for donors, it is W_R, and for acceptors W_L. It means that given a splice site, the models rather consider what is inside of an intron that succeeds (or precedes) the splice site than what is before (or after) that intron.

7.2.3 Different regularization constants

For the experiments of the approach of different regularization constants C^+ and C^-, we set the rest of parameters to $d = 15, W_L = 60$, and $W_R = 100$. We also fixed the negative regularization constant $C^- = 1$, and then tried various values of C^+—see Table 7.3.

We experimented with the donor splice sites only. As the previous section showed, the task of donor classification is easier than the acceptor one, and it is sufficient to explore only one of the task in order to compare the two different approaches of dealing with class imbalances.

The donor dataset consisted of 94 613 positive and 3 185 820 negative examples, that is 3 280 433 examples in total. We used 85% examples for testing, which leaves 15% for training. It is an unusual split ratio which we chose, however, in order to keep a reasonable execution time of the SVM training (and 15% of the dataset is still almost half a million examples).

Table 7.3 displays performance of a models with different values of C^+. This approach led to substantial improvement in precision, however, in exchange for deterioration of recall. The value of C^+ does not affect the outcome very much as all the experiments arrive at more or less the same performance.

Note that since we did not modify the class priors, we do not compute adjusted precision in this case.

C^+	Acc [%]	Pr [%]	Rec [%]
10	98.351	74.290	65.499
20	98.350	74.343	65.350
50	98.351	74.246	65.580
100	98.342	74.126	65.316
200	98.355	74.449	65.422
500	98.349	74.283	65.386
1000	98.346	74.257	65.279

Table 7.3. Results for donor model achieved with different regularization constants. *Acc* is accuracy, *Pr* is precision, and *Rec* is recall.

7.3 Generalization

The previous sections showed how a model, trained on a single organism (*Armost1*), performs on sequences of the very same organism. In this section, we explore how the performance develops if we train a model on a whole phylum of fungi, and then test it on a class, an order, a family, a genus, or an organism that belongs to the phylum. Table 7.4 contains the results. Description of how we computed the results and discussion on them follows after the table.

phylum	class	order	family	genus	species
Ascomycota^{521}	Eurotiomycetes^{143}	Eurotiales^{109}	Aspergillaceae^{101}	Aspergillus^{81}	Aspwe1^{1}
40.39/74.54	45.19/80.55	48.22/81.72	47.35/81.80	47.49/81.74	49.64/84.85
47.16/62.56	49.52/69.02	50.11/71.11	52.51/70.89	51.64/70.84	57.48/74.75
Basidiomycota^{317}	Agaricomycetes^{239}	Agaricales^{79}	Mycenaceae^{18}	Mycena^{16}	Mycalb1^{1}
46.41/85.78	46.81/86.18	45.76/84.51	43.73/81.30	42.13/80.73	42.65/80.07
37.77/75.05	39.87/77.42	39.94/76.91	40.32/75.67	39.82/75.60	39.45/74.33
Blastocladiomycota^{4}	Blastocladiomycetes^{3}	Blastocladiales^{3}	Blastocladiaceae^{2}	Allomyces^{1}	Allma1^{1}
51.26/84.81	53.36/79.70	52.67/79.66	57.59/86.12	59.64/85.32	60.43/85.70
41.69/78.09	48.51/76.89	48.24/76.67	54.88/84.02	56.89/83.37	56.32/83.61
Chytridiomycota^{22}	Chytridiomycetes^{14}	Chytridiales^{5}	Chytriomycetaceae^{4}	Chytriomyces^{1}	Chytri1^{1}
36.06/73.93	36.31/74.85	38.64/74.81	40.48/78.47	40.90/84.83	40.98/84.56
29.48/61.17	30.54/62.25	35.23/65.68	36.21/68.13	35.51/73.22	34.93/74.33
Cryptomycota^{2}					Rozal1_1^{1}
47.06/83.47					42.86/82.71
51.26/76.56					48.27/72.22
Microsporidia^{8}					Enche1^{1}
55.88/90.48					00.00/00.00
70.59/57.14					00.00/00.00
Mucoromycota^{50}	Mucoromycetes^{36}	Mucorales^{36}	Lichtheimiaceae^{7}	Lichtheimia^{2}	Liccor1^{1}
43.84/83.96	46.13/85.81	46.18/86.04	47.02/86.29	63.18/93.05	62.37/92.92
37.80/74.56	41.65/77.84	41.23/77.92	40.99/78.16	55.80/87.76	51.61/85.99
Zoopagomycota^{16}	Kickxellomycetes^{6}	Kickxellales^{6}	Kickxellaceae^{6}	Coemansia^{3}	Coere1^{1}
41.73/64.23	30.53/29.78	28.69/29.50	29.53/29.39	28.40/46.43	21.71/47.44
41.75/48.67	18.62/22.09	18.49/21.64	18.94/21.69	16.89/38.20	16.08/39.21

Table 7.4. The table consists of 8 "multirows", each of which shows classification performance on different taxonomic ranks (columns) belonging to the phylum specified in the first column. Each cell states the name of a given taxonomic rank and performance on donors (1st row) and acceptors (2nd row). The performance is displayed as precision/recall, e.g., 74.56 is the recall for the acceptors of phylum *Mucoromycota*. Superscripts contain the number of organisms in each taxonomic rank.

7.3.1 Method

To perform the generalization experiments, we used the parameters that we found in Section 7.2.2 to be optimal for *Armost1*. One can, of course, expect that they will not be optimal for all organisms but finding the optimal parameters for each dataset would be extremely time-consuming as it would require to perform a grid search as in Section 7.2.2.

We trained a model for each phylum on a dataset consisting of splice site windows that were randomly sampled from *positive strands* of all fungi that belong to the phylum. Table 7.5 displays sizes of the datasets.

	donors		acceptors	
	positive	negative	positive	negative
Ascomycota	22010	225647	39982	228868
Basidiomycota	49159	198868	94237	202835
Blastocladiomycota	35675	213710	77378	203848
Chytridiomycota	39165	210557	75892	212051
Cryptomycota	18096	51535	59589	60965
Microsporidia	52	49267	104	64587
Mucoromycota	39787	206312	83044	208477
Zoopagomycota	25614	223840	48860	225567

Table 7.5. Number of positive and negative examples that were used to train models of the phyla.

Then, given a model trained on a phylum P, we tested the model on a class C that belongs to phylum P, order O that belongs to class C, and so on. For example, a model trained on phylum *Ascomycota* was tested on class *Eurotiomycetes*, order *Eurotiales*, family *Aspergillaceae*, genus *Aspergillus*, and species *Aspwe1*. We therefore test how a general model behaves on more specific data.

The taxonomic ranks (except for species) were chosen deterministically: always the one with the highest number of members. We started with the phyla which were given. Then for each phylum, we picked a class that belongs to the phylum and has the highest number of fungi, and so on up to species which always has a single member, so we picked a random one.

7.3.2 Discussion on the results

We omitted accuracies from the table in order to fit the table on a single page, but all of them were above 90% (multiple above 95%). The high accuracies are achieved by correct classification of the negative examples as we have already seen before. However, let us focus more on the positive examples, therefore on precision and recall.

To get a better overview of the results, see Figure 7.1 which contains line plots representing the development of precision and recall with respect to the taxonomic ranks for both the donors and acceptors.

The general phylum models of both the donors and acceptors achieved better precision and recall on more specific data (i.e., on lower level taxonomic ranks) for phyla

- *Ascomycota*,
- *Blastocladiomycota*,
- *Chytridiomycota*,
- and *Mucoromycota*.

In the case of phylum *Basidiomycota*, the performance of the donor model slightly decreases for more specific data, but the acceptor model maintains its performance. On the contrary, we can observe significant deterioration in the performance of the models of phylum *Zoopagomycota*.

The improvements in both precision and recall of multiple phylum models lead to a hypothesis that the trained phylum models describe the major groups of fungi within the phyla very well. Furthermore, the used parameters, that follow from the experiments on *Armost1* (see Section 7.2.2), seem like a reasonable choice either for the phylum models.

However, the unsatisfactory results of the *Zoopagomycota* model suggest the opposite. The parameters we used are probably unfit for this phylum and the underlying fungi. Also the sequences within this phylum might hold different properties than the other phyla which would require further analysis.

Notice that the results correspond with the differences in intron lengths in each phylum—see Table 4.2. The models fail on *Zoopagomycota*, species of which have the longest introns. Perhaps the window sizes should be adjusted for this phylum.

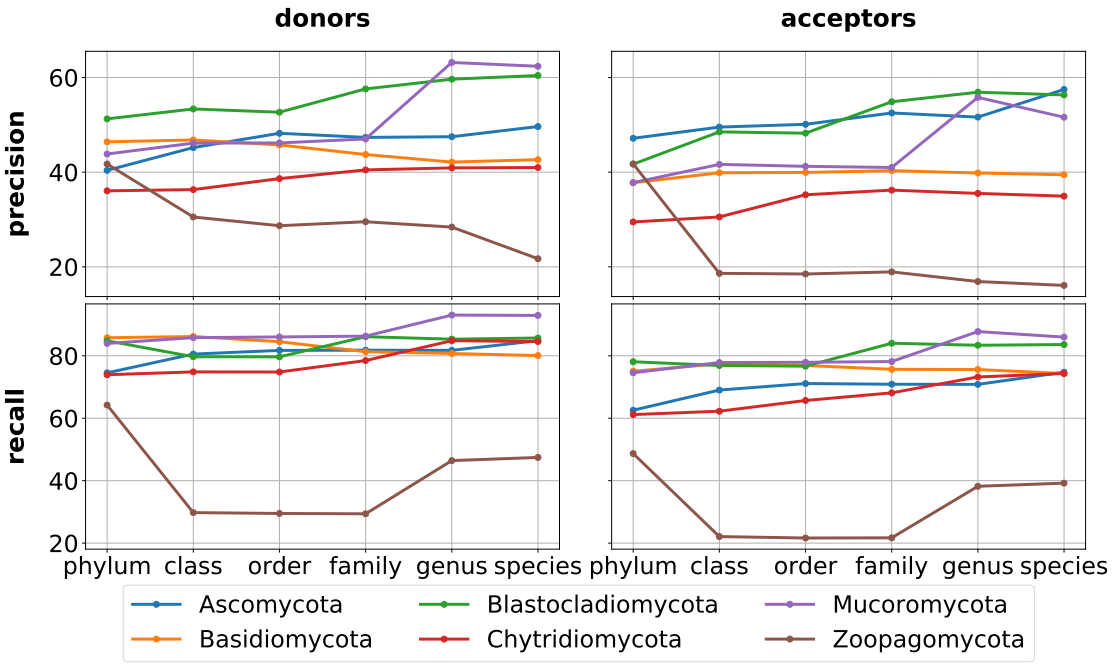

Figure 7.1. Precision and recall of donor and acceptor splice site classification. The figure does not contain results for model of phyla Cryptomycota and Microsporidia because they do not belong to any class, order, family, or genus.

7.4 Single model

The previous section showed that when applied on the respective fungi, the phylum models, apart from *Zoopagomycota*, perform comparably to the *Armost1* model. However, it would be more convenient to have just just a single model for both the donors and acceptors.

We performed experiments similar to the above but with only two models, one for donors and another for acceptors, which we trained using randomly sampled windows of all fungi. Given results of the experiments, we computed arithmetic differences of the current results and the previous results (i.e., a model per phylum). Figure 7.2 contains

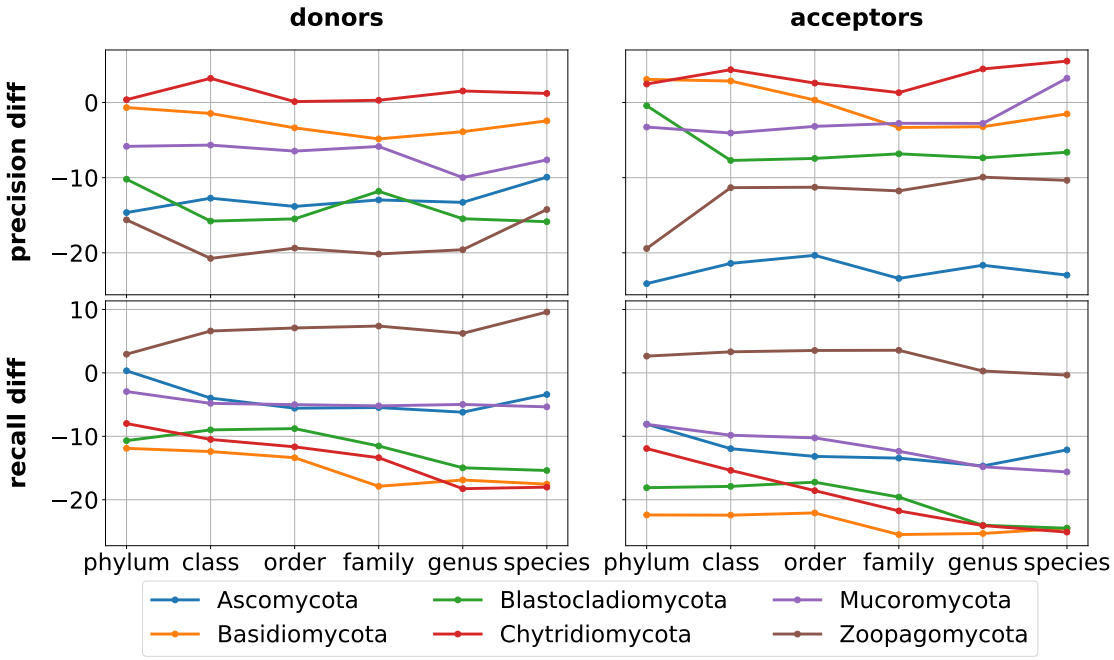

Figure 7.2. Precision and recall of donor and acceptor splice site classification for universal models. The figure does not contain results for model of phyla Cryptomycota and Microsporidia because they do not belong to any class, order, family, or genus. The differences are in percentage points.

plots representing the differences *current* − *previous*; positive values, therefore, mean improvement and negative the opposite.

At most of the domain, there was a deterioration for both the donors and acceptors. Some differences go up to -20 percentage points, which is a significant decrease. We thus must conclude that we could not accomplish a universal model able to classify donors nor acceptors of all the fungi with a consistent performance.

7.5 Execution time

The execution time of the splice site classification depends heavily on implementation of the SVM and the kernel. The library we used, *Shogun*, provides efficient implementations. It is difficult to express the time required to classify the splice sites because the implementation employs parallel execution. However, it is possible to make a rough estimate based on the performed classifications.

There was a classification of

- 1 236 958 donor splice sites that took 104 minutes to complete on 8 CPUs,
- and 1 962 935 acceptor splice sites that took 158 minutes to complete on 8 CPUs.

A simple calculation $n\_examples/time/n\_cpus$ gives 1 487 examples/minute for the donor model, and 1 553 examples/minute for the acceptor model.

Of course, the parameters of the SVM and kernels play an important role in the execution times. There would be longer executions for wider windows or higher degrees.

Chapter 8
Intron classification

Following the classification process, we established in Section 5.1, we have so far tackled only the problem of the splice site classification in detail. The second task is to generate pairs of positively classified complementary splice sites. We already discussed this task in Section 5.3.1 where we also included pseudocode of an algorithm that deals with the task entirely. The implementation is trivial, so we continue with the next task—the intron classification.

This chapter provides a code snippet of a Python script we used to train the intron classification models. We also give an insight into parameter selection and performance of models when applied on the whole phyla.

8.1 Script

We stick to the library (Shogun) we used for the splice site classification, however, we employ a different kernel function called spectrum kernel (see Section 5.4) for the intron classification. The script is very similar to the previous (Listing 7.1), it differs only in the creation of the kernel. See Listing 8.1 for a snippet.

```
import shogun as sg
import numpy as np

charfeat = sg.StringCharFeatures(sg.DNA)

charfeat.set_features(train_sequences)
train_features = sg.StringWordFeatures(charfeat.get_alphabet())
train_features.obtain_from_char(charfeat, order - 1, order, 0, False)

charfeat.set_features(test_sequences)
test_features = sg.StringWordFeatures(charfeat.get_alphabet())
test_features.obtain_from_char(charfeat, order - 1, order, 0, False)

preproc = sg.SortWordString()
preproc.init(train_features)
train_features = preproc.apply(train_features)
test_features = preproc.apply(test_features)

kernel = sg.CommWordStringKernel(train_features, train_features,
                                 False, cache_size)
```

Listing 8.1. A snippet of code used for training and evaluation of a splice site model.

Essentially, the features for this kernel are numeric vectors where each dimension represents a different k-mer. It is therefore required to precompute this features from the sequences first. Given the precomputed features, the rest of the process is the same

as in the case of splice sites: instantiate the kernel, pass it to a new SVM, train the model and use it to predict.

8.2 Parameter selection

For the intron classification, we use the spectrum kernel, which has only a single parameter, the order l. Another parameter is the regularization constant C of support vector machines. In this section, we show how each of these parameters affects model performance.

Similarly to the splice site classification, we start with a single, arbitrary organism; we picked *Kocim1* from phylum *Basidiomycota* this time. We used the *Armost1* models to classify all donor and acceptor splice site candidates present within the DNA sequences of the fungus. Based on the classification results, we created a dataset of intron candidates by pairing positively classified donors and acceptors (see Section 5.3.1).

The dataset consisted of 41 013 intron candidates, 8 815 of which were true introns and 32 198 were not. For testing, we used 40% of the dataset. The rest was used for training of models with

- the regularization constants $C^- = 1, C^+ \in \{1, 2, 3, 4\}$,
- and the order $l \in \{1, 2, 3, 4, 5\}$.

The output of the intron classification is the output of the whole process. Therefore, we aim to maximize the classification recall. Table 8.1 contains first 10 configurations with the highest recall. Also, the classes are not significantly unbalanced in this case, and thus we display accuracies too. In the table, we can observe the the trade-off between precision and recall. The first configuration, which achieves the highest recall, has one of the lowest precisions among the configurations. The highest accuracy is achieved using configuration $C^+ = 1, l = 5$ which however has one of the lowest recall.

C^+	l	Acc [%]	Pr [%]	Rec [%]
4	4	80.330	52.293	96.682
4	3	78.861	50.429	96.653
3	3	79.587	51.334	96.568
4	5	82.208	55.007	94.555
3	4	82.427	55.435	92.995
3	5	83.427	57.135	91.634
2	4	85.170	60.958	86.217
2	5	85.511	62.047	83.919
1	5	85.969	67.220	67.754
1	4	85.475	66.532	65.230
1	3	83.457	63.579	53.914

Table 8.1. The results of trained models with the best recall. *Acc* is accuracy, *Pr* is precision, and *Rec* is recall.

8.3 Performance on phyla

The approach (especially the spectrum kernel) relies on statistical differences in k-mer distributions between intron and non-intron sequences. We suppose that the statistical properties are preserved among the individual phyla. In this section we attempt to test the assumption.

8.3.1 Method

Since the input of intron classifiers is always based on the output of the splice site classifiers, we also train the intron models in this manner. Therefore, to train and test an intron model for each phylum (there are eight of them), it is required to classify eight datasets using splice site models of the individual phyla beforehand.

Of course, in order to use a splice site model, one needs to train it first, and since the training is a time-consuming task, we will use the models trained and described in Section 7.3.

We will utilize the fact that the splice site models were trained on *positive* strands only. If we restrict ourselves to negative strands only, we can freely use any splice site candidate of any fungi without having to worry about picking a candidate that was used during the training of the splice site models.

We, furthermore, wish to test the whole classification process on one fungus of each phylum (see Chapter 9). It would be practical for the tests to use the models we are about to train for the intron classification experiments. Thus, when creating datasets for these experiments, we will skip one fungus for each phylum and that fungus will be later used for the overall tests.

The data for this task are more complicated to obtain than for the splice site classification, because to obtain intron candidates we need all splice site candidates from a continuous DNA sequence. To accomplish this, we used algorithm described in Listing 8.2.

```
for each phylum P
    pick a random fungus F
    until we have enough data do
          pick a fungus G, G != F
          randomly select a set S of N scaffolds of the fungus G
          for each scaffold T from the set S
              extract all splice site candidates from neg. strand of T
```

Listing 8.2. Pseudocode of an algorithm we used to extract splice site candidates. The fungi G were selected in alphabetical order. The parameter N was chosen with respect to the number of members in each phylum (greater N for smaller phyla).

As a result, we obtained eight datasets for eight phyla, which we subjected to the splice site classification. Then, by pairing the positively classified complementary splice sites, datasets for the intron classification were created. Table 8.2 specifies the sizes of the datasets.

phylum	positives	negatives	total
Ascomycota	4 547	16 249	20 796
Basidiomycota	20 313	177 536	197 849
Blastocladiomycota	5 936	43 088	49 024
Chytridiomycota	13 887	129 558	143 445
Cryptomycota	267	6 021	6 288
Microsporidia	17	3	20
Mucoromycota	14 670	98 955	113625
Zoopagomycota	5 200	29 141	34341

Table 8.2. Sizes of datasets that were used to train and test the intron classifiers.

For the experiment, we used 70% of the data for training and 30% for testing. All the models used $C^- = 1, C^+ = 4, l = 4$ which are the parameters of the model that achieved the highest recall before (see Table 8.1). See Table 8.3 for the results of the intron models.

phylum	Acc [%]	Pr [%]	Rec [%]
Ascomycota	78.073	49.923	94.648
Basidiomycota	85.656	41.425	95.914
Blastocladiomycota	82.282	39.822	90.623
Chytridiomycota	85.230	38.683	89.846
Cryptomycota	93.217	28.571	40.000
Microsporidia	83.333	83.333	100.000
Mucoromycota	83.428	43.250	90.843
Zoopagomycota	83.034	46.966	93.269

Table 8.3. Results of the intron classification. *Acc* is accuracy, *Pr* is precision, and *Rec* is recall. Figure 8.1 contains a schematic illustration of the results for *Basidiomycota*.

Notice that the model of *Microsporidia* phylum detected all the introns that were in the dataset. However, the phylum had only a few introns [tab:intron-class-datasets]. The model of *Cryptomycota* achieved the lowest precision and recall among the models. That is perhaps due to the ratio between positive and negative examples in the dataset (there is 30 times more negatives than positives).

Generally, the models achieve high recall but low precision, meaning they find the majority of true introns present among the candidates, but also give a lot of false positives.

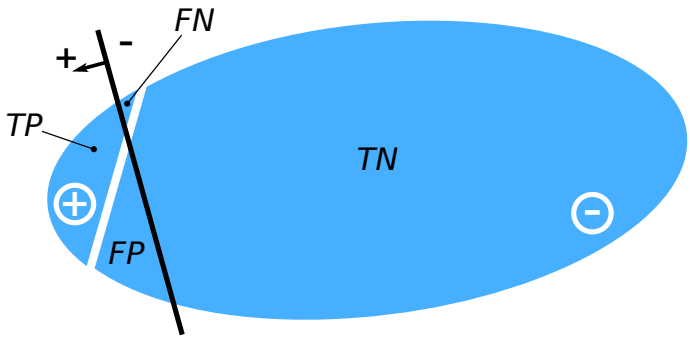

Figure 8.1. Schematic illustration of the intron classification results for *Basidiomycota* phylum. The blue bubble represents input dataset that contains examples of two classes, positive (denoted by the circled plus) and negative (denoted by the circled minus). The black line displays a virtual boundary between positively and negatively classified examples. Labels *TP, FP, TN, FN* denote areas representing true positives, false positives, true negatives, and false negatives, respectively. The proportions correspond to the computed results from Table 8.3.

8.4 Execution time

Similarly to Section 7.5, we give the execution time for the intron model too. A classification of 225 832 intron candidates took 40 minutes to complete on 8 CPUs. Therefore, about 5 646 examples got classified per minute. Compared to the splice site models, the intron model is faster to evaluate.

Chapter 9
Complete classification

Now, each step of the classification process (see Figure 5.2) has been introduced. We have also presented individual results of each step. This chapter describes the results of the whole process when applied on several fungi.

9.1 Method

When creating the datasets for the intron classification experiments, we left out a single fungus for each phylum (see Listing 8.2). We thus have a set of species that can be used to test the whole classification process:

- *Aspwe1* (phylum *Ascomycota*)
- *Mycalb1* (phylum *Basidiomycota*)
- *Allma1* (phylum *Blastocladiomycota*)
- *Chytri1* (phylum *Chytridiomycota*)
- *Rozal1_1* (phylum *Cryptomycota*)
- *Enche1* (phylum *Microsporidia*)
- *Liccor1* (phylum *Mucoromycota*)
- *Coere1* (phylum *Zoopagomycota*)

The *negative* strands of these organisms have not been used for the training of the splice site models nor the intron models. We therefore considered the negative strand only, and applied the whole process.

1. Extract splice site candidates from (the negative strand of) all scaffolds,
2. classify the splice site candidates,
3. create intron candidates by pairing the positively classified complementary splice sited from the previous step,
4. classify the intron candidates,
5. output the positively classified intron candidates.

We used the phylum models from Section 7.3 for the splice site classification, and the phylum models from Section 8.3 for the intron classification.

9.2 Results

Table 9.1 how many introns are on the negative strand of a species in total, and how many we detected.

The performance varies for different species. However, a substantial loss of the introns usually happened before the intron classification. Therefore, we can state that the performance of splice site models influences the overall performance to a great extent. Thus the performance could be improved by more involved analysis of the parameters

species	total	potential	found
Aspwe1	13 168	12 874	12 153 (92.29%)
Mycalb1	89 312	84 454	73 453 (82.24%)
Allma1	23 162	20 693	17 949 (77.49%)
Chytri1	23 259	13 893	11 797 (50.72%)
Rozal1_1	10 424	3 776	2 332 (22.37%)
Enche1	16	7	7 (43.75%)
Liccor1	26 632	24 182	21 712 (81.53%)
Coere1	1 903	1 685	644 (33.84%)

Table 9.1. Results of the classification process when applied on a whole species. Column *total* contains the total number of introns that are present on the negative strand for each species, column *potential* shows how many true introns were among the intron candidates before the intron classification, and the last column presents the number of true positives returned by the intron classifiers (i.e. the final number of correctly detected introns).

of the splice site model for each phylum, or by more specific definition of the target domain (e.g., only some classes of fungi).

Since the overall performance depends heavily on the performance of the splice site models, the worst results were achieved for the species that belong to the "unusual" phyla in terms of intron lengths (see Table 4.2).

Chapter 10
Conclusion

The goal of the thesis was to implement a solution able to detect introns present in the provided DNA of fungi. Based on the biological background and statistical properties of the data, a solution consisting of a combination of multiple-purpose support vector machines was designed. Although there were applications of machine learning to detect splice sites already, we built a complete pipeline that can identify introns.

The solution employs two SVM models for the donor and acceptor splice site classification. These models, however, tend to produce many false positives, therefore another SVM, the intron model, was added.

For each of the SVM models, a grid search through sets of reasonable parameters was performed. The grid search results showed how each parameter influences the performance of the models. For the splice site models, the window sizes were especially relevant.

As it turned out, a single donor (or acceptor) model is too general and thus achieves poor performance only. Therefore one donor (or acceptor) model per phylum was suggested.

Despite being trained using the same parameters, the models achieved decent results. For specific fungi, the classification process detected more than 80% of all introns.

10.1 Future work

Using a different set of parameters for models of different phyla promises further improvements in the classification performance. However, since the grid search through parameters demands much time and computational resources, it is left for future work.

As mentioned, an intron is always between two exons. This fact has not been directly used in the thesis (although it might have been used through the WD kernels). Introducing an exon model that would classify the neighboring sequences of each intron could bring up new kind of information to the process.

Introns are located in genes. Genes, however, form only a small fraction of the DNA sequences, the rest is non-coding DNA. It would be useful to preprocess the input DNA sequences and remove all the non-coding DNA. This should reduce the number of false positives coming from the splice site classification. Perhaps a hidden Markov model could be used to locate the genic areas [48–49].

References

[1] A. Mashaghi, and A. Katan. A physicist's view of DNA. *arXiv preprint arXiv:1311.2545*. 2013.

[2] I. Miko, and L. LeJeune. *Essentials of genetics*. Cambridge, MA: NPG Education, 2009.

[3] A. Purcell. *DNA – Basic Biology*. `https://basicbiology.net/micro/genetics/dna`. 2016. Online; accessed February 2019.

[4] B. Alberts, A. Johnson, J. Lewis, P. Walter, M. Raff, and K. Roberts. *Molecular Biology of the Cell 4th Edition: International Student Edition*. `https://www.ncbi.nlm.nih.gov/books/NBK26850/figure/A756/`. 2002. Online; accessed February 2019.

[5] *What is a gene? – Genetics Home Reference – NIH*. `https://ghr.nlm.nih.gov/primer/basics/gene`. 2019. Online; accessed February 2019.

[6] J. Huang, T. Li, K. Chen, and J. Wu. An approach of encoding for prediction of splice sites using SVM. *Biochimie*. 2006.

[7] David M., Constantine J. A., and others. *Fungus*. `https://www.britannica.com/science/fungus`. 2019. Online; accessed February 2019.

[8] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support vector machines and kernels for computational biology. *PLoS computational biology*. 2008.

[9] S. Clancy. *RNA splicing: introns, exons and spliceosome*. `https://www.nature.com/scitable/topicpage/rna-splicing-introns-exons-and-spliceosome-12375`. 2008. Online; accessed February 2019.

[10] X. Xie. *Gene discovery*. `https://www.ics.uci.edu/~xhx/courses/CS284A/lectures/Lecture2and3.pdf`. Online; accessed April 2019.

[11] G. Rätsch, and S. Sonnenburg. Accurate Splice Site Detection for Caenorhabditis elegans. *Kernel methods in computational biology*. 2004.

[12] X. Zhang, C. A. Tolzmann, M. Melcher, B. J. Haas, M. J. Gardner, J. D. Smith, and J. E. Feagin. Branch point identification and sequence requirements for intron splicing in Plasmodium falciparum. *Eukaryotic cell*. 2011.

[13] M. Mohri. *Foundations of machine learning*. Cambridge, MA: MIT Press, 2012.

[14] V. Franc. *Support Vector Machines*. `https://cw.fel.cvut.cz/wiki/_media/courses/be4m33ssu/svm_ws18.pdf`. 2018. Online; accessed February 2019.

[15] S. Sonnenburg, G. Rätsch, and K. Rieck. Large scale learning with string kernels. 2007.

[16] J. Brabec, and L. Machlica. Bad practices in evaluation methodology relevant to class-imbalanced problems. *arXiv preprint arXiv:1812.01388*. 2018.

[17] M. G. Reese, F. H. Eeckman, D. Kulp, and D. Haussler. Improved splice site detection in Genie. *Journal of computational biology*. 1997.

[18] S. Brunak, J. Engelbrecht, and S. Knudsen. Prediction of human mRNA donor and acceptor sites from the DNA sequence. *Journal of molecular biology*. 1991.

[19] H. Ogura, . Agata, M. Xie, T. Odaka, and H. Furutani. A study of learning splice sites of DNA sequence by neural networks. *Computers in biology and medicine*. 1997.

[20] T. Cai, and Q. Peng. *Predicting the splice sites in DNA sequences using neural network based on complementary encoding method.* In: *International Conference on Neural Networks and Brain*. 2005.

[21] S. Sonnenburg, G. Rätsch, A. Jagota, and K. Müller. *New methods for splice site recognition.* In: *International Conference on Artificial Neural Networks*. 2002.

[22] M. Yamamura, and O. Gotoh. Detection of the splicing sites with kernel method approaches dealing with nucleotide doublets. *Genome Informatics*. 2003.

[23] P. K. Meher, T. K. Sahu, A. R. Rao, and S. D. Wahi. A computational approach for prediction of donor splice sites with improved accuracy. *Journal of theoretical biology*. 2016.

[24] S. Sonnenburg, G. Schweikert, P. Philips, J. Behr, and G. Rätsch. *Accurate splice site prediction using support vector machines.* In: *BMC bioinformatics*. 2007.

[25] S. Sonnenburg, G. Rätsch, and B. Schölkopf. *Large scale genomic sequence SVM classifiers.* In: *Proceedings of the 22nd international conference on Machine learning*. 2005.

[26] A. C. Lorena, and A. C. P. L. F. de Carvalho. *Human splice site identification with multiclass support vector machines and bagging*. 2003.

[27] S. Rampone. Recognition of splice junctions on DNA sequences by BRAIN learning algorithm. *Bioinformatics (Oxford, England)*. 1998.

[28] V. V. Solovyev, A. A. Salamov, and C. B. Lawrence. Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Research*. 1994.

[29] J. L. Segovia-Juarez, S. Colombano, and D. Kirschner. Identifying DNA splice sites using hypernetworks with artificial molecular evolution. *Biosystems*. 2007.

[30] S. W. Roy, and D. Penny. Intron length distributions and gene prediction. *Nucleic acids research*. 2007.

[31] A. T. Ivashchenko, and Sh. A. Atambaeva. Variation in lengths of introns and exons in genes of the Arabidopsis thaliana nuclear genome. *Russian Journal of Genetics*. 2004.

[32] D. M. Kupfer, S. D. Drabenstot, K. L. Buchanan, H. Lai, H. Zhu, D. W. Dyer, B. A. Roe, and J. W. Murphy. Introns and splicing elements of five diverse fungi. *Eukaryotic cell*. 2004.

[33] Sh. A. Atambayeva, V. A. Khailenko, and A. T. Ivashchenko. Intron and exon length variation in Arabidopsis, rice, nematode, and human. *Molecular biology*. 2008.

[34] E. V. Kriventseva, and M. S. Gelfand. Statistical analysis of the exon-intron structure of higher and lower eukaryote genes. *Journal of Biomolecular Structure and Dynamics*. 1999.

[35] A. Kalogeropoulos. Automatic intron detection in nuclear DNA sequences of Saccharomyces cerevisiae. *Yeast*. 1995.

[36] C. Yin. Representation of DNA sequences in genetic codon context with applications in exon and intron prediction. *Journal of bioinformatics and computational biology*. 2015.

[37] R. Gupta, A. Mittal, K. Singh, P. Bajpai, and S. Prakash. *A time series approach for identification of exons and introns.* In: *Information Technology,(ICIT 2007). 10th International Conference on*. 2007.

[38] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*. 2000.

[39] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of computational biology*. 2000.

[40] X. H. Zhang, K. A. Heller, I. Hefter, C. S. Leslie, and L. A. Chasin. Sequence information for the splicing of human pre-mRNA identified by support vector machine classification. *Genome Research*. 2003.

[41] G. Rätsch, S. Sonnenburg, and C. Schäfer. *Learning interpretable SVMs for biological sequence classification.* In: *BMC bioinformatics*. 2006.

[42] G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*. 2004.

[43] J. T. L. Wang, S. Rozen, B. A. Shapiro, D. Shasha, Z. Wang, and M. Yin. New techniques for DNA sequence classification. *Journal of Computational Biology*. 1999.

[44] D. Loewenstern, H. Hirsh, P. Yianilos, and M. Noordewier. DNA sequence classification using compression-based induction. *Center for Discrete Mathematics & Theoretical Computer Science*. 1995.

[45] M. K. Sakharkar, V. T. K. Chow, and P. Kangueane. Distributions of exons and introns in the human genome. *In silico biology*. 2004.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, and others. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011.

[47] S. Sonnenburg, and others. *Shogun 6.1.0*. 2017. `https://doi.org/10.5281/zenodo.1067840`.

[48] J. Henderson, S. Salzberg, and K. H. Fasman. Finding genes in DNA with a hidden Markov model. *Journal of Computational Biology*. 1997.

[49] M. Stanke, O. Schöffmann, B. Morgenstern, and S. Waack. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC bioinformatics*. 2006.

Appendix A
List of abbreviations

- ANN — artificial neural network
- AUC — area under the curve
- CDS — coding sequence
- CLI — command line interface
- DNA — deoxyribonucleic acid
- DNF — disjunctive normal form
- FN — false negative
- FP — false positive
- FPR — false positive ratio
- JGI — Joint Genome Institute
- ML — machine learning
- PR — precision-recall
- RNA — ribonucleic acid
- ROC — receiver operating characteristic
- SVM — support vector machine
- TN — true negative
- TP — true positive
- TPR — true positive ratio
- WD — weighted degree kernel
- WDS — weighted degree kernel with shifts

Appendix B
The probability of a sequence not containing AG

The question being asked is what is the probability that a DNA sequence of length n does not contain a dimer AG assuming that the nucleotides are uniformly distributed and independent of each other.

Let us define a recursive function $F(n)$ which returns the number of all DNA sequences of size n that do not contain any AG:

$$\begin{aligned} F(0) &= 1, \\ F(1) &= 4, \\ F(n) &= 4 \cdot F(n-1) - F(n-2). \end{aligned}$$

There is only a single, empty, sequence for $n = 0$. For $n = 1$, there are four sequences as there are four different nucleotides.

For a greater n, we find the value by computing $F(n-1)$, which is the number of all sequences of length $n-1$ not containing AG, and multiplying that number by 4 because we can append four possible nucleotides to each of the sequences. However, the resulting number also includes such sequences of length $n-1$ that end with an A, and appending a G to them would produce an AG. There are $F(n-2)$ of those sequences as there are $F(n-2)$ sequences not containg an AG to which we can append an AG and get a sequence of length n which ends with the AG. Therefore we subtract $F(n-2)$.

Since the total number of DNA sequences of size n is 4^n, the probability in question is equal to $P(n) = \frac{F(n)}{4^n}$.