Universität Duisburg-Essen
Fakultät für Ingenieurwissenschaften
Abteilung Informatik und Angewandte Kognitionswissenschaft
Lehrstuhl Intelligente Systeme

- Bachelor-Arbeit -
im Bachelor-Studiengang
Angewandte Informatik

Adaptive Kernel Methods for Sequence Classification in Bioinformatics

Heiko Strathmann

Gutachter:
Prof. Dr. J. Pauli
Prof. Dr. D. Hoffmann

Zeitraum: 07. Oktober 2010 bis 07. Januar 2011

This work describes usage of *string kernel* equipped *Support Vector Machines* (SVM) to perform classification of sequences of amino-acids.

Two kernels are compared, namely the Distant Segments kernel (Boisvert et al., 2008) and the Spectrum kernel (Leslie et al., 2002). A broad range of data is used to test them.

Since SVM do not cover selection of all of a classifier's parameters, in this work, two of them need to be determined by hand: regularization parameter and kernel parameter. Two methods for parameter search are compared: the widely used *grid-search* and a self-developed improvement, which is based on *bisection*.

Grid-search has no guarantees regarding its correctness. In contrast, it is shown that when certain requirements are met, along with being totally correct, the bisection based method reduces time complexity for searching for one parameter from linear (grid-search) to logarithmic. Empirical results suggest that the requirements for the introduced bisection based search are met in the context of sequence classification and search for regularization parameters.

The gained results show that the introduced overall approach generates high-quality classifiers. The worst obtained one still does $\sim 75\%$ correct classifications, best obtained classifiers are nearly perfect. The DS-kernel, with exceptions, performs slightly better than the Spectrum kernel.

In addition, an entropy based measure for a dataset's extend of disorder is introduced and examined for connections to classification severity. The observation is made that divergent datasets form less severe classification tasks.

The introduced approach leads to results, which are well usable in practice and therefore qualifies itself for further examination. In addition, many of the introduced techniques are also suitable for non-biological, sequence based data.

Contents

1. Introduction

1.1. Task description

The task of this work is to perform binary classification of sequence based, biological data, with use of kernel methods for supervised learning, namely Support Vector Machines equipped with string-kernels.

Classification here means to ascribe data to a known set of classes or categories. *Binary* classification induces two categories. In context of biological data, this means that data is categorized by *having* or *not having* a certain attribute. An example are viruses, which *are* or *are not* resistant to a certain drug.

To build a classifier, some kind of learning has to take place. The task of this work involves *supervised learning*, meaning that a set of already categorized data, a training set, is used to build a classifier. This classifier should on the one hand be able to correctly classify data in the training set, to *memorize* some of it, and on the other hand be able to correctly classify unknown data, which is similar to data in the training set, to *generalize*. These two abilities are in conflict with each other.

Support Vector Machines are a popular tool for classification tasks. The concept is based on a linear separation of representations of data in a dot-product space. Using a *kernel*, which induces the latter, they can be extended to directly work on any kind of data, like in this work: character sequences of different lengths.

Since the underlying dot-product space may be of a large dimension, linear separation in it corresponds to non-linear separation of input data and allows to work on highly complex data. With the use of *string-kernels*, non-linear classification may be performed on sequential data without the need of preprocessing it.

Data, which is provided along the task of this work, consists of binary labeled sequences of *amino-acids*, namely *polypeptides*, which are chains of between ten and one-hundred amino-acids, and larger chains, so called *proteins*. In the following, it is motivated why a sequence based representation of biological data may be used for classification.

1.2. Biological background and kernel methods

The "central dogma" in molecular biology is that information of an organism is stored in its genome, namely the *DNA*. DNA consists of four different types of molecules called *nucleotides* from which sequences are built. These sequences encode the "construction

plan" of an organism. When a gene is read by a cell, DNA is transcribed into *RNA*, which is a similar copy of the original molecules. Then, this RNA is fed into a structure called *ribosome*, which translates the RNA-sequence to a sequence of amino-acids, a protein or polypeptide. There are 20 amino-acids and therefore, they can easily be coded into letters. A list of all amino-acids and their letter representation can be found in appendix C.

Proteins are produced in cells, which are the smallest independent parts of an organism. Every protein has a different function and can be seen as a tool, which the cell produces to accomplish a certain task. For example, *enzymes* are proteins, which catalize biochemical reactions, for example in context of food processing in the human body.

The three-dimensional structure is most important regarding function of a protein, however, this structure is determined by the underlying sequence of amino-acids. The "central paradigm" in bioinformatics states that similar sequence leads to similar molecular structure, which itself leads to similar function.

The motivation to work on sequences of amino-acids comes from this paradigm. It is expected to determine function of yet *unknown* sequences by examining sequences with *known* function. Applications are for example a gene test for HIV-patients to determine whether the virus is resistant to a certain drug, protein super-family detection or to determine if a protein binds to a certain type of cancer cells to build effective drugs.

Since capacities of databases and speed/quality of protein transcriptions are increasing enormously, the amount of available sequence data grows very fast in contrast to possibilities of determining its function manually. Computer aided processing is though an important tool to handle such masses of data. It is expected to derive new biological knowledge from experiments on available data.

Biological data is by nature very complex, making it hard to efficiently build models. In the mid 1990s kernel based learning methods were developed which were able to analyze nonlinear problems with the same efficiency, which was achievable with linear approaches. A concept, which was developed along and benefits from kernel methods is the concept of *Support Vector Machines*, a learning algorithm which offers a strong theoretical fundament for linear classification problems. They have empirically shown to be robust and powerful in many works. With use of kernel methods, strength of SVM learning becomes available for nonlinear problems. This results in the ability to efficiently perform pattern recognition on complex data.

Another advantage is that usage of kernels allows to process non-numerical data directly, without the need to transform it into a numerical representation before. This is especially useful when using input data of variant length, like strings.

The overall concept is a promising approach for handling biological data and is used since about ten years. New, finer methods for handling protein data are still developed and form an active field of research.

1.3. Literature Overview

Since the focus of this work is not on biology, only a brief and coarse introduction to it is given. All details, which are necessary for this work, can be found in textbooks, like for example (Merkl and Waack, 2009).

Support Vector Machines also have made their way into various textbooks. The book (Cristianini and Shawe-Taylor, 2000) gives an introduction to basic concepts, which come along. In (Schölkopf and Smola, 2001) some advanced concepts, like soft-margin classifiers are described. An analysis of statistical learning theory behind SVM can be found in (Vapnik, 2000).

Pattern recognition is a vast field in machine learning. In (Bishop et al., 2006), a detailed introduction can be found. (Shawe-Taylor and Cristianini, 2004) describes how to use kernel methods for pattern recognition both concerning theoretical concepts as concrete applications. (Schölkopf and Smola, 2001) also contains a large part about kernels.

The use of kernel methods for classification of sequence based, biological data began around ten years ago. The first used approach, the *Fisher*-Kernel (Jaakkola et al., 1999) computes a similarity measure of proteins and is based on prior knowledge, which is gained using a hidden Markov model.

String-kernels compute a similarity measure between two strings. Computing similarities of sequences has been studied for a long time in bioinformatics. On the base of bio-chemical similarities of single amino-acids, the *Smith-Waterman*-algorithm computes the optimal local alignment[1] of two amino-acid chains (Smith and Waterman, 1981). A kernel, which is based this algorithm is the *Local Alignment Kernel* (Saigo et al., 2004). Although knowledge of bio-chemical processes is included here, more general approaches, which do not take external knowledge into account, led to better results. Besides, this kernel has many parameters to select, which is a problem, as described later in this work.

A very general way of computing a similarity measure of two strings is the *Spectrum Kernel*, (Leslie et al., 2002) or (Shawe-Taylor and Cristianini, 2004, 347-351). It is a very simple and intuitive approach based on counting occurrences of subsequences of a certain length. It has shown to perform well in the context of protein classification. Its main advantage is that it is computable in linear time, which is a nice property when handling huge masses of data, as often the case in a biological context. It is noteworthy that *no* external biological knowledge is included. Therefore, it may be applicable for other problems dealing with sequence based data.

An extension to the Spectrum Kernel is the *Blended Spectrum Kernel*, which counts occurrences of sequences up to a certain length (Shawe-Taylor and Cristianini, 2004, 347-351).

[1] An alignment is a way of arranging two or more sequences to identify common parts. For details, see (Merkl and Waack, 2009)

In (Boisvert et al., 2008), an extension to the BS-kernel, the so called *Distant Segments Kernel* is introduced. It is also does not use any biological knowledge, but takes distances of subsequences into account, which is biological meaningful. According to this paper, the DS-kernel outperforms the BS-kernel and the Spectrum kernel and is the "state-of-the art" for certain types of problems.

There are also string-kernels, which do not work on sequences of characters, but on sequences of words, so called *word-based* kernels. These are for example used in computer vision (e.g. (Csurka et al., 2004)) and text categorization tasks (e.g. (Joachims, 1998)). Since the task of this work is to classify sequences of amino-acids, which are represented by single letters, word based kernels are not suitable.

1.4. Selected Approach and Placement

Methods, which are used in this work are mainly part of the task: SVM and string-kernels. As for SVM, the so-called soft margin C-SV classifier is used, which is a standard choice in literature. The kernel choice is partly based on results of (Boisvert et al., 2008), which imply that the DS-kernel is "state-of-the-art". To compare the impact of taking sequences *up to* a certain length into account to taking sequences *of* a certain length into account, the Spectrum kernel is compared to the DS-Kernel. Both kernels are also interesting from a non-biologist's point of view, because they might be applicable for other, non-biological problems, which are based on sequential data.

Both, the kernels and the SVM have one parameter to set manually. To do this, a *grid-search* is performed. The latter is also examined for possible improvements and an advanced method, which is based on bisection, is introduced and compared to the grid-search. To perform a search for best parameters, a classifier has to be evaluated and to be compared.

To evaluate a classifier, it has to be tested, especially for its generalization ability. In this work, the widespread technique of *cross-validation*, as for example described in (Hsu et al., 2003), is used. Techniques for averaging results of different cross-validation-folds are taken from (Forman and Scholz, 2009). *ROC*-graphs are a widely used tool for classifier comparison and are used on the base of (Fawcett, 2003). Other measures for comparing classifiers are standard choices, namely *Accuracy* and *F-measure*.

Software is implemented in the language R, using SHOGUN (Sonnenburg et al., 2006), which is a software framework for machine learning applications and, among others, uses the widespread LIBSVM (Chang and Lin, 2001) SVM implementation. There exist other machine learning packages for R, for example kernlab (Karatzoglou et al., 2004), which was used at first, until a bug, which caused software freezes hindered further usage. Kernel matrices are pre-calculated, since on-line evaluations of kernels take a lot of time in context of string kernels. Many available SVM implementations do not natively support strings as input data, or easy use of custom kernel matrices

(e.g. OpenCV[2], SVMLight[3] or pure LIBSVM), however, SHOGUN does.

Datasets are chosen in such way that a broad spectrum of attributes is covered: Simple and complex data, short and long sequences, small and large length variance, easy and severe classification tasks, etc. To avoid biased results, multiple types of classification tasks are included: Drug resistance, virus attributes/identification and protein super-family detection. Even though all datasets contain problems in context of bioinformatics, the focus is *not* on biological attributes of these data, but more on the general approach of classification of different kinds of data.

Although this work has a strong connection to bioinformatics, many of its results are not limited to this subject.

The concept of SVM is applied to many non-biological pattern recognition problems. The use of string-kernels is also not limited to bioinformatics. In particular the kernels, which are used in this work, are not based on external biological knowledge and may be applied to other problems, for example in context of computer-vision. Consequently, results regarding connections between attributes of datasets and classification performance may be re-used in other contexts. Evaluation of classifiers is a standard topic, so all methods and results, which are introduced are of generic value. Parameter search is another standard topic, which even forms an own area of research. The search-techniques which are described are usable in any SVM/kernel context.

The used datasets solely contain biological problems, so their individual results are a contribution to bioinformatics and may be used to compare the approach described in this work to other techniques.

1.5. Chapter Overview

Note: In the beginning of each chapter, a short summary of its content is given. Then all resources, which are used in the chapter are stated with reference to each section, to omit permanent citation during the text.

Chapter 1 contains the task description, background information, literature overview, the selected approach for solving the task and placement of this work.

Chapter 2 introduces theoretical preliminaries, which are used in this work. This includes a definition of classifiers, basics of statistical learning theory and risk minimization, the concept of support vector machines with focus on their construction and their margin, the kernel trick and a description of the two kernels, which are used.

Chapter 3 gives an overview of methods, which are used in this work. First, an overview of the approach of building a classifier is given. This is followed by the concept of stratified cross-validation. Then, used performance measures are introduced,

[2] http://opencv.willowgarage.co
[3] http://svmlight.joachims.org

namely Accuracy, F-Measure and AUC, and methods for decreasing experimental variance are introduced. The important subject of parameter selection starts with introduction and analysis of the grid-search. Then, a self-developed, bisection based technique for maximizing an unknown function, which is based on stronger assumptions and may be applied to parameter search, is described. Finally, a measure for disorder of a dataset, the sequence based entropy is described.

Chapter 4 starts with a description of performed experiments and used data. Then, results of the experiments are presented. These are interpreted regarding different performance measures, experimental variance reduction, attributes of used data and the parameter search. Then, results of selected datasets and kernels are compared. Finally, results of the bisection method for parameter search, are given.

Chapter 5 summarizes approach and results of this work and gives a perspective for further work.

Appendix A is a brief description of software, which is implemented. All used components and tools are briefly described.

Appendix B contains a proof of the introduced algorithm for bisection based parameter search, which was omitted in the main text.

2. Theoretical preliminaries

In this chapter, the basic theoretical concepts that are needed throughout this work are introduced. Section 2.1 starts with fundamental terms around *classification*, to have a formal setting. Then, basics of *statistical learning theory* and *risk minimization* are given, in particular details about *Empirical Risk Minimization* and the more important *Structural Risk Minimization*.

The latter forms the fundament of *Support Vector Machines*, which are described for binary, linear classification problems in section 2.2, with focus on their construction and the *margin*.

Section 2.3 describes the fundamentals of kernels, which extend the SVM concept to non-linear problems. The kernels, which are used in this work are described in detail, namely the *Spectrum Kernel* and the *Distant Segments Kernel*.

Most subjects in this chapter are based on external resources, except for some formal details. Note, that due to limited space, only a quick and brief introduction is given. For further details, see section 1.3 for literature.

Section 2.1.1 is a collection of definitions and terms that are used in the chapter. As for the rest of the statistical learning theory part in section 2.1, the text is mainly based on two resources. The beginning, section 2.1.2 and section 2.1.3 are a summary of (Vapnik, 2000, 17-22). The example in 2.1.3 is taken from (Schölkopf and Smola, 2001, 8). Section 2.1.4 comes from (Schölkopf and Smola, 2001, 9-11) and (Vapnik, 2000, 93-95).

Section 2.2, which introduces support vector machines, is a summary of the main parts of (Schölkopf and Smola, 2001, 189-205).

The introduction of kernels in section 2.3 states used resources. The formal setting for working with strings, which is given in section 2.3.1 is not based on external resources. The introduced kernels are taken from (Boisvert et al., 2008) and (Leslie et al., 2002). The formal description was extended at some points to be more accurate. Section 2.3.4 is not based on any external resources.

2.1. Statistical Learning Theory

2.1.1. Fundamental Terms

Let X be a set called *instance space* in which classification takes place. Its elements are called *patterns*. Let Y be a set containing elements that are referred to as *class labels*. Assuming that every $x \in X$ corresponds to some $y \in Y$, there exists a total function $s : X \to Y$ which maps every pattern in the instance space to a class label.

This function, called *supervisor*, is usually unknown and is to be estimated. For every $x \in X$, it holds $y := s(x)$. A *classifier* is a function $c : X \to Y$ which is wanted to approximate s. A classifier behaves *perfect* if for any $x \in X$, it holds $c(x) = s(x)$.

In the context of this work, X consists of strings over a given finite alphabet. The classification problem is binary, meaning that $|Y| = 2$.

Let $T = \{(x_i, y_i) \mid (x_i, y_i) \in X \times Y \text{ and } y_i = s(x_i)\}$ be a *training set* of m examples $(1 \leq i \leq m)$. A *learning algorithm* is an algorithm that outputs a classifier after reading the training set. This is a *supervised learning* situation.

Learned classifiers should be able to *generalize*. This means that given a yet unseen pattern $x \in X$, a classifier c should predict the corresponding label $y = c(x)$ in such way that $(x, c(x))$ is in some sense similar to the examples in the training set.

Let each x_i of all (x_i, y_i) in the training set be randomly and independently drawn according to a fixed, but unknown probability distribution function $F(x)$. Let the output of the supervisor s be according to a conditional function $F(y|x)$, which is fixed but unknown. Let the training set consist of m independent and identically distributed (i.i.d.) observations drawn according to $F(x, y) = F(x)F(y|x)$.

A *learning machine* implements a set of functions $f(x, \alpha), \alpha \in \Lambda$, where Λ is a set of parameters. A classifier may be derived from this set. The problem of learning is that of choosing from the given set of functions $f(x, \alpha)$, the one, which best approximates the output of the supervisor function.

2.1.2. Risk Minimization

The problem of choosing from a set of functions the one, which is best suited for a certain problem is closely related to risk minimization. It is intuitive to choose the function with minimal erroneous results in classification.

Given an element $x \in X$, a *loss function* $L(y, f(x, \alpha))$ measures the *loss*, or *discrepancy* between output of the supervisor and $f(x)$. In the binary case, with $Y = \{-1, 1\}$ and $f(x, \alpha) \in Y$, the loss function is given by

$$L(y, f(x, \alpha)) = \begin{cases} 0 & \text{if } y = f(x, \alpha), \\ 1 & \text{if } y \neq f(x, \alpha). \end{cases}$$

The expected value of the loss, given by

$$R(\alpha) = \int L(y, f(x, \alpha)) dF(x, y),$$

is called the the *risk functional* ot *actual risk*. Its value equals the probability of misclassification. The goal is to find the function f and the parameter α that minimize $R(\alpha)$. The only information to do this is in the training set.

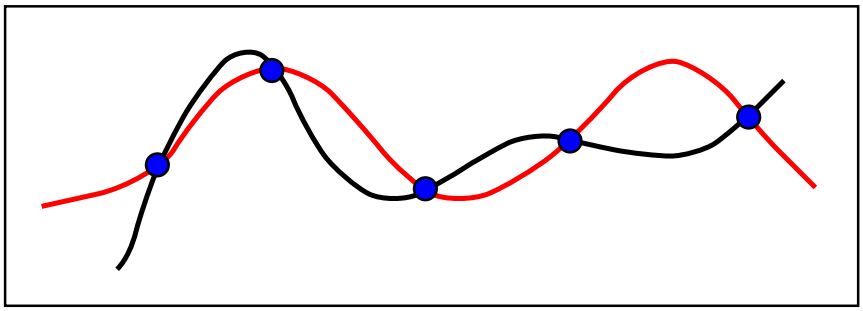


Figure 2.1.: Minimizing the empirical risk may fail: Both curves do perfectly explain the training data (blue points), but lead to different models.

2.1.3. Empirical Risk Minimization (ERM)

It is not possible to calculate $R(\alpha)$, if the underlying probability distribution function $F(x, y)$ is unknown. However, it is still possible to calculate the *empirical risk functional*

$$R_{emp}(\alpha) = \sum_{i=1}^{m} L(y_i, f(x_i, \alpha)), \tag{2.1}$$

which is independent of $F(x, y)$.

The empirical risk converges to the actual risk for $m \to \infty$. Unfortunately the number training patterns is limited, so trying to minimize $R(\alpha)$ by minimizing $R_{emp}(\alpha)$ might fail. There are cases where a classifier with a low empirical risk is derived from the training set, but does not behave well on other, unknown data. This is called *overfitting*. More formal this means that for any function (or classifier) $f : X \to Y$ and any test set $(\bar{x}_1, \bar{y}_1), ..., (\bar{x}_{\bar{m}}, \bar{y}_{\bar{m}}) \in X \times Y$ of size $\bar{m}$, satisfying $\{\bar{x}_1, ..., \bar{x}_{\bar{m}}\} \cap \{x_1, ..., x_m\} = \emptyset$, there exists another function f^* such that $f^*(x_i) = f(x_i)$, for all $i \in \{1, ..., m\}$, but $f^*(\bar{x}_i) \neq f(\bar{x}_i)$, for all $i \in \{1, ..., \bar{m}\}$. Figure 2.1 shows an example.

As only the training data is given, there is no chance of knowing which of the functions is preferable. To overcome this problem, the complexity of the functions is taken into account. This leads to the SRM-principle.

2.1.4. Structural Risk Minimization (SRM)

The basic idea of the SRM-principle is to have a trade-off between quality of approximation of training data and complexity of approximating classifier functions. In (Vapnik, 1998), it is shown that with probability $1 - \delta$, it holds that

$$R(\alpha) \leq R_{emp}(\alpha) + \phi(h, m, \delta), \tag{2.2}$$

where $\phi(h, m, \delta)$, called the *confidence term*, increases monotonically in h. h is the VC-dimension, which is a measure of capacity/complexity of a learning machine (see Vapnik (2000) for details).

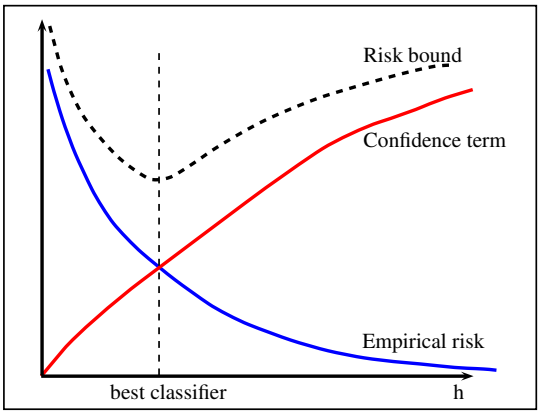


Figure 2.2.: The actual risk is bounded by the empirical risk and the confidence term.

The inequality (2.2) itself is not used to derive risk bounds in practice, but forms the theoretical fundament of the SVM method. In particular, it is well suited to explain the SRM-principle:

For $\phi(h, m, \delta)$ to be small, the complexity of a learning machine has to be kept small (in relation to m) since $\phi(h, m, \delta)$ increases monotonically in h. At the same time, to keep the empirical risk small, h must be large enough to provide a set of functions, which are able to model possibly complex hidden dependencies in $F(x, y)$. The empirical risk is monotonically decreasing in h, therefore the capacity of the learning machine increases with a larger h, and the machine can fit the data better. Given multiple learning machines, the one that leads to the lowest bound for the actual risk is chosen. Since the empirical risk monotonically decreases in h, the confidence term monotonically increases in h and the risk is bounded by the sum of these two, its minimum lies at their point of intersection. Figure 2.2 depicts this principle. *"The SRM principle defines a trade-off between the quality of the approximation of the given data and the complexity of the approximating function."* (Vapnik, 2000, 95).

2.2. Support Vector Machines

Support Vector Machines (SVM) are based on a learning algorithm, which implements structural risk minimization by keeping the empirical risk fixed and minimizing the confidence term. They are a result of statistical learning theory. The SVM method is based on the case of *linear separability* of data and then is extended to situations, in which data is not linearly separable. In the following, it is assumed that the training data has the form $T = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$ with each $(\mathbf{x}_i, y_i) \in X \times \{1, -1\}$ for $1 \leq i \leq m$ and $X \subseteq \mathcal{H}$ where $\mathcal{H}$ is a dot product space[1].

[1] A dot product space is a vector space endowed with a dot product.

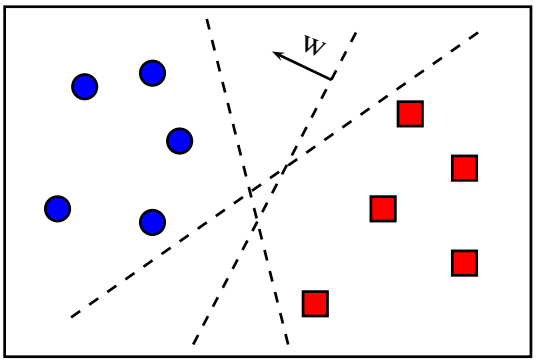


Figure 2.3.: Examples of possible linear separations of patterns.

2.2.1. Separating Hyperplanes

Two sets $U \subseteq \mathcal{H}$, and $V \subseteq \mathcal{H}$ are said to be *linear separable* if there exists a vector $\mathbf{w} \in H$ and a number $b \in \mathbb{R}$, so that for any $\mathbf{u} \in U$ and any $\mathbf{v} \in V$, it holds

$$\langle \mathbf{w}, \mathbf{u} \rangle \leq b < \langle \mathbf{w}, \mathbf{v} \rangle.^2 \tag{2.3}$$

If A and B are finite and linear separable, they are also *absolutely linear separable*, meaning that $\leq$ in the inequality becomes $<$ (see for example (Rojas and Feldman, 1996)). Figure 2.3 shows an example of linear separation.

Two sets of vectors in $\mathcal{H}$ that are linear separable can be separated by a $|\mathcal{H}|$-dimensional hyperplane. Analogous to (2.3), any hyperplane, which separates the data can be written as

$$\{\mathbf{x} \in \mathcal{H} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}, \quad \mathbf{w} \in \mathcal{H}, \quad b \in \mathbb{R}, \tag{2.4}$$

where $\mathbf{w}$ is a vector orthogonal to the hyperplane. If $\mathbf{w}$ has unit length, then $\langle \mathbf{w}, \mathbf{x} \rangle$ is the length of $\mathbf{x}$ along the direction of $\mathbf{w}$. In general, $\mathbf{w}$ is scaled by $||\mathbf{w}||$, where $||\mathbf{w}||$ is the euclidean norm $\sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}$ of $\mathbf{w}$. $\mathbf{x}$ and b may be multiplied by the same non-zero constant without changing the hyperplane. Using this, a hyperplane may be brought to a standard form. Given a set of vectors $\mathbf{x}_1, ..., \mathbf{x}_m \in X$, the pair $(\mathbf{w}, b) \in \mathcal{H} \times \mathbb{R}$ is called *canonical* form of the hyperplane (2.4) with respect to $\mathbf{x}_1, ..., \mathbf{x}_m$, if it is scaled such that

$$\min_{i=1,...,m} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1. \tag{2.5}$$

Given a canonical form of a hyperplane, which separates training data $(x_1, y_1), ..., (x_m, y_m) \in X \times \{1, -1\}$, it holds that

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1.$$

A hyperplane corresponds to the *decision function* or *classifier*, given by

$$f_{\mathbf{w},b} : \mathcal{X} \to \{1, -1\} \quad \text{with} \quad f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b). \tag{2.6}$$

[2] $\langle \mathbf{w}, \mathbf{x} \rangle$ is the dot product $\sum_{i=1}^{|\mathcal{H}|} w_i x_i$ (with w_i and x_i being the i-th component) of $\mathbf{w}$ and $\mathbf{x}$.

2.2.2. The Role of the Margin and Optimal Margin Hyperplanes

Given a hyperplane, which is similar to (2.4)

$$\rho_{(\mathbf{w},b)} := \frac{y(\langle \mathbf{w}, \mathbf{x} \rangle)}{||\mathbf{w}||} \tag{2.7}$$

is called the *geometrical margin* of $\mathbf{x}$. The minimum value

$$\min_{i=1,...,m} \rho_{(\mathbf{w},b)}(\mathbf{x}_i, y_i) \tag{2.8}$$

is called the (geometrical) margin of training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$.

In (Vapnik, 2000), a theorem is shown, which explains why a large margin is desirable. Given training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ with all $\mathbf{x}_i \in \mathcal{H}$ being situated in a sphere of radius of R and a set of hyperplane classifiers (2.6) with the margin Δ, the classifiers VC-dimension h is bounded by the following inequality

$$h \leq \min \left(\lfloor \frac{R^2}{\Delta^2} \rfloor, |\mathcal{H}| \right) + 1.$$

Consequently, a large margin leads to a small confidence term in the risk bound (2.2). Given a canonical form of a hyperplane that separates some training data, along with (2.5), (2.10), (2.7) and (2.8), it follows that the geometrical margin of the training data is $\frac{1}{||\mathbf{w}||}$. Therefore, maximizing the margin is equivalent to minimizing $||\mathbf{w}||$.

The theorem also gives a finite bound for the VC-dimension of a set of classifiers in an infinite dimensional dot-product space, which is important for classification in such a space.

A hyperplane that generalizes well can be constructed by solving the following minimization problem

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{minimize}} \frac{1}{2}||\mathbf{w}||^2, \tag{2.9}$$

subject to

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1, \tag{2.10}$$

which is called the *primal optimization problem.*

The problem may be solved using the *Lagrangian*

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i (y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1), \tag{2.11}$$

where $\boldsymbol{\alpha} = (\alpha_i, ..., \alpha_m)$ are called *Lagrange multipliers.* The Lagrangian must be maximized with respect to α_i, and minimized with respect to $\mathbf{w}$ and b. This corresponds to finding a saddle point. Consequently, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i y_i = 0, \tag{2.12}$$

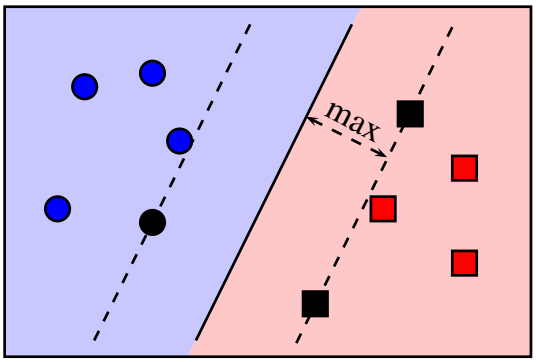


Figure 2.4.: Example for a linear classifier with maximal margin. The blackened patterns are the *support vectors* of the separating hyperplane.

and

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \Leftrightarrow \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{2.13}$$

According to the *Karush Kuhn Tucker conditions*, only Lagrange multipliers $\alpha_i \neq 0$, which are situated at the saddle point, correspond to the constrains (2.10). The patterns $\mathbf{x}_i$ for which $\alpha_i \geq 0$ are called *Support Vectors* and give the name to the method. Figure 2.4 shows an example of a linear separation with maximum margin.

Substituting (2.12) and (2.13) into (2.11) leads to the *dual form* of the primal optimization problem

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \, W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \tag{2.14}$$

subject to

$$\alpha_i \geq 0, \text{ for all } i \in \{1, ..., m\} \qquad \text{and} \qquad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{2.15}$$

Substituting (2.13) into the decision function (2.6) leads to

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right). \tag{2.16}$$

This is an expression which is evaluated by only using the dot product of the to be classified patterns and the Support Vectors.

2.2.3. Nonlinear Classification and Kernel Trick

The above introduced method fails if given training data is *not* linear separable. This problem can be overcome by introducing a high dimensional dot-product space called the *feature space*, mapping data into and perform classification in such a space. Any

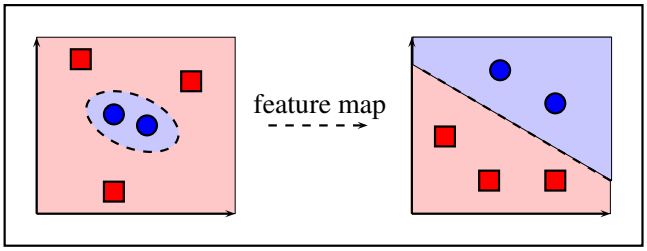


Figure 2.5.: The idea of kernel methods for classification: Usage of a feature map allows linear classification of non-linear data in a higher dimensional feature space. Left is the input space X, right is the feature space $\mathcal{F}$.

data is linear separable if an appropriate high dimension of such feature space is chosen.

Let $\mathcal{F}$ be such a high dimensional dot-product space. The *feature map*, given by

$$\Phi : X \to \mathcal{F},$$

transforms every element $\mathbf{x}_i$ of the training data into its so-called *feature vector* $\Phi(\mathbf{x}_i)$ in $\mathcal{F}$.

The optimization problem (2.14) and the decision function (2.16) are modified by substituting all vectors by their feature vectors. Figure 2.5 shows an example. This leads to some computational problems. Since the dimension of $\mathcal{F}$ can become very large (even infinite) the computation of Φ may be expensive since *every* element of the training data has to be transformed to its feature vector.

The described problem of computational expense is avoided in a very elegant way by introducing so called *kernel functions* in the form of

$$k : X \times X \to \mathbb{R} \quad \text{with} \quad k(x, x') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle.$$

Using a kernel (function) allows to solve the optimization problem (2.14) and to evaluate the decision function (2.16) *without* the need to compute or even to know Φ. The optimization problem then becomes

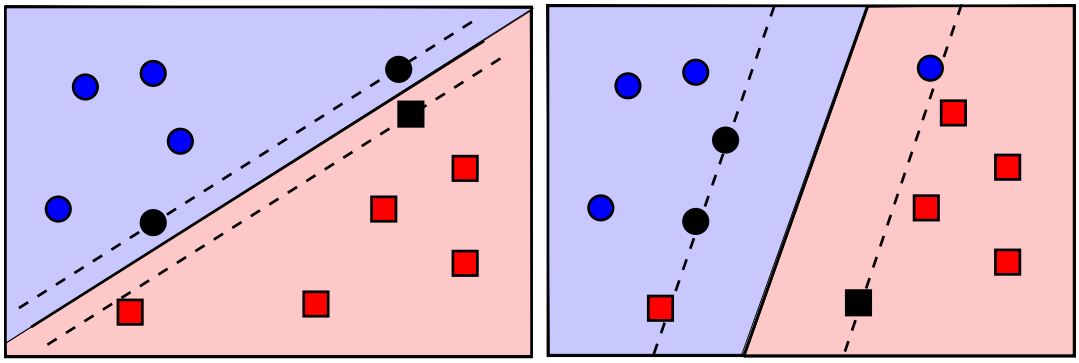
$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}, \mathbf{x}_i), \tag{2.17}$$

with subject to

$$\alpha_i \geq 0, \quad i \in \{1, ..., m\} \qquad \text{and} \qquad \sum_{i=1}^{m} \alpha_i y_i = 0.$$

The decision function becomes

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \right). \tag{2.18}$$



Hard-margin classifier Soft-margin classifier

Figure 2.6.: Left is an example of a hard margin classifier. Single outliners drastically change the classifier and the margin becomes smaller, which is not desirable. Using a classifier, which allows single outliners (right), leads to a better classifier.

Since the individual elements of the feature vectors are not used in any form, $\mathcal{H}$ may even be infinite. Another very important advantage is that only output values of the kernel function are used for building a classifier. Thus, the input data now is not anymore limited to a dot product space, but may take any form. Using appropriate kernel functions, one may build classifiers working on different data-structures like graphs, vectors in spaces with different dimensions or, like in this work: strings. Since a dot-product can be seen as a similarity measure in $\mathcal{X}$, the kernel function's output can be seen as a similarity measure in $\mathcal{F}$.

2.2.4. Soft Margin Hyperplanes

In practice, a separating hyperplane constructed by the above method is not the best choice, since all patterns are taken *equally* into account. This means that outliners near to the hyperplane may drastically change the result. Figure 2.6 shows an example. An algorithm, which tolerates a certain fraction of outliners is desirable. Unfortunately, the problem of finding a hyperplane whose training error is bounded by a constant is NP-hard (Ben-David and Simon, 2001). Instead, patterns in the feature space are allowed to violate the margin inequality (2.10). This is done by introducing the so-called *slack variables*

$$\xi_i \geq 0, \quad 1 \leq i \leq m \tag{2.19}$$

to relax the separations constraints

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i, \quad 1 \leq i \leq m. \tag{2.20}$$

By making ξ_i large enough, the new constraints (2.19) can always be met. In order not to obtain this trivial solution, the slack variables need to be penalized in the

objective function (2.9). This is referred to as C-SV classifier and is done by solving the following minimization problem for some $C > 0$,

$$\min_{\mathbf{w}\in\mathcal{H},b\in\mathcal{R},\boldsymbol{\xi}\in\mathbb{R}^m} \frac{1}{2}||\mathbf{w}||^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i,$$

subject to the constraints (2.20) and (2.19).

The optimization problem (2.17) then becomes

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j k(\mathbf{x},\mathbf{x}_i),$$

with subject to

$$0 \geq \alpha_i \geq \frac{C}{m}, \quad i \in \{1,...,m\} \qquad \text{and} \quad \sum_{i=1}^{m}\alpha_i y_i = 0.$$

C is referred to as *regularization parameter* of the SVM. Depending on the size of C, the margin may become larger than in the hard-margin case, but never smaller. The resulting hyperplane is not different to one built with a hard margin when $\xi_i = 0$ for all $1 \leq i \leq m$. A very high value of C has the same effect. The violations of (2.20) are penalized very hard, resulting in a smaller margin. A very low value of C results in a larger margin, as the penalty is small even for large values of ξ_i. Since a margin, which is either very small or very large leads to more errors in classification of unknown examples, it has to be chosen in such way that the number of errors is minimal.

2.3. Kernels

The so called *gram matrix* (or *kernel matrix*) $\mathbf{K}$ of a kernel $k : X \times X \to \mathbb{R}$ and patterns $\mathbf{x}_1, ..., \mathbf{x}_m \in X$ is given by

$$\mathbf{K}_{i,j} = k(x_i, x_j). \tag{2.21}$$

Given a feature map $\Phi : \mathcal{X} \to \mathcal{F}$, the kernel is trivially given by $k(x, x') = \langle\Phi(\mathbf{x}), \Phi(\mathbf{x}')\rangle$. Given a symmetric function $k : X \times X \to \mathbb{R}$ and patterns $\mathbf{x}_1, ..., \mathbf{x}_m \in X$, the *Mercer condition* (Mercer, 1909) induces that k is a kernel function if and only if its gram matrix $\mathbf{K}$ is positive semi-definite, meaning that for any $\mathbf{z} \in \mathbb{R}^m$

$$\mathbf{z}^T\mathbf{K}\mathbf{z} \geq 0. \tag{2.22}$$

This criterion can be used to find new kernels. The choice of the kernel function is critical regarding performance of a learning machine which uses SVM. Different kernels lead to different feature spaces with different dimensions and has consequences

for classification quality. Note that kernels have to be chosen *a-priori*, since their performance strongly depends on the type of underlying data.

As suggested in (Graepel, 2001), in this work, *all* kernels are normalized implicitly before being used for SVM learning. Given a kernel $k : X \times X \to \mathbb{R}$, the normalized kernel $\widehat{k} : X \times X \to \mathbb{R}$ is given by

$$\widehat{k}(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)k(x', x')}}. \tag{2.23}$$

Details can be found in (Shawe-Taylor and Cristianini, 2004, 112).

2.3.1. String Kernels

In this work, Support Vector machines are used to classify sequences of amino-acids, which can be represented by characters, so kernels which on strings are used. A string kernel computes a similarity measure of two strings. This may be done generically, or by taking a-priori knowledge of nature of underlying data into account. For example, there are kernels, which work on sequences of amino-acids that take chemical similarities of individual letters into account (an example is in 1.3). However, the possibility of applying these kernels to *any* kind of sequence based data is restricted. Additionally, recent results show that biologically motivated kernels are even outperformed by generic ones (Boisvert et al., 2008). The kernels used in this work do *not* depend on any biological or chemical knowledge and thus can be applied to any kind of sequence based data.

In the following, $\mathcal{A}$ is an finite alphabet with $|A| = l \geq 1$. A *string* of length $n \in \mathbb{N}_0$ is a sequence $s = a_1...a_n$ of $|s| = n$ characters ($a_i \in \mathcal{A}$). ϵ is the empty string with $|\epsilon = 0|$.

$$\mathcal{A}^* = \{a_1...a_n \mid a_i \in \mathcal{A}, n \in \mathbb{N}_0\}$$

is the set of all strings that can be built using $\mathcal{A}$. $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\epsilon\}$ is the set of strings with at least one character. The set of all possible strings of a fixed length $k \in \mathbb{N}$ over an alphabet $\mathcal{A}$ is given by

$$\mathcal{A}^k = \{s | s \in \mathcal{A}^* \text{ and } |s| = k\}.$$

This set has $|\mathcal{A}^k| = |\mathcal{A}|^k$ elements. Using any partial order $\leq$ on $\mathcal{A}$ (for example lexicographic order), it is possible to enumerate these strings by

$$\mathcal{A}^k = \{s_1, ..., s_{|\mathcal{A}^k|}\}.$$

Given two strings $u, v \in \mathcal{A}$ given by $u = u_1...u_s$ and $v = v_1...v_t$, $(s, t \in \mathbb{N}_0)$, the *concatenation* $uv = u_1...u_s v_1...v_t$ is also a string.

$X \subseteq \mathcal{A}^+$ is the input space of the given data: a set of non-empty strings over the alphabet $\mathcal{A}$.

2.3.2. Spectrum Kernel

One of the state-of-the-art methods of efficiently computing a similarity measure for strings without using any further knowledge is the widespread *Spectrum Kernel* (Leslie et al., 2002). It is based on the number of (continuous) subsequences in the two input strings. The function $\phi : \mathcal{A}^+ \times \mathcal{A}^+ \to \mathbb{N}_0$ given by

$$\phi(s, s') = |\{(\alpha, \beta) | \alpha, \beta \in \mathcal{A}^* \text{ and } s = \alpha s' \beta\}|$$

counts how often s' appears in s.

For a given $p \in \mathbb{N}$, an alphabet $\mathcal{A}$ and the resulting enumerated set (enumerated using any order) $\mathcal{A}^p = \{s_1, ..., s_{|\mathcal{A}^p|}\}$, the feature space is $\mathcal{F} = \mathbb{N}_0^{|\mathcal{A}^p|}$ and the feature map $\Phi_p : \mathcal{A}^+ \to \mathcal{F}$ is given by

$$\Phi_p(s) = (\phi(s, s_1), ..., \phi(s, s_{|\mathcal{A}_p|})).$$

The components of the vectors in the feature space are the number of times, every possible string of the length p over the alphabet $\mathcal{A}$ occurs in the string s.

The p-Spectrum kernel $k : \mathcal{A}^+ \times \mathcal{A}^+ \to \mathbb{R}$ is then the dot-product of two feature vectors

$$k_p(s, s') = \langle \Phi_p(s), \Phi_p(s') \rangle.$$

The specific order of elements in $\mathcal{A}^p$ is not important, since the result of the kernel based on the dot product is not influenced. An example will be given at the end of this section.

In contrast to the dimension of the feature space $|\mathcal{A}^p| = |\mathcal{A}|^p$, which grows exponentially in p, the number of non-zero components of a string's feature vector is bounded by $|s| - p + 1$: There are only this many possibilities for placing a substring of length p in s. This property allows to efficiently compute kernel values without having to compute Φ_p. Instead, it is possible to compute $k(s, s')$ in $O(p \cdot \max(|s|, |s'|))$.

2.3.3. Distant Segments Kernel

The Spectrum kernel does not consider the position of used substrings and does only take subsequences *of* a certain length into account. A recently proposed method for handling the latter is the *Distant Segment Kernel* (Boisvert et al., 2008). It is, in some sense an extension of the spectrum kernel, since it includes relative positional information of segments in a string of symbols. Additionally, it takes substrings *up to* a certain length into account. It is based on the size of the set of all substrings of a string of length δ that begin with α and end with α', given by the function $\phi : \mathcal{A}^+ \times \mathcal{A}^+ \times \mathcal{A}^+ \times \mathbb{N} \to \mathbb{N}_0$ with

$$\begin{aligned} \phi(s, \alpha, \alpha', \delta) = |\{ \quad (\alpha, \alpha', t, \beta, \beta') \quad | \quad & s = \beta \alpha t \alpha' \beta', \\ & \delta = |s| - |\beta| - |\beta'|, \\ & t, \beta, \beta' \in \mathcal{A}^* \quad \}|. \end{aligned}$$

Given a fixed substring length maximum δ_m, a fixed maximum segment length $\theta \in \mathbb{N}$ of α, α', and if

$$\alpha, \alpha' \text{ each take all possible string values in } \mathcal{A}^+ \text{ with } |\alpha|, |\alpha'| \leq \theta,$$
$$\delta \text{ takes all possible values with } 2 \leq |\alpha| + |\alpha'| \leq \delta \leq \delta_m,$$

the feature map $\Phi_{\delta_m,\theta} : X \to \mathcal{F}$ maps a string s to a feature vector, where each component has the form

$$\Phi_{\delta_m,\theta}(s) = (..., \phi(s, \alpha, \alpha', \delta), ...).$$

The position of these components in the feature vector is, like in the Spectrum kernel, dependent on the used partial order of all possible instances of α and α'. Any order is convenient.

The distant segments kernel $k_{\delta_m,\theta} : X \times X \to \mathbb{R}$ is given by

$$k_{\delta_m,\theta}(s, s') = \langle \Phi_{\delta_m,\theta}(s), \Phi_{\delta_m,\theta}(s') \rangle.$$

It is computed for a fixed maximum value of θ of segment sizes and a fixed maximum value δ_m of substring length. The dimension of its induced feature space grows exponentially in θ and δ_m and is non-trivial to calculate. However, it is much larger than the dimension of the Spectrum kernel. An example will be given at the end of this section.

The algorithm given to compute k for two given strings in (Boisvert et al., 2008) was shown to have the complexity $O(|s| \cdot |s'| \cdot \min(|s|, |s'|, \delta))$.

2.3.4. Examples and kernel parameters

Feeding two strings $s = ABA$ and $s' = BABA$ into the Spectrum kernel with $p = 2$ gives $k_2(s, s') = 3$. The substring AB occurs once in each string, the substring BA occurs once in s and twice in s'.

Using the DS-kernel with $\delta_m = \theta = 2$ and the same strings s, s' gives $k_{2,2}(s, s') = 7$. Table 2.1 shows substrings that were used for the calculation. Note that only substrings which occur in s and s' are counted. The feature vector contains an element for each *possible* substring.

Using both kernels, size of the induced feature space grows exponentially as their kernel parameter grows. Feature vectors of the Spectrum kernel only contain substrings of length of the kernel parameter. This may be a disadvantage since for complex data, a high dimensional feature space is needed to model all hidden dependencies. Setting the kernel parameter to a higher value results in a high dimensional feature space. However, as the kernel parameter grows larger, substrings of less length are not taken into account anymore, which is a kind of information loss. In addition, it is unlikely that long substrings appear twice in a set of strings, so feature vectors become sparse.

The DS-kernel does not suffer from this problem since it takes *all* substrings of length *up to* the kernel parameter into account. In addition, the dimension of the

starts with	ends with	# in ABA	# in $BABA$
A	A	1	1
A	B	1	1
B	A	1	3
A	BA	1	1
AB	A	1	1

Table 2.1.: Number of occurrences of substrings of $s = ABA$ and $s' = BABA$ which start and end with a certain substring. Note that only substrings which occur in s and s' are stated.

DS-kernel's feature space is *much* larger than the dimension of the Spectrum kernel's feature space. This theoretically results in a classifier that is able to model more complex data.

3. Methods

The following chapter is a summary of methods that are used in this work. First a brief description of the process that has to be performed to build a classifier is given in section 3.1. Each part of the overall procedure is then introduced in detail: Section 3.2 describes *cross-validation* as main technique for measuring classifier performance and generalization ability. Section 3.3 introduces different performance measures, namely *Accuracy*, *F-measure* and *AUC*, their meaning, advantages, disadvantages and how to use them in context of cross-validation. Since *experimental variance* is an important subject for all statistically based experiments, techniques for increasing confidence of results are introduced in section 3.4. In section 3.5 a method of how to select appropriate parameters for kernel and SVM, called *grid-search* is described. Section 3.6 contains a faster *bisection*-based for searching the maximum of an unknown function, which may, under some assumptions, be used as a replacement for grid-search. Both techniques are analyzed in detail. Finally, in section 3.7, a measure of disorder of datasets, the *sequence based entropy* is developed on the base of the Shannon-entropy.

Cross-validation is a standard technique, as the used performance measures are so sections 3.2 and 3.3.1 do not use other resources than stated. The introduction of ROC in section 3.3.2 is based on (Fawcett, 2003). The ideas for averaging performance measures in section 3.3.3 come from (Forman and Scholz, 2009). Experimental variance and how to gain confidence of results (section 3.4) is a standard mathematical subject and the introduced techniques are commonly used. The description of parameter selection in section 3.5 is not based on other external resources than stated. The method to improve the introduced search techniques in section 3.6 is based on the standard idea of bisection but does not use any further resources, neither does the proof in appendix B. The idea of using an entropy- and sequence-based measure for the order or disorder of datasets in section 3.7 came during discussions with the advisors of this work.

3.1. Overview

To build an accurate classifier in the current context, first, parameters of SVM and kernel have to be chosen. This results in a classifier. Then, the classifier has to be evaluated. To do this, it has to be trained on input data, then it has to be tested on data that was not used for training before. The latter comes because otherwise, generalization ability would not be measured. Using results of the test, a performance measure has to be calculated. Possible measures will be introduced later. With the latter, it is possible to compare different classifiers.

To find a good classifier, a search for best parameters has to be performed: The above procedure has to be repeated for various different classifiers, until an appropriate one is found. The way, the parameters are chosen and when to stop the search depends on the search strategy.

In the following, all parts are introduced in detail.

3.2. Cross-Validation, Stratified Cross-Validation

There is the need of testing a built classifier in order to measure its performance. If testing is done on training data, there is a strong risk over-fitting it. Some data have to be left out for later testing, to test generalization ability. Since the underlying distribution of training data is unknown, it is impossible to identify a subset, which, when left out, does not change the results. To overcome this problem, *every* element of the given data is left out once in multiple tests.

The set of m training samples $T = \{(x_1, y_1), ..., (x_m, y_m)\}$ is randomly partitioned into $k \geq 2$ subsets T_i $(1 \leq i \leq k)$ of equal size with $\bigcup_{i=1}^{k} T_i = T$ and $T_i \cap T_j = \emptyset$ for $1 \leq i, j \leq k$ and $i \neq j$. Afterwards, k classifier C_i $(1 \leq i \leq k)$ are built, each based upon $\bigcup_{j=1, j\neq i}^{k} T_j$. Each classifier C_i is tested on the remaining partition T_i, which was left out in the training and is yet unknown to the classifier. By this procedure, every element in the training data is used for testing once. If $k = m$ the procedure is called *leave one out* cross-validation. A widely used value is $k = 10$[1] and is used throughout the work. In addition, note that small changes to k do not significantly affect the results. When using a small number of partitions, too much data is used for testing, instead for training, resulting in a bleary approximation of the data. When using an overly large number of partitions, too much data is used for training, instead for testing, resulting in over-fitting the data. The results of all k tests have to be combined to gain a result for the used classifier.

When using a performance measure which depends on class-label distributions in training data, there might be problems with randomly generated partitions. For example, when using training data where the ratio of the class labels is strongly unsymmetric, it might happen that there are partitions, which contain only patterns of one class label, which leads to undefined values of some performance measures. In (Forman and Scholz, 2009), it is demonstrated that this leads to several problems.

The random partitioning is therefore constrained. For any pair of partitions T_i, T_j $(1 \leq i, j \leq k)$, it is claimed that

$$||\{(x, y) \mid (x, y) \in T_i \wedge y = 1\}| - |\{(x, y) \mid (x, y) \in T_j \wedge y = -1\}|| \leq 1.$$

This is called *stratified* cross-validation. Its use prevents any occurrences of undefined performance measures. In addition, experimental variance is reduced since each

[1] See (Forman and Scholz, 2009; Boisvert et al., 2008; Schölkopf and Smola, 2001)

of the k classifiers is tested under the same conditions regarding class label distributions.

3.3. Performance Measures

To measure performance/quality of a classifier, its output on test data is evaluated. In this work, only binary classification tasks are considered. Let the names of the two classes be *positive* and *negative*. There are four possibilities regarding correctness of a classifier's output:

True positive if a sample from class *positive* is classified correctly,

False positive if a sample from class *negative* is classified wrong,

True negative if a sample from class *negative* is classified correctly,

False negative if a sample from class *positive* is classified wrong,

In the following, the abbreviations TP, FP, TN, FN are used. Also $P := TP + FN$ and $N := TN + FP$. The following measures are derived from the latter:

3.3.1. Basic Performance Measures

The *Accuracy*, given by

$$\frac{P}{P+N}$$

represents the rate of correct classifications. Accuracy is easy to read and understand and is good for a first estimate of reliability of a classifier, but it lacks information of the class specific performance. The following measures take this into account.

The *Precision*, given by

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

is the rate of positive classifications that are correct. The counterpart *Recall* or *Sensitiviy*, given by

$$\frac{TP}{P}$$

is the rate of positive elements that are classified correctly. The difference is subtle but significant. There may be classifier that has a high precision but a low Recall/Sensitivity. The counterpart to the Sensitivity, the *Specificity*, given by

$$\frac{TN}{N}$$

is the rate of negative elements that are classified correctly.

Precision, Recall/Sensitivity and Specificity are all measures of completeness of the classifier's output with reference to the classes.

The class specific measures are independent of the class label ratio of the training data. Using the Accuracy, an asymmetric ratio may be a problem. For example, a classifier that always outputs the same class gains 90 percent Accuracy on test data, which contains 90 percent labels of that class. A single scalar measure, which takes the class label distribution into account is the so-called *F1-measure* or *F-measure*, given by

$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}.$$

It is the balanced harmonic mean of Precision and Recall. The F-Measure combines aspects of many measures into one. It is robust to asymmetric class label ratios and takes the class specific accuracy and completeness of a classifier into account. It is yet undefined in some situations, namely if $TP = FP = 0$ (Precision undefined) or $TP = FN = 0$ (Recall undefined). To prevent this, according to (Forman and Scholz, 2009), the definition is extended to equal zero in the mentioned situations.

$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{(\frac{TP}{TP+FP}) \cdot (\frac{TP}{TP+FN})}{(\frac{TP}{TP+FP}) + (\frac{TP}{TP+FN})} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

3.3.2. Receiver Operating Characteristic and Area Under the Curve

Another widely used tool to visualize classifier performance is the *Receiver Operating Characteristic (ROC)* curve. "ROC graphs are commonly used in medical decision making, and in recent years have been increasingly adopted in the machine learning and data mining research communities." (Fawcett, 2003, 1). Since many researchers use ROC based values to compare classifiers, the procedure is also applied in this work.

ROC graphs are two-dimensional graphs in which *True Positive Rate* ($\frac{TP}{P}$) is plotted on the vertical axis and *False Positive Rate* ($\frac{FP}{N}$) is plotted in the horizontal axis. They depict relative trade-offs between benefits (TP) and costs (FP). For drawing ROC graphs, the underlying classifier must output a *score*. The graph is drawn by successively increasing a threshold on that score and counting the number of errors, which a classifier does when using the threshold for decision of a pattern's class. When using SVM, usually the term, which is substituted into the *sign*-function, in equation (2.18) is used for this purpose. Another possibility would be to use probabilistic output as for example described in (Platt, 1999). A basic algorithm for creating a ROC curve can be found in (Fawcett, 2003).

Since single scalar values are easier to compare, the ***A**rea **U**nder the **C**urve* (AUC), is often used to do this. As the AUC is a portion of the area of the unit square, its value is situated in $[0, 1]$. The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

Depending on the output of the score function, there might be problems when using ROC curves and the AUC. There might be cases when a classifier, which is acutally better than another one has a worse AUC. The sensitivity/specificity should always

come with the ROC curve and derived measures to prevent overemphasizing the results. There will be an example in the results section.

3.3.3. Performance Measures and Cross-Validation

There are many ways to average results of different cross-validation runs. Intermixing these brings problems, as for example, results of different folds may be uncalibrated or even incompatible to each other. Also, some methods of averaging lead to biased results. This work follows the suggestions of (Forman and Scholz, 2009).

Let $(TP)_i, (FP)_i, (FN)_i, (TN)_i$ be the number of true/false positives/negatives in cross-validation run i $(1 \leq i \leq k)$. Also let $(P)_i = \sum_{i=1}^{k} ((TP_i) + (FN_i))$ and $P = \sum_{i=1}^{k} ((TP_i) + (FN_i))$ be the numbers of positive/negative samples in run i..

The averaged *F-Measure* is computed by using the sum of the classifier's answers in all k test runs.

$$\frac{2 \cdot \sum_{i=1}^{k} (TP)_i}{2 \cdot \sum_{i=1}^{k} (TP)_i + \sum_{i=1}^{k} (FP)_i + \sum_{i=1}^{k} (FN)_i}$$

For the AUC, it is not possible to merge scores of the cross-validation runs, since they are not calibrated. Instead, the averaged AUC is computed as the mean of results of single runs. Let $(AUC)_i$ be the AUC value of cross-validation run i, the mean is given by

$$\frac{1}{k} \sum_{i=1}^{k} (AUC)_i$$

The *Accuracy* is also used for reference. It has the nice property that its value is independent of the method (averaging or merging), which was used for calculation:

$$\frac{\sum_{i=1}^{k} (TP)_i + \sum_{i=1}^{k} (TN)_i}{\sum_{i=1}^{k} (P)_i + \sum_{i=1}^{k} (N)_i} = \frac{1}{k} \cdot \sum_{i=1}^{k} \frac{(TP)_i + (TN)_i}{(P)_i + (N)_i}$$

3.4. Reduction of Experimental Variance

So far, all results of the experiments are in form of single values. To supply confidence intervals and to discuss relevance of gained results, *experimental variance* has to be studied. Experimental variance in context of this work means that when an experiment is repeated multiple times, the observed results are not equal for each repetition. This behavior is primarily is caused by the introduced main technique for classifier evaluation, cross-validation, which is based on *random* partitioning of data.

Normally, in a deterministic experiment, same input leads to same output. However, due to use of cross-validation, which does random partitioning, there is a probabilistic factor introduced. Consequently there is the problem that results may be hard to interprete because they are too noisy (example are in the results section).

As already seen, stratified cross-validation is used to keep the ratio of class labels at least constant in each fold. Another way of reducing variance is simply to repeat an experiment multiple times and to average its results. There is no point in repeating a deterministic part in an experiment, so the repetition has to take place before the point where cross-validation and thereby randomness is introduced. Therefore, the cross-validation procedure is repeated a certain number of times. Since underlying data is equal for every repetition, results are comparable and the *arithmetic mean* of the gained results may be used. With the number of repetitions increasing, variance will be reduced. The size of a dataset also has an impact on variance: Small datasets lead to a larger variance than large datasets. Therefore, the number of repetitions should be chosen with size, available time and desired confidence in mind.

To be able to discuss relevance of results of an experiment, confidence intervals are determined. These give a range in which, the *actual* value of a result is situated with a certain probability.

To gain confidence intervals, it is important to have knowledge of the underlying probability distribution. Since the only random part in the performed experiments is random partitioning of data, it may be assumed that averaged results are normally distributed, especially with the number of repetitions getting larger. According to the *central limit theorem*, the average mean of n iid[2] random variables is normally distributed as n goes to infinity, irrespective of the distribution of the variables. However, to be sure, the distribution will be checked in the experiments.

Due to the computational expense of the introduced search techniques, it is unfeasable repeating the experiments a many times in order to reduce variance. Instead, the search is performed with an appropriate number of repetitions to eliminate noise. Then, the resulting single classifier is evaluated with many more repetitions, which results in very tight confidence intervals.

3.5. Parameter Selection

To build a classifier, reasonable parameters have to be selected. In this work, two parameters are involved, namely the regularization parameter C of the SVM and the parameter of the used string-kernel. To find the best possible parameters, the classifier, which leads to the best performance quality has to be found. This problem can be formalized in the following way: Assuming that a measure which is normed to $[0, 1]$ is

[2] independent and identically distributed

used for measuring the performance of a classifier, let

$$f : \mathbb{R} \times \mathbb{N} \to [0, 1]$$

be the function that describes the dependency between the classifier's parameters and its performance quality. f is *unknown* and the goal is to find its maximum. It is possible to evaluate f at any point, which corresponds to building a classifier with the given parameters and measuring its performance. This introduces computational costs. Let the costs for evaluating f be given by a constant $k \in N$. For example, the number of SVM classifiers that are trained to evaluate f may be used to determine the costs.

To find the best parameters, a search has to be devised. It can be divided into multiple nested loops, one for each parameter. This way, in the innermost loop, all parameters are fixed except one. In the following, a standard search technique, which is often used in SVM context, is introduced.

3.5.1. Grid-Search

In (Hsu et al., 2003), a simple search of all possible pairs of parameters, called *grid-search*, is proposed. For a given interval for both parameters, all possible pairs of parameter pairs are tried, first with a coarse grid to determine the region where the best parameters located, and then with a finer grid to gain more insight into that region.

The implicit assumptions made on f (which are not described in (Hsu et al., 2003)) are:

- f is "smooth" (does not oscillate fast),
- the intervals where the parameters, which lead to the maximum of f, are situated are known beforehand.

Empirical results allow the assumption that f is smooth. There will be a discussion in the results section. The intervals where the best parameters are situated can derived using knowledge of the nature of the used classifier as discussed in the following.

3.5.2. Parameters to Search for

The *regularization parameter* C is searched in an interval similar to $[2^{-5}, 2^{15}]$, (taken from (Hsu et al., 2003)). There is no point in selecting the regularization parameter of a C-SV classifier smaller than zero. Similarly, for high values, performance will get worse because the margin of the SVM gets smaller, resulting in worse generalization ability. First a *coarse* exponential search is performed to identify the best region of the results, e.g. $2^{-5}, 2^{-4}, ..., 2^{14}, 2^{15}$. Then a *fine* exponential search is performed in that region to sharpen the extremum, e.g. with 2^3 being the result of the coarse search: $2^2, 2^{2.1}, ..., 2^{3.9}, 2^4$. Note that this introduces two parameters for the step width. These will be discussed later.

In contrast, the *kernel parameter* of the used string kernels is used for *substring length*, so there is no point in searching for it exponentially. Instead, it is searched for *linearly*. Since it is a natural number, the smallest possible change, which is one, may be used as step width. A reasonable interval is e.g. $[1, \min(35, L)]$, where L is the minimum length of a sequence in the underlying dataset. First, substrings of zero length do not exist in practice. Second, the probability that a substring of a given length occurs in an amino-acid sequence converges to zero as the length of this substring increases to infinity. Consequently, in context of amino-acid sequences, it is unlikely that long subsequences appear twice, but more likely that subsequences of small or medium size appear multiple times, since they may form "motifs" whose biochemical function is needed in various places.

3.5.3. Different Performance Measures

A remaining question is: Which performance measure is used to compare classifiers and to find an optimum? In (Hsu et al., 2003), the *Accuracy* is suggested. As already mentioned in section 3.3.1, there might be problems using the Accuracy on asymmetric training sets. Therefore, along with the Accuracy, a number of different performance measures will also be used to find best parameters, namely: Accuracy, AUC and F-measure. The results will be compared. Since all introduced performance measures are normed to $[0, 1]$, their choice does not affect the content of this section.

3.5.4. Grid-Search: Parameters, Performance and Costs

Along knowledge of intervals, where the search has be performed, the step width of the coarse and the fine part of the search are needed as parameters. These parameters affect the accuracy (not: Accuracy) of the search: The smaller they are, the more accurate the search is. If the values are too large, maxima may be "overseen". If they are too small, the costs will explode. The grid-search suffers from multiple problems:

- It is not guaranteed to find the maximum of the function f. The problem is that there might be maxima, which are hidden between two evaluations of f, while the *evaluated* values are *small*. Figure 3.1 depicts this problem. The assumption that f is smooth decreases the chances for these kind of situations. In practice, the step width is set relatively small to avoid such behavior. However, this leads to the second problem, as described in the following.

- Since the step width of the search is constant, and though has to be set relatively small to have an appropriate resolution in neighborhood of a maximum, many effort is put into evaluation in regions of the function that do not contribute anything to the final result. That's why the computational costs of this kind of search technique are huge.

The number of model evaluations grows exponentially with the number of parameters. In the given situation, there fortunately were only two. When using string

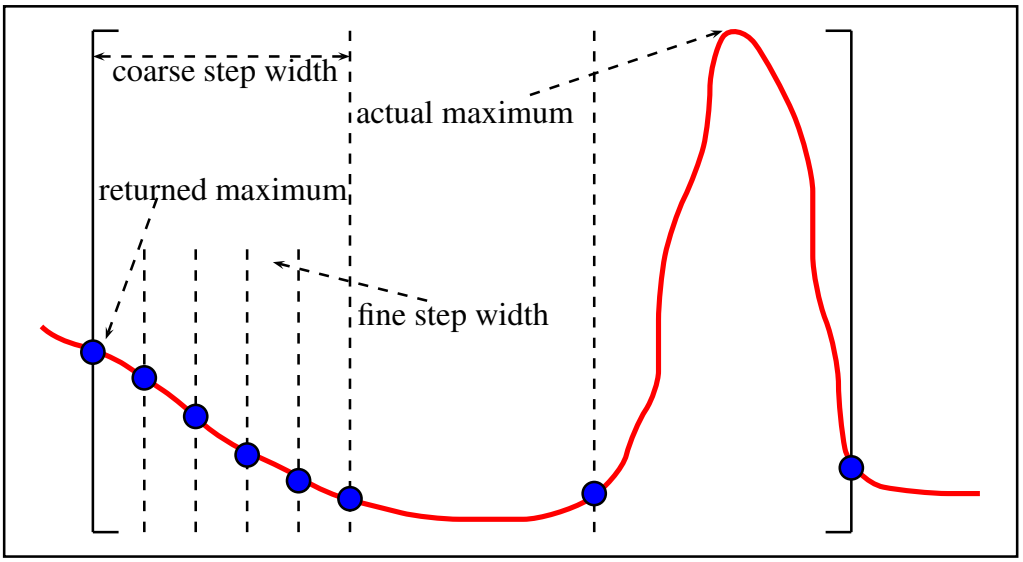


Figure 3.1.: Illustration of a situation where the grid-search fails. Because the step width is too large, the maximum is "overseen" and even worse, a point is returned, which is far away from the actual maximum.

kernels, the computation of the kernel matrix, which has to be done for each kernel parameter, is quite time consuming, especially when using large datasets containing long sequences.

Assuming that the area, in which the fine search takes place has a size of 2α, the costs for one evaluation of f given by k, given an interval $]a, b[$ and step width parameters $\alpha \in \mathbb{R}$ for coarse step and $\beta \in \mathbb{R}$ for fine step, the costs for the grid-search for *one* search parameter are given by

$$K_{grid} = \left(\frac{b-a}{\alpha} + 1\right) \cdot k + \left(\frac{2 \cdot \alpha}{\beta} + 1\right) \cdot k = \left(\frac{b-a}{\alpha} + \frac{2 \cdot \alpha}{\beta} + 2\right) \cdot k. \tag{3.1}$$

This number is multiplicated with itself for every additional parameter, resulting in K^2_{grid} for two parameters, K^3_{grid} for three, and so on. Due to this exponential cost raise, the grid-search is *unfeasible* for a larger number of parameters. However, note that the costs are *fixed*, so the procedure is guaranteed to *terminate*.

An advantage is that the individual parameter searches are independent, therefore the procedure may be parallelized.
Under some additional assumptions, the search may be accelerated.

3.6. Bisection Based Method for Searching for Regularization Parameters

3.6.1. Motivation: Costs and Accuracy

As it will be described in the results parts, search-curves for regularization parameters are nearly *continuous* and only have one maximum. There are theoretical reasons for this kind of behavior. The nature of the regularization parameter of the C-SV classifier explains why only one maximum is likely to appear: The parameter is a control variable for the classical trade-off between memorizing data and generalization ability. It penalizes vectors in the feature space that are close to the separating hyperplane. A low value results in more margin errors and a weak performance measure, a high value results in a hyperplane that fits data too close, introducing more errors on unseen data. There exists an extremum, where the trade-off is optimal. Additionally, subtle changes of the parameter lead to subtle changes of the classifier performance, which explains that the search-curves are nearly continuous. Examples are in the result section.

With these curves in mind, an advanced approach of finding the best parameter can be thought of.

3.6.2. Basic Idea: Bisection

The task of finding the maximum of f by only using single evaluations can be extended to find the maximum of f by only using single evaluations such that the sum of all costs (number of evaluations of f) is *minimal*. With the additional information that there is only one local maximum, the naive grid-search approach of evaluating f for more or less all possible parameters, may be improved concerning costs. Furthermore, an approach for finding the maximum, that converges to the actual maximum can be formulated, so finding a maximum is possible with any desired accuracy while avoiding the evaluation of f for many useless points in the search interval.

In the following, for simplicity, problem and technique are described for searching the maximum of a one-dimensional function

$$f : \mathbb{R} \to \mathbb{R},$$

which is unknown and to be maximized by only using single evaluations of it, while keeping the sum of evaluations small. This approach may easily be applied to search for the regularization parameter C of a SVM, and also to a search for other parameters, as long as the following assumptions are met:

- f is continuous,
- f has only one maximum in the search interval $]a, b[$ and
- an interval $]a, b[$ where the global maximum is situated is known (may be arbitrary large).

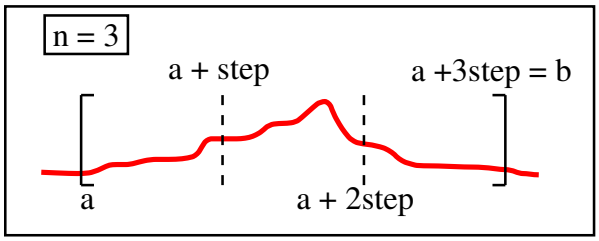


Figure 3.2.: Illustration how the sub-intervals are built.

Since the function f is unknown and therefore the derivatives are also unknown, the Newton method or any gradient based methods for finding extreme points are not usable.

The method, which is implemented is based on bisection for finding a function's root. The basic idea of the approach, which will be introduced, is the following: If there are found three parameters x, y, z with $x < y < z$ and

$$f(x) < f(y) > f(z),$$

the only maximum is definitely located in $]x, z[$. An algorithm may work as the following:

1. Split the interval where the maximum is situated into three subintervals of equal size (see figure 3.2).

2. Evaluate f at all four interval borders (there is one more interval border than there are intervals, see figure 3.2).

3. If, among these results, there are three values x, y, z with $x < y < z$ and $f(x) < f(y) > f(z)$, then the maximum lies in $]x, z[$. Use this interval and start from 1.

This way, the interval where the maximum is situated becomes smaller on every iteration. The shrinking factor is $\frac{2}{3}$ (meaning that the size of the interval is reduced by $\frac{1}{3}$.) There might be situations, where step three does not lead to any refinement. In this case, the interval is split into a larger number of equal values and the procedure is started from step one. The shrinking factor of the search interval then becomes larger, but, at costs of a larger number of evaluations of f.

3.6.3. Algorithm

The implementation is a little more complicated. Algorithm 3.1 shows the basic procedure. Inputs are: the search interval $]a, b[$ to start with, the termination criterion *tolerance* (will be explained in the following) and the unknown function f, which has to be maximized. n is the number sub-intervals, which are used during one iteration. It is initialized with $n = 3$ in line 1. With this number, $step = \frac{b-a}{n}$ is calculated in line 7 and represents the size of each sub-interval. Note that the number of evaluation

points is $n+1$ (because $a + 0 \cdot step$ also has to be evaluated). The i-th evaluation point ($0 \leq i \leq n$) is given by

$$a + i \cdot step.$$

The rightmost evaluation point is given by

$$a + n \cdot step = a + n \cdot \frac{b-a}{n} = b.$$

Figure 3.2 depicts how the intervals are built.

The for-loop, which starts in line 8 works in the following way: A "sliding window", which contains three sequential evaluation points is moved from the left-most position to the right-most position, i.e. left-most:

$$x = a,$$

$$y = a + step,$$

$$z = a + 2 \cdot step,$$

rightmost:

$$x = a + (n-2) \cdot step,$$

$$y = a + (n-1) \cdot step,$$

$$z = a + (n) \cdot step = b.$$

The resulting three function values $f(x), f(y), f(z)$ are evaluated and saved on each position (line 12). If among these window positions, one is found with $f(x) < f(y) > f(z)$, the search interval borders $]a, b[$ are set to $]x, z[$ and the procedure is started from the beginning (line 13 ff.):

There might be situations where in all window positions, no refinement is found. In this case, the number of subintervals n is increased by one. This is done via the boolean $flag$ variable. It is set to $true$ in the beginning of each iteration in line 6. If a refinement is found, it is set to $false$ in line 17. If, after the for-loop, it holds $flag = true$, no refinement was found and n will be increased by one. Note that n is set to three again when a refinement is found in line 16.

The main loop (line 2 to 24) can only be exited in line 3, which contains the termination criterion of the algorithm: If the search interval is smaller than the provided *tolerance* parameter, its center is returned.

A proof of the total correctness of algorithm 3.1 is rather technical and can be found in appendix B.

To clarify algorithm 3.1, see the following examples. For simplicity, only parts, which are needed for understanding the algorithm are depicted. Vertical lines represent sub-interval borders, outer vertical lines represent search interval borders. Letters below these line represent the current search window position. If a letter is framed, its corresponding function value is the largest of the three evaluated ones in search

Algorithm 3.1 Algorithm that outputs a value that is close to argmax $f(x)$ with any accuracy.

Inputs are:

- $a, b \in \mathbb{R}$, with argmax $f(x) \in]a, b[$,
- $tolerance \in \mathbb{R}$,
- $f : \mathbb{R} \to \mathbb{R}$.

Output is:

- $o \in \mathbb{R}$ with $|\operatorname{argmax} f(x) - o| < tolerance$.

$n \leftarrow 3$

loop
  if $b - a < tolerance$ **then**
    return $a + \frac{b-a}{2}$
  end if

  $flag \leftarrow TRUE$
  $step \leftarrow \frac{b-a}{n}$

  for $i = 0$ to $(n - 2)$ **do**
    $x \leftarrow a + (i) \cdot step$
    $y \leftarrow a + (i + 1) \cdot step$
    $z \leftarrow a + (i + 2) \cdot step$
    save evaluations of $f(x), f(y)$ and $f(z)$

    if $f(x) < f(y)$ **and** $f(y) > f(z)$ **then**
      $a \leftarrow x$
      $b \leftarrow z$
      $n \leftarrow 3$
      $flag \leftarrow$ **false**
      break for loop
    end if
  end for

  if $flag$ **then**
    $n \leftarrow n + 1$
  end if
end loop

window. Circles show, which points are evaluated.

Figure 3.3 shows a simple example of how the described algorithm works. In this example, the number of subintervals stays $n = 3$. Note how the *step* variable is decreased every time, a maximum is found. Also note that the reduction of the search interval becomes smaller on every refinement.

Figure 3.4 shows a more complex example. In step 3.4b, all positions of the search window did not lead to a refinement, so the number of sub-intervals is increased by one.

3.6.4. Parameters, Performance and Costs

With its assumptions met, the described algorithm outperforms the classical approach of coarse and fine grid-search in various ways

- The number of evaluations of f is less. Especially in regions far from the maximum. The convergence speed of the described method is numerically very large in the first iterations of the algorithm. It gets slower, as the search interval becomes smaller, so in practice the *tolerance* parameter has to be set to a value, which is not overly small.

- When the step width parameters of the grid-search are overly large, it may "oversee" the maximum (see figure 3.1) in contrast to the introduced method, where the position of the maximum is always *guaranteed* to be in the current search interval. The step parameters of the grid-search have to be set a-priori to perform well, while the tolerance parameter of the introduced method is just a measure how accurate the result is at minimum.

The costs of the described method are depending on the function that is maximized. Assuming three sub-intervals are sufficient to find a refinement in every iteration ($n = 3$ throughout the algorithm), the search interval's size is multiplicated with $\frac{2}{3}$ in each iteration. Given the *tolerance* parameter, the algorithm terminates after x iterations if

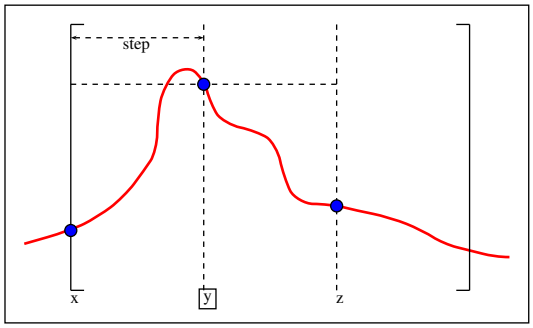
$$(b-a) \cdot \left(\frac{2}{3}\right)^x \leq tolerance \Rightarrow x \geq \lceil \log_{\frac{2}{3}} \left(\frac{tolerance}{b-a}\right) \rceil.$$

In every iteration, the costs are *at most* $4 \cdot k$ (based on the above assumption $n = 3$), where k are the costs for one evaluation of f. The overall costs then are bounded from above by

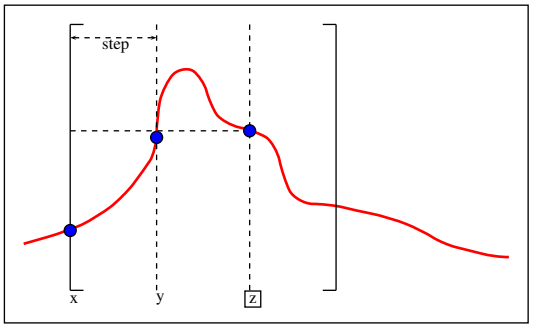
$$K_{bisection} \leq 4 \cdot \lceil \log_{\frac{2}{3}} \left(\frac{tolerance}{b-a}\right) \rceil \cdot k,$$

which, in comparison to the grid-search, is a decrease of the underlying complexity class, namely from linear to logarithmic.

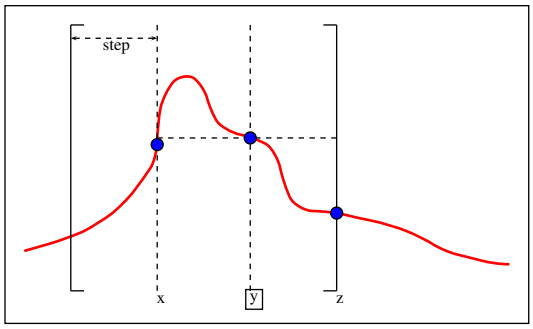
Example: Assuming a search interval is given by $]0, 100[$, grid-search parameters are given by $coarse = 1$ and $fine = 0.1$ and the parameter $tolerance = fine = 0.1$ is



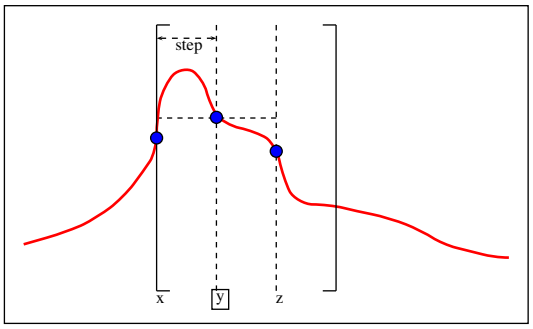
(a) (n=3) Start. Evaluation leads to refinement. Search is continued in x,z.



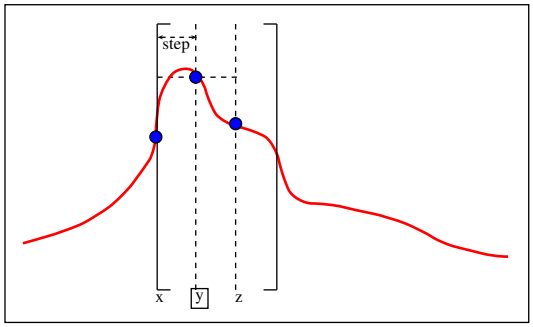
(b) (n=3) Evaluation does not lead to refinement because point in the middle is not higher than outer ones. Search window is moved one step to the right.



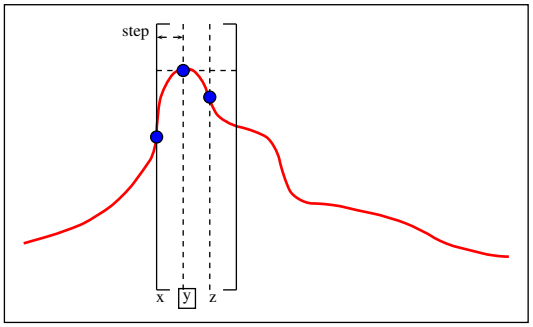
(c) (n=3) Evaluation leads to refinement. Search is continued in the interval x,z



(d) (n=3) Evaluation leads to refinement. Search is continued in the interval x,z

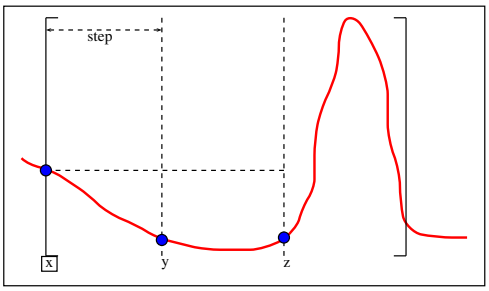


(e) (n=3) Evaluation leads to refinement. Search is continued in the interval x,z

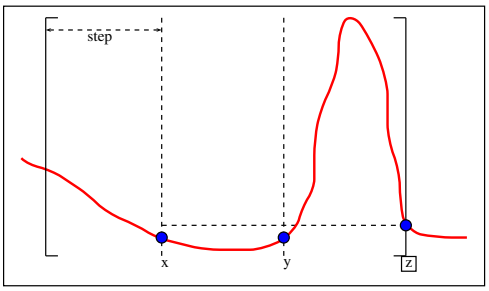


(f) (n=3) Evaluation leads to refinement. Search is continued in the interval x,z

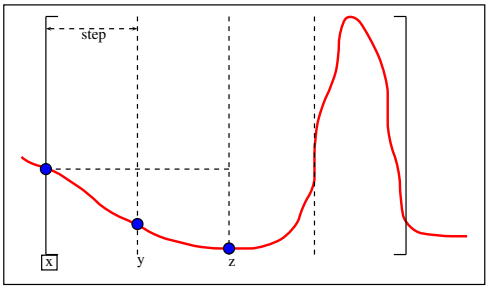
Figure 3.3.: Illustration how the described method works on a simple example. Vertical lines represent sub-interval borders, outer vertical lines represent search interval borders. Circles show, which points are evaluated. When of three evaluated points, the middle one is higher than the outer ones, the search is continued in that interval. Note that there is no need to increase the number of sub-intervals in this example, since three are sufficient.



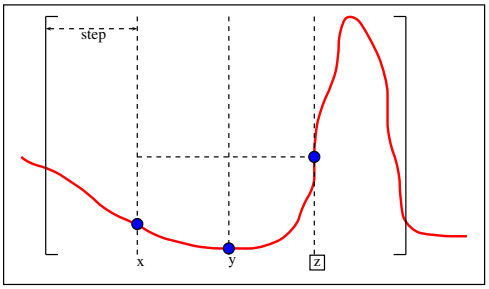
(a) (n=3) Start. Evaluation does not lead to refinement. Search positions x, y, z are moved one step to the right.



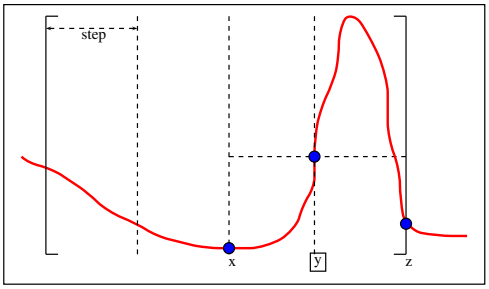
(b) (n=3) Evaluation does not lead to refinement. At this point, all positions did not lead to a refinement, so the number of subintervals is increased by one.



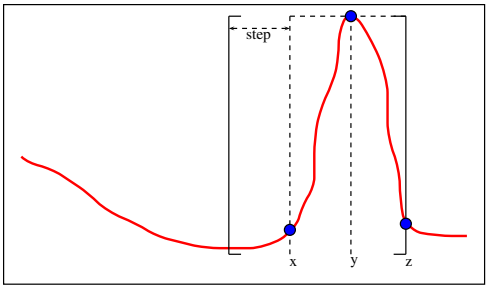
(c) (n=4) Start from left again. Evaluation does not lead to refinement. Search positions x, y, z are moved one step to the right.



(d) (n=4) Evaluation does not lead to refinement. Search positions x, y, z are moved one step to the right.



(e) (n=4) Evaluation of leads to refinement. Search is continued in the interval x,z.



(f) (n=3, skipped one step) Evaluation of leads to refinement. Search is continued in the interval x,z.

Figure 3.4.: Illustration how the described method works on a more complex example. The outer vertical lines represent the search interval, the circles show, which points are evaluated. When of three evaluated points, the middle one is higher than the outer ones, the search is continued in that interval. This example also depicts how the search resolution becomes finer when no refinement of the search interval can be found.

given to the described method. According to equation 3.1, the costs for the grid-search then are

$$K_{grid} = \left(\frac{100 - 0}{1} + \frac{2 \cdot 1}{0.1} + 2 \right) \cdot k = 122 \cdot k.$$

In contrast, while reaching at minimum same accuracy as the grid-search, the costs of the introduced method are bounded by

$$K_{bisection} \leq 4 \cdot \lceil \log_{\frac{2}{3}} \left(\frac{0.1}{100 - 0} \right) \rceil \cdot k = 72 \cdot k.$$

In practice, not every iteration leads to the worst case of four evaluations, so the number is likely to be smaller. Note that this example is based on the assumption that it is never necessary to increase the number of sub-intervals in the described method (see above for details). However, when the number of sub-intervals *is* increased, the shrinking factor of the search interval becomes larger, resulting in less necessary iterations, and therefore damps the costs for evaluating more points of the to be maximized function.

Some improvements concerning the number of evaluations can be reached when the evaluated values of f are saved during the algorithm, to avoid double evaluation of certain values. Additionally, when no refinement could be found using a certain *step* size, values of larger distance, which were yet evaluated (presuming they were saved), could be used to find a refinement before increasing the number of sub-intervals. Both ideas do further decrease the costs, but do not change the logarithmic complexity.

3.6.5. Problems: Continuity & Uncertainty

The described bisection method for finding a function's maximum has one major point of weakness: The assumption that the underlying function is continuous. If f is *not* continuous, the described method is *not* guaranteed to work. It is not shown in this work that f is continuous and therefore, no guarantees regarding the behavior of the described method can be made.

However, empirical results show that f is at least *nearly* continuous (these will follow in the results section). In practice, this is sufficient. If the amount of "discontinuity" is not overly large (i.e. f has no major "jumps"), the described method still works up to a certain point. It then can be seen as a kind of heuristic. Experimental results lead to usable results using the approach and are described in the results section. Additionally, the naive grid-search technique, which is widely used technique for parameter selection also works worse when the underlying function is not continuous.

It would be interesting to examine the actual function f regarding continuity, however, it is out of scope of this work to do this and therefore only mentioned in the last section of this work.

In practice, the search curves sometimes contain small *local maxima*. These are for example caused by noise or experimental variance (see section 3.4). For the introduced

bisection based algorithm, this is a problem because it is based on the assumption that there exists only one maximum and though, it may "turn to the wrong direction". In the following, a method of adding robustness against these weakly distinctive maxima is described.

Formally, from now on, the evaluation of f is not directly possible anymore. Instead, it is only possible to evaluate another function $\tilde{f} : \mathbb{R} \to \mathbb{R}$, given by

$$\tilde{f}(x) = f(x) + n,$$

where $n \in [-\epsilon, \epsilon] \subset \mathbb{R}$ is a random number bounded by $\epsilon \in \mathbb{R}$. This number represents uncertainty in the evaluation of f.

The following changes are made to algorithm 3.1:

- ϵ is provided as input parameter,
- each usage of f is replaced by $\tilde{f}$ and
- the maximum check in line 13, namely (f is already replaced by $\tilde{f}$)

  $$\textbf{if } \tilde{f}(x) > \tilde{f}(y) \textbf{ and } \tilde{f}(y) > \tilde{f}(z) \textbf{ then}$$

  is replaced by

  $$\textbf{if } \tilde{f}(x) + \epsilon > \tilde{f}(y) - \epsilon \textbf{ and } \tilde{f}(y) - \epsilon > \tilde{f}(z) + \epsilon \textbf{ then}$$

This way, only a refinement is found, when of three sub-interval borders x, y, z with $x > y > z$, the middle one leads to a function value, which is at least 2ϵ larger than the outer ones, namely $f'(x) + \epsilon < f'(y) - \epsilon > f'(z) + \epsilon$. Since the latter implies $f(x) < f(y) > f(z)$, any output of the modified algorithm is still guaranteed to be correct (according to the proof in appendix B). The advantage of this modification is that maxima, which are smaller than 2ϵ are ignored by the algorithm. This leads to a much more robust search. Figure 3.5 shows an example. Even multiple maxima are no problem, if they are small, so the requirements of the modified algorithm are less restricted than without the modification.

Despite the nice property of adding robustness against maxima, which are smaller than 2ϵ, the modification snatches the algorithm's guaranteed termination. It does *not* terminate when

$$tolerance < b^* - a^*,$$

where b^*, a^* are the closest values to $m = \operatorname{argmax} f(x)$, with the property

$$f(a^*) + \epsilon = f(b^*) + \epsilon = f(m) - \epsilon.$$

Figure 3.6 depicts the smallest possible search interval for a certain uncertainty bound. Unfortunately, $b^* - a^*$ is *unknown*.

However, in practice, the modification may still be of value. When the uncertainty bound ϵ is set to an appropriate small value in contrast to the parameter *tolerance*, the modified algorithm is at least *likely* to terminate. When is *does* terminate, its output is *guaranteed* to be robust against maxima smaller than 2ϵ, which is a useful statement. Non-termination may be avoided by adding a maximum number of iterations and returning the maximum value, which was found up to this moment. Setting $\epsilon = 0$ lets the modificated algorithm behave equivalent to the original one.

Further modification could also restore the total correctness, if the algorithm is modified in such way that the search interval converges to the smallest possible one.

3.6.6. Conclusion

The described bisection based algorithm outperforms the classical naive grid-search regarding costs, which are reduced from linear to logarithmic, and accuracy, since the grid-search may "oversee" maxima, in contrast to the introduced method, which outputs the maximum with any desired accuracy. These properties come at the cost of requiring continuity of the underlying function. This may not be proven in the scope of this work, however, *approximate* continuity is sufficient in practice (see 3.6.5)

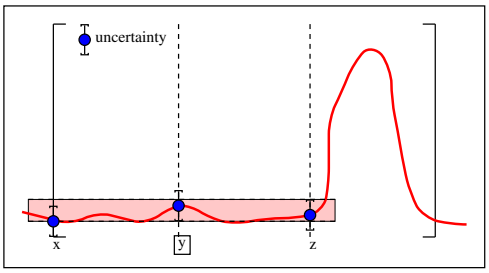
To handle small local maxima, which are for example caused by noise or experimental variance, the introduced modification (see 3.6.5) to algorithm 3.1 makes it robust against maxima, which are smaller than a certain uncertainty bound, which has to be provided as parameter. This modification snatches the algorithm's guaranteed termination, but this may be handled by wise parameter selection and/or a "maximum number of iterations" restriction. However, a further modification could eventually restore the total correctness if the search interval would converge to the smallest possible one.

3.7. Entropy

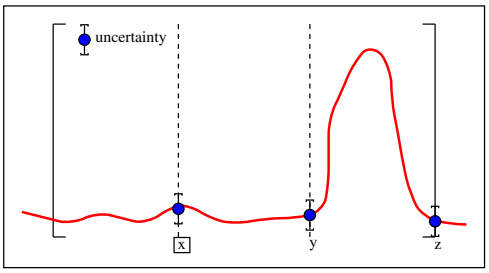
3.7.1. Motivation: Disorder of Data

When evaluating a classification technique, it is always of interest to spot both datasets, which lead to good and to weak results and to compare their attributes. If a correlation is found between attributes of datasets and results regarding classifier performance, it may be used to predict the applicability of a classifier to certain types of data. Since kernel choice is crucial for SVM-classification, it would be nice to find attributes of datasets that may be used to give a coarse prediction, how well a SVM with a certain kernel will work on certain data.

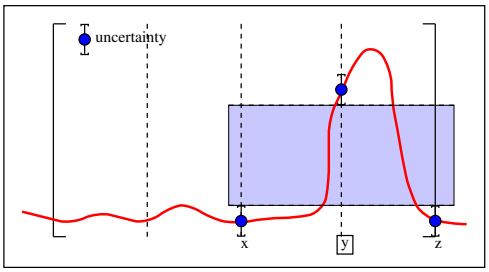
A straightforward attribute of a dataset is its *order* or *disorder*. It seems intuitive that a dataset, which contains homogeneous data is harder to classify than a divergent dataset. As described in section 2.3, kernels do calculate a pairwise similarity measure of data, so a homogeneous dataset (from the kernel's view) leads to feature vectors, which are distributed closely in the feature space. A divergent dataset leads to feature



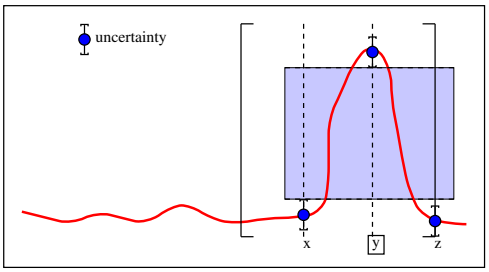
(a) (n=3) Although the middle point is located higher than the outer ones, no refinement is found, since its lower bound of uncertainty is not higher than the upper bounds of uncertainty of the outer points. Search window is moved one step to the right.



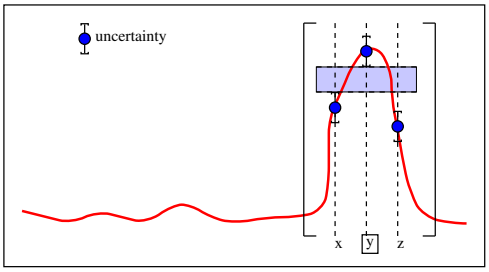
(b) (n=3) Evaluation does not lead to refinement. At this point, all positions did not lead to a refinement, so the number of subintervals is increased by one.



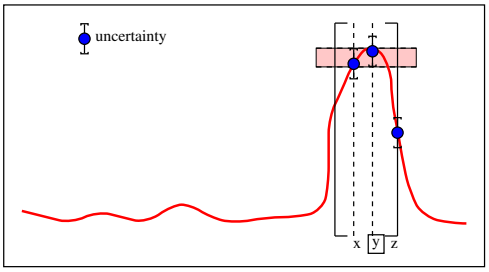
(c) (n=4, skipped two steps) Middle point is located higher than the outer ones. Since its lower bound of uncertainty is higher than the upper bounds of uncertainty of the outer points, a refinement is found.



(d) (n=3, skipped one step) Same as before.



(e) (n=3, skipped one step) Same as before. Note that the distance of the mentioned borders is not very large here.



(f) (n=3, skipped one step) Although the middle point is located higher than the outer ones, no refinement is found, since its lower bound of uncertainty is not higher than the upper bound of uncertainty of the outer points.

Figure 3.5.: An example where the introduced modification of algorithm 3.1 prevents it from "taking a wrong way" to a small local maximum, which may be caused by noise or experimental variance. The blue area depicts the distance between the lower uncertainty bound of the highest point to the upper uncertainty bound of the outer points. The red area depicts the amount of additional distance that would be needed between the uncertainty bounds to lead to a refinement.

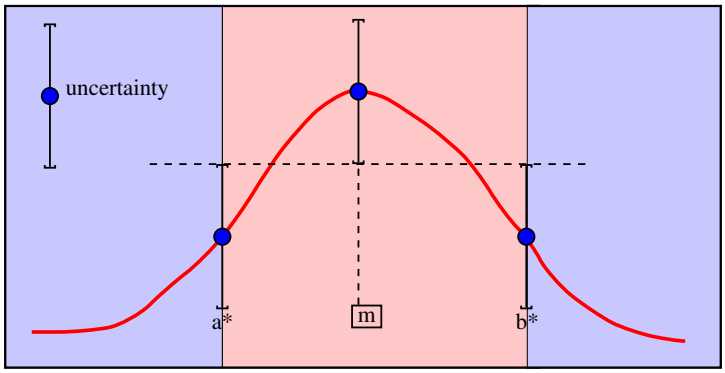


Figure 3.6.: The closest, the search interval borders can get to the actual maximum when the modificated algorithm is used. It is not possible for the algorithm to terminate, when the search interval borders are moved into the red area. In practice, the *tolerance* parameter has to be set at minimum to the width of the red area. Otherwise the algorithm does *not* terminate. Unfortunately, this width is unknown.

vectors, which are distributed with more distance to each other.

One method of comparing order or disorder of datasets, which contain sequence based data, as in the setting of this work, is the normed (Shannon)-*entropy*. It is defined as the expected value of self information of a discrete random variable with possible outcomes $\{x_1, ..., x_n\}$,

$$-\sum_{i=1}^{n} p(x_i) \log_n (p(x_i)) \leq 1, \tag{3.2}$$

where $p(x_i)$ denotes the probability of x_i being the output of the random variable. The latter is normally a source of single characters. The probability of one such character is replaced by the frequency of its appearance in the dataset.

3.7.2. Sequence Based Entropy

This single character approach is not suitable for handling amino-acids. For example, the sequences AABBBB and BABABB have the same entropy (same number of individual amino-acids), but a complete different biological meaning. Therefore, a *sequence* of amino-acids is seen as one outcome of the random variable. Given a (training) set of m amino-acid sequences $T = \{s_1, ..., s_m\}$ over the alphabet $\mathcal{A}$ and a fixed length l with $1 \leq l \leq \min\{|s_1|, ..., |s_m|\}$, the set S of all subsequences of length l in T is given by

$$S = \bigcup_{i=1}^{m} \{s \mid s_i = \alpha s \beta \text{ and } |s| = l \quad \alpha, \beta \in \mathcal{A}^*\}. \tag{3.3}$$

Example: Given the set $T = \{AAA, ABC, CBAA\}$, the set of all subsequences of length l in T is given by

$$\begin{array}{ll} S = \{A, B, C\} & \text{for } l = 1, \\ S = \{AA, AB, BC, CB, BA\} & \text{for } l = 2, \\ S = \{AAA, ABC, CBA, BAA\} & \text{for } l = 3, \\ S = \{CBAA\} & \text{for } l = 4. \end{array}$$

The frequency of appearance of one such subsequence s in T can be calculated by dividing the number of occurrences of s in T by the number of possible occurrences of a string of length $|s|$ in T:

$$F_{s,T} = \frac{\sum_{i=1}^{m} |\{(\alpha, \beta) \mid s_i = \alpha s \beta \quad \alpha, \beta \in \mathcal{A}^*\}|}{\sum_{i=1}^{m} |s_i| - |s| + 1}. \tag{3.4}$$

Example: Given the set $T = \{AAA, ABC, CBAA\}$ and $s = A$, the frequency of s in T is given by (ϵ is the empty string)

$$F_{s,T} = \frac{|\{(AA, \epsilon), (A, A), (\epsilon, A)\}| + |\{(\epsilon, BC)\}| + |\{(CB, A), (CBA, \epsilon)\}|}{3 + 3 + 4} = \frac{3}{5}.$$

For $s = AA$ its frequency in T is given by

$$F_{s,T} = \frac{|\{(\epsilon, A), (A, \epsilon)\}| + 0 + |\{(CB, \epsilon)\}|}{2 + 2 + 3} = \frac{3}{7}.$$

By taking the elements of the set S, given by equation (3.3), as possible outcomes in the entropy term (3.2) and by replacing the probability of appearance by the frequency given by equation (3.4), it is possible to calculate the sequence based entropy of a dataset for a fixed subsequence length l.

Example: Given the set $T = \{AAA, ABC, CBAA\}$ and $l = 2$, the set of all subsequences of length $l = 2$ is given by

$$S = \{AA, AB, BC, CB, BA\}.$$

The frequencies of its elements are:

$$F_{AA,T} = \frac{3}{7}, \qquad F_{AB,T} = \frac{1}{7}, \qquad F_{BC,T} = \frac{1}{7}, \qquad F_{CB,T} = \frac{1}{7}, \qquad F_{BA,T} = \frac{1}{7}$$

The summands of the sequence based entropy are:

$$-F_{AA,T} \cdot \log_5 (F_{AA,T}) = -\frac{3}{7} \cdot \log_5 \left(\frac{3}{7}\right) \approx 0.215.$$

Since the frequency of the remaining four summands is equal, their value is equally given by

$$-\frac{1}{7} \cdot \log_5 \left(\frac{1}{7}\right) \approx 0.173$$

	$AABBBB$	$BABABB$
$l = 1$	0.918	0.918
$l = 2$	0.864	0.960

Table 3.1.: The sequence based entropy takes individual substring "motifs" into account and though is able to state differences between datasets which contain the same number of individual letters. The Shannon-entropy ($l = 1$) is the same for both strings. However, $AABBBB$ seems intuitively "more ordered" than $BABABB$. When using $l = 2$, this condenses in the sequence based entropy, which is lower for $AABBBB$, meaning that through this view, $AABBBB$ contains more redundancy than $BABABB$.

Summing up results in 0.906, which is close to the maximum possible value of one, since T does not contain much redundancy.

Table 3.1 compares the sequence based entropy to the Shannon-entropy for the two sequences from the example in the beginning of this section, namely $AABBBB$ and $BABABB$. The different biological meaning of the two sequences is taken into account by the sequence based entropy in contrast to original one. Note that the Shannon-entropy is a special case of the sequence based entropy with $l = 1$.

3.7.3. Properties

The sequence based entropy has some interesting properties when it is applied to sets of amino-acid sequences:

- Its value converges to one as l goes to $\min\{|s_1|, ..., |s_m|\}$. The probability of appearance of a subsequence in a set of amino-acids sequences goes to zero, as the length of that sequence grows larger. When l is large enough, all sequences only appear once, resulting in a maximum sequence based entropy. The dataset is at maximum disorder from this perspective.
- Its value is equal to the Shannon-entropy when $l = 1$.
- From a biological perspective, not single amino-acids are responsible for the function of a sequence, but subsequence of a certain length, which form "motifs". These motifs may share single characters, but have a different meaning. Consequently, the sequence based entropy will decrease when l is increased from one. When l reaches a certain value, the number of subsequences of that length in the dataset will be at a maximum level. From this perspective, the redundancy is maximal, so the sequence based entropy is minimal. In- or decreasing l will result in less redundancy and a higher sequence based entropy, as described above. Figure 3.7 shows an example of this behavior.

The minimum of all sequence based entropies over all possible sequence lengths is used as a single scalar value of order/disorder of a dataset. This value is used to

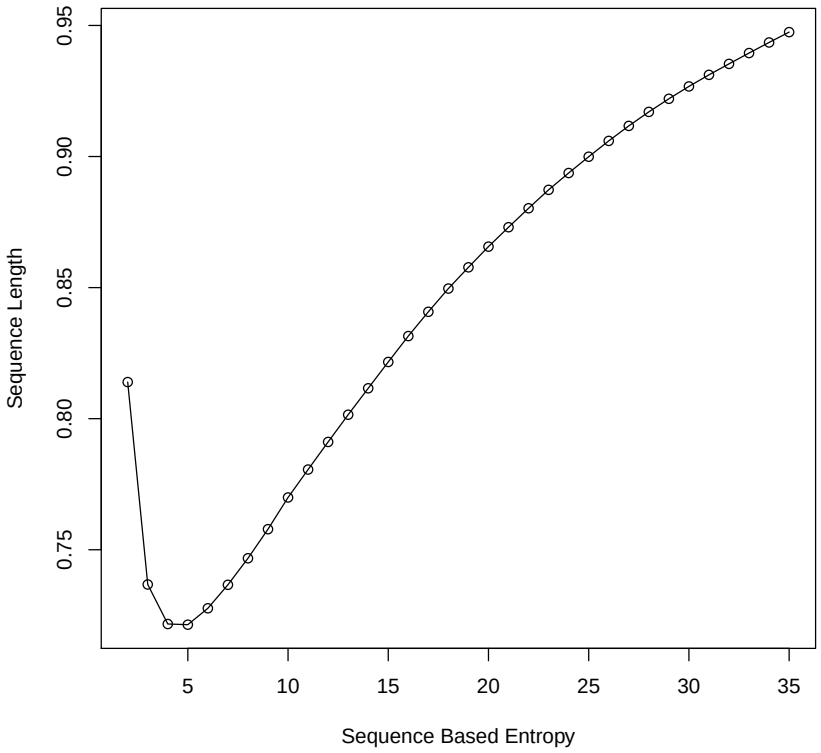


Figure 3.7.: Sequence based Entropy for different subsequence lengths. This dataset has maximum redundancy when using subsequences of length $l = 5$. There is a minimum at this point and its value, 0.72, used to compare this dataset.

compare the datasets and their corresponding results as mentioned in the motivation section 3.7.1.

4. Experimental Results

In the following chapter, all performed experiments and gained results are described. First, in section 4.1, overall leitmotifs of performed experiments and attributes of selected data are described. Then, in section 4.2, results regarding selection of a performance measure for parameter search are given. Afterwards, in section 4.3, introduced methods for experimental variance reduction are verified. Section 4.4 continues with a collection of results, which concern the search for best parameters. Results regarding regularization and kernel parameter and their connections to the underlying feature space are described. Section 4.5 contains a description, comparison and interpretation of results of the overall approach of finding the best classifier for a dataset. Different kernels and datasets are compared. In addition, attributes of datasets are examined for correlations with their corresponding results. Finally, in section 4.6, the introduced bisection based technique for maximizing an unknown function is applied to the search for best regularization parameters. A comparison is made with the naive grid-search, regarding computational costs and differences/similarities of results.

All experiments were performed with the exclusive use of the software, which is described in appendix A. All interpretations and results are solely based on these experiments.

Since this chapter is quite large, every section, which contains an experiment, starts with a brief motivation of the latter and closes with a short summary of gained results.

Also note that all plots concerning regularization parameters have an exponentially scaled horizontal axis, since the exponent to the base of two is varied. Plots concerning kernel parameters have a linearly scaled horizontal axis. The vertical axis of all plots (except a single QQ-Plot, but more in this later) represents a classifier's reached performance measure.

4.1. Experiment Description

4.1.1. Preliminaries: Performance Measures and Variance

Before starting with experiments, a performance measure, which will be used for parameter search has to be selected from the introduced ones: Accuracy, AUC or F-Measure. Therefore, these are tested for applicability by assaying and comparing their individual results. As described in section 3.4, experiments which involve cross-validation suffer from experimental variance. The introduced methods of its reduction are tested: Results gained with multiple evaluation runs are checked for the presumed

normal distribution, which would allow to use their arithmetic mean without the risk of information loss. In addition, results, which are gained *without* repetitions are compared to results, which were gained *with* the use of repetitions.

4.1.2. Main Experiment: Search for Best Parameters

When these subtasks are accomplished, the main experiment may begin, which is to search for classifier parameters, which lead to high quality results. Namely, a search is performed for kernel parameter and the C-SV classifier's margin regularization parameter. This procedure is repeated for all datasets, which will be introduced later in this section, and both introduced kernels.

The search for the best regularization parameter of a C-SV classifier is performed with a fixed kernel parameter. The latter successively takes all values in a search interval as described in section 3.5.2. The regularization parameter is searched in an interval as described in section 3.5.2. In the first step, the search is performed with the naive grid-search technique as described in section 3.5.1. This way, since nearly all parameters in the search interval are tried, a coarse plot of the function, which describes dependencies between regularization parameter and classifier performance can be visualized. These plots are examined regarding connections of their shape to underlying classifier parameters. In addition, since grid-search depends on the assumption that underlying functions are smooth, function plots are used to verify this attribute.

The bisection based search technique, as described in section 3.6 needs continuity to work. It is not possible to derive such an attribute from function plots, but the latter are used to check, if underlying functions are *nearly* continuous. If they *are* nearly continuous, the introduced, faster method is also used for determining best regularization parameters and the results are compared to the ones, which were found by the classical grid-search. In addition, the computational costs of both techniques are compared to verify the cost reduction, which was suggested in section 3.6.4.

As mentioned above, after finding the best regularization parameter for a fixed kernel parameter, the kernel parameter is varied according to section 3.5.2. This results in plots, which depict dependencies between kernel parameter and classifier performance. The underlying functions are definitely *not* continuous, since their parameter is a natural number. In addition, different kernel parameters induce different feature spaces, so subtle changes of the kernel parameter may drastically change the results. Still, an examination of the plots is made. It is for example expected that large kernel parameters for the Spectrum kernel lead to weak results, as suggested in section 2.3.4. Additionally, since kernel parameters of string kernels are responsible for the dimension of the feature space, the results are examined with the latter in mind.

4.1.3. Different Data and Kernels

All of the above described experiments and examination emphases are performed with using both, the Spectrum- and the DS-Kernel. Their individual results are compared.

	A	B	C	D	E	F
#Positives	205	43	648	179	368	309
#Negatives	1151	112	661	82	387	319
#Total	1356	155	1309	261	755	628
$\frac{\#Positives}{\#Negatives}$	0.18	0.38	0.98	2.18	0.95	0.97
Mean length	34	20	232	160	99	240
Length deviation	0.6	0.4	52.0	42.8	1.2	5.6
Min. length	32	19	145	110	99	141
Max. length	38	21	360	462	107	249
Entropy minimum	0.70	0.71	0.96	0.94	0.72	0.73
Used length	3	4	2	3	5	7

Table 4.1.: Attributes of used datasets. For a more detailed view on the distribution of the sequence lengths, see histograms in appendix D. Entropy curves are in appendix E.

The expectation is that the DS-kernel outperforms the Spectrum kernel, according to (Boisvert et al., 2008).

In addition, all experiments are applied to a number of different datasets. These are chosen in such way that a broad spectrum of problems, which are related to classification in bioinformatics, is covered. Results of different datasets are compared regarding their quality, underlying classifier parameters and number of needed support vectors. In addition, attributes of datasets are compared to results that they lead to. Attributes include for example the sequence based entropy as described in section 3.7. The selected datasets are described in the following section.

Table 4.1 shows numerical properties of the selected datasets. The minimum of the sequence based entropy and the used sequence length for calculating it is taken from the curves in appendix E. In the following, the datasets are described briefly. Asymmetric in this context means an asymmetric class label ratio, i.e. much more positives than negatives. Lengths are referred to with terms like “small”, “medium” or “large”. The exact analogies can be found in histograms of sequence length distributions in appendix D.

Dataset A Strongly asymmetric, large size dataset of short sequences that labels a the co-receptor usage of HI viruses, which can be used to classify them as described in (Boisvert et al., 2008).

Dataset B A small asymmetric dataset of very short sequences which describe HI viruses, which are or are not resistant to a drug called Bevirimat. Described in (Heider et al., 2010).

Dataset C Large, nearly perfect symmetric dataset, which labels the positive or negative membership of its elements to a protein class called *small GTPase*. De-

scribed in (Heider et al., 2009). This dataset is very divergent due to the large variety of GTPases. The individual sequences are very long and complex.

Dataset D Protein super-families taken from Interpro[1]. The positive set consists of globins[2], the negative set consists of calycines[3]. Since these are super-families of proteins, they contain a lot of different sequences. Globins have an all-alpha-fold, calycines an all-beta-fold meaning that the three-dimensional structure differs a lot.

Dataset E Nearly symmetric dataset, which labels the resistance of HI viruses to a drug called *RTV*. Described in (Rhee et al., 2006). Medium sequence lengths.

Dataset F Nearly symmetric dataset, which labels the resistance of HI viruses to a drug called *DDI*. This one consists of much longer sequences. Described in (Rhee et al., 2006).

4.2. Different Performance Measures

To decide, which performance measure to use for parameter search, all three introduced measures, namely Accuracy, AUC and F-measure, are compared regarding their applicability for classifier comparison. Therefore, parameter search is performed using all of them and then results are examined to derive the measure, which is most practical for classifier comparison.

4.2.1. AUC

The AUC is not suitable for comparing classifiers when no other information is provided. There are cases when maximizing the AUC leads to parameters, whose resulting classifier is worse than other classifiers on same data. As supposed in section 3.3.2, classifiers with a good AUC may still have a low Specificity, resulting in a lot of samples classified FP. Consequently, comparing classifiers by only using their AUC is *not* a wise strategy. Table 4.2 shows an example of the described problem. The classifier with maximized AUC tends to label all examples positive, resulting in zero FN and thereby a maximum Sensitivity, but a lot FP and thereby a low Specificity. The other classifier does much less errors, even though its Sensitivity is smaller. F-Measure and Accuracy also differ significantly.

4.2.2. Accuracy versus F-measure

As mentioned in section 3.3.1, there might be problems with asymmetric datasets when using Accuracy to compare classifiers. However, in the performed experiments, this was not the case. Both, Accuracy and F-Measure led to results, which were

[1] http://www.ebi.ac.uk/interpro
[2] IPR009050 with known structure
[3] IPR011038 with known structure

	AUC maximized	A better result
Regularization parameter	$2^{12} = 4096$	$2^{0.5} = \sqrt{2} = 1.41$
Kernel parameter	5	2
AUC	**0.9976, [0.9975, 0.9977]**	**0.9938, [0.9937, 0.9939]**
Specificity	**0.5570**	**0.9718**
Sensitivity	1	0.9668
F-Measure	0.9079	0.9767
Accuracy	0.8608	0.9683

Table 4.2.: Maximizing the AUC sometimes sometimes leads to non-optimal results. The AUC of the left example is larger than the AUC of the right one, even though the right classifier is of better quality. Example of dataset D with Spectrum kernel, 500 runs. For the AUC, the 95-% confidence interval is given. Note that the borders do not touch each other and consequently it is unlikely that the AUC difference is caused by noise.

not significantly different. However, the problem is not likely to appear, when the classifiers are of good quality and do little errors, as the case in this work.

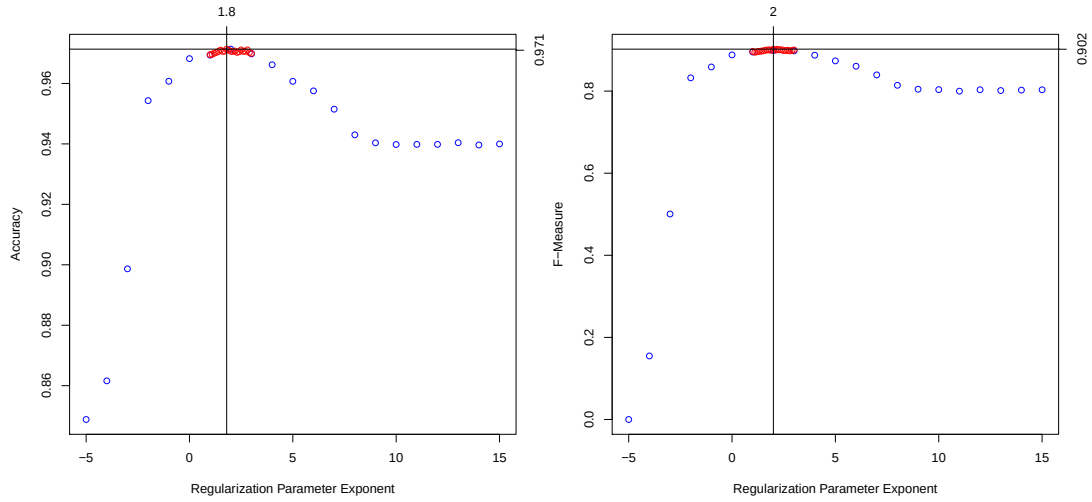
Still, there is a difference: The range of results of each tested regularization parameter differs. The F-measure, which is much more sensible to errors than the Accuracy, quickly becomes zero when the classifier is not of good quality. This already happens if one class is completely classified wrong, in contrast to the Accuracy, which then still takes a value larger than zero. Consequently, the F-measure results in more distinctive search-curve maxima. Figure 4.1 shows an example of the difference. Therefore, the F-Measure is used for the regularization parameter search.

4.2.3. Interim Summary

Since classifiers with a large AUC may be worse than classifiers with a smaller AUC, the AUC is *not* used for classifier comparison in parameter search. Experiments intimate that the two remaining measures, Accuracy and F-measure are both applicable for parameter search. However, since *theoretically* there might be problems with the Accuracy and asymmetric datasets and in addition, the F-measure is more sensible than the Accuracy and leads to more distinctive search-curve maxima, the F-measure is used for parameter search and classifier comparison in the later experiments.

4.3. Experimental Variance: Repetitions and Averaging

Experiments, which involve cross-validation suffer from experimental variance as described in section 3.4. The extend of this variance is examined and visualized. Results of single repetitions of experiments are compared to results of multiple repetitions. Since averaging results of multiple repetitions of the same experiment is possible without information loss, when they are normally distributed, results are examined for this



Accuracy F-measure

Figure 4.1.: The Accuracy is not as sensible as the F-measure. In contrast to the first impression, the results, which are based on the F-measure, are situated in a much wider range and curve maxima are more distinctive. To see that, note that the scaling of the vertical axis differs. Both curves for dataset A with Spectrum kernel and kernel parameter 8.

attribute

Figure 4.2a shows the need for variance reduction to gain plausible results: There is a lot of noise. It is hard to search for maxima in this situation. Consequently, as described in section 3.4, cross-validation is repeated a certain number of times and results are averaged.

According to the central limit theorem, when the number of repetitions grows larger, the distribution of the results of one regularization parameter may be approximated by the normal distribution. Then, the arithmetic mean may be used to represent the results, since it converges to the actual value. This guarantees that no information is lost when averaging results and thereby smoothing the search-curves. Figure 4.2c shows a box-plot diagram, which supports that the results are nearly normal distributed for a large number of repetitions: Most values are symmetrically centered around the mean. Figure 4.2d shows the QQ-Plot[4] of the normal distribution and the distribution of the best found results based on the F-Measure (since this measure is used later on). It shows that the values are nearly exactly normal distributed. The same plot on the base of the Accuracy (omitted here) looks similar. Figure 4.2b depicts that averaged results lead to search-curves, which are much easier to interprete, especially in the context of parameter search.

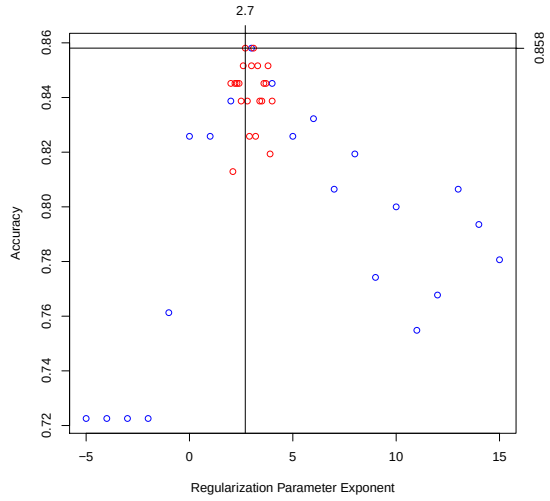
In later experiments, the number of repetitions is set to a large number to be as accurate as possible. Since extends of observed experimental variance is especially large when using datasets of small sizes, experiments on small datasets are repeated a larger number of times ($\sim$ 100 to 200). Experiments on larger datasets are repeated the necessary number of times to approximate their distribution by the normal distribution.

4.4. Parameter Search

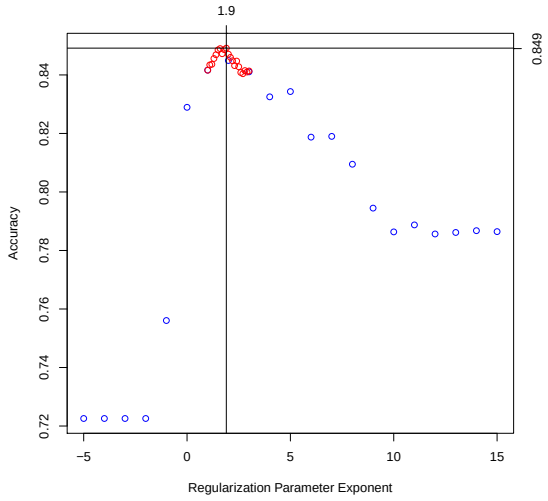
In the following, observations concerning search for best parameters are presented. Two types of parameters need to be determined: Regularization parameters, which influence the margin of the used C-SV classifiers and kernel parameters of the used string kernels, which influence subsequence lengths used by the kernels.

Regularization parameters are searched for exponentially to the base of two, as described in section 3.5.2. Resulting search curves, which are generated by plotting regularization parameter exponent against resulting averaged F-measure, are examined for differences and commonalities, particularly regarding continuity, maxima, overall shape and connections between shape and underlying kernel parameter. The expectation is that the curves are nearly continuous and have only one distinctive maximum (apart from small maxima caused by noise).

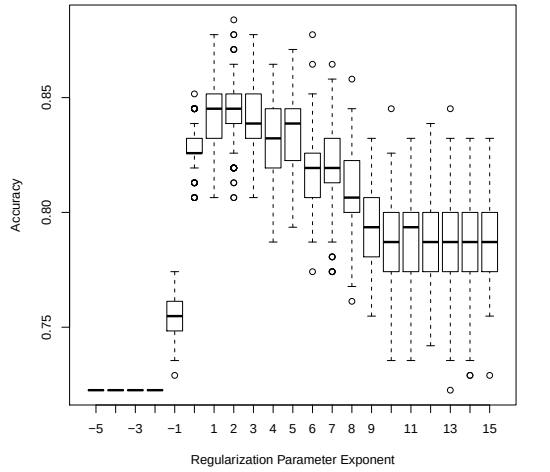
[4] In a QQ-Plot, quantiles of two random variables are plotted against each other. If the distribution of these variables equals, the plot shows a straight line. QQ-Plots, in which a normal distributed variable (theoretical quantiles) is plotted against a random variable (sample quantiles), may be used as a descriptive test for normal distribution. As QQ-Plots are a standard technique, details may be found in any statistics textbook.



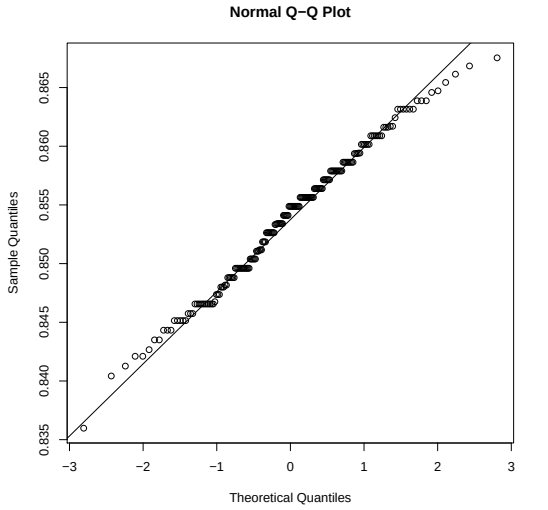
(a) Single experiment



(b) Repeated 200 times



(c) Box-plot for 200 repetitions



(d) QQ-Plot of normal distribution and distribution of results using the best regularization parameter

Figure 4.2.: Depiction of experimental variance and its reduction. Note that the results are nearly normal distributed. Parameter search curve using Accuracy. Based on dataset B with Spectrum kernel with parameter 2. The QQ-Plot is built using the F-Measure results of the best regularization parameter.

Each search for regularization parameter is done with a fixed kernel parameter. Then, the latter are searched linearly with step size one, as described in 3.5.2. Plots are generated by plotting kernel parameter against F-measure. Note that each point in the resulting plots represents the results of a search for regularization parameter. The resulting curves are examined for differences and commonalities, particulary in connection with the results' underlying kernel.

4.4.1. Regularization Parameter

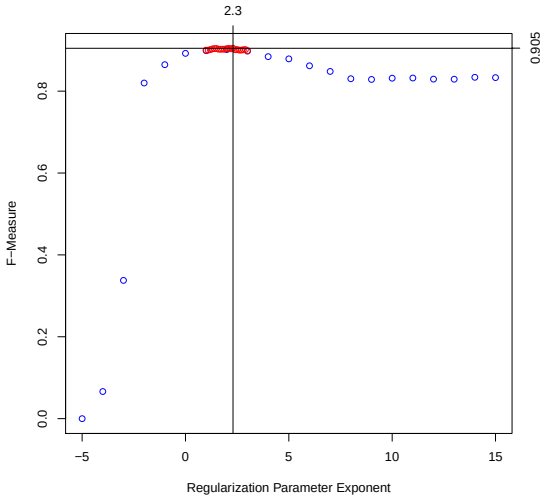
The search-curves of the search for regularization parameter depict the classical trade-off between generalization ability and memorizing data. Figure 4.3 shows selected examples, note that the horizontal axis is scaled exponentially. For low parameter values, resulting classifiers perform weakly. The same holds for larger parameter values (up to a certain point, from which larger parameter values do not change the result anymore. These are discussed later). The parameter, which leads to the best classifier is situated somewhere in between. All observed curves share these attributes. In particular, starting from the maximum of observed search curves, any change of regularization parameter in direction away from the maximum leads to worse results. This observation is well-founded theoretically as mentioned above. The described observations allow the assumption that regularization parameter search-curves only contain *one* distinctive maximum, so one of the requirements for the faster, bisection based search technique is empirically confirmed.

The other requirement, namely, continuity of the search-curves, is also seen as confirmed, since all of the reported search-curves are at least *nearly* continuous. They do not contain large "jumps", especially the fine grid-search reveals a very smooth form.

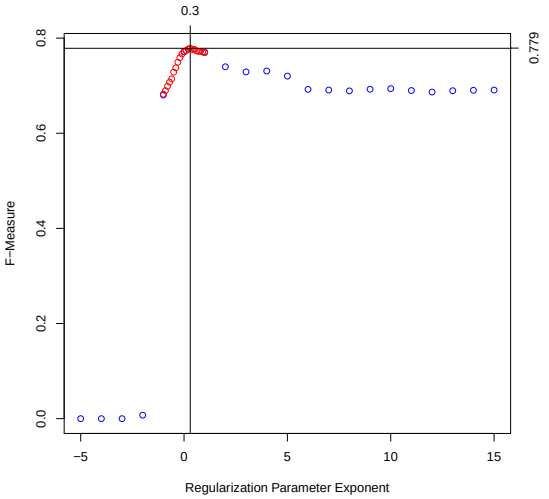
4.4.2. Dimension of Feature Space and Regularization Parameter

When looking at the curves in figure 4.3, it is surprising that at some point, larger values of the regularization parameter do affect the classifier's performance anymore. The curves stay more or less constant from a certain parameter on. Normally, in context of a trade-off variable between generalization ability and extend memorization, larger values lead to worse results from a certain point on.

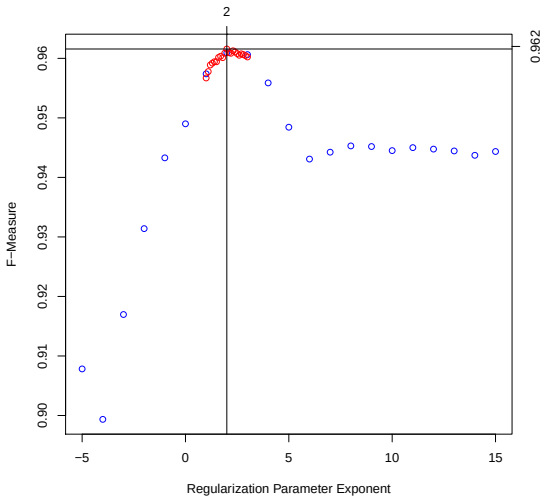
The explanation is the extreme large dimension of the underlying feature space. The larger a feature space gets, the easier it is to find a separating hyperplane with little or even no patterns located in its margin. Consequently, when penalty for patterns on the margin is increased, the hyperplane does not change. Figure 4.4 depicts a search for best regularization parameter for a small and a large kernel parameter. The large one induces a feature space, whose dimension is exponentially larger, than the dimension of the space, which is induced by the small kernel parameter. In feature spaces of larger dimension, the phenomenon that large values of regularization parameters do not affect classifiers anymore already occurs with smaller regularization parameters. Since regularization parameters affect sharpness of the SVM margin, the consequence of this observation is that in extremely large dimensional feature spaces, soft margin



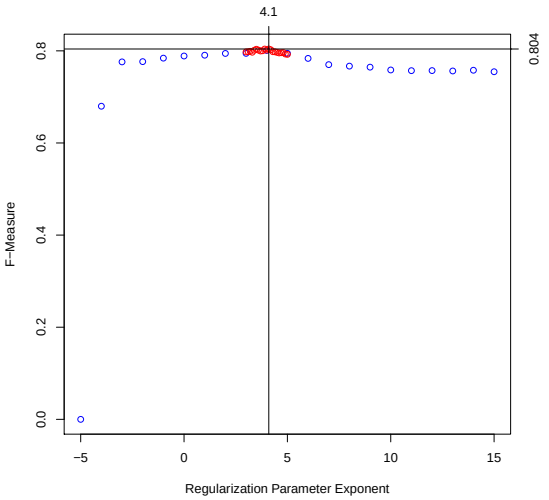
Dataset A, kernel parameter 7



Dataset B, kernel parameter 8

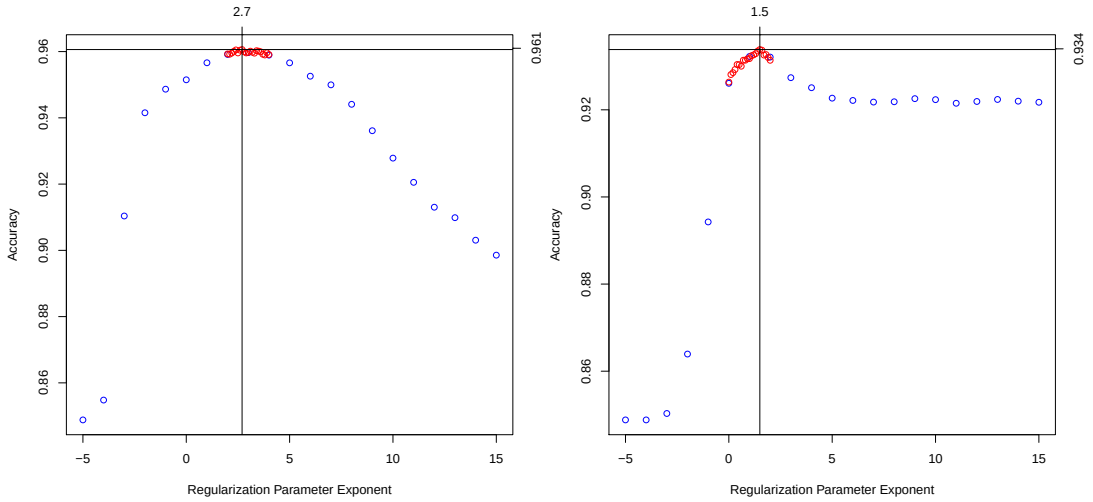


Dataset E, kernel parameter 3



Dataset F, kernel parameter 4

Figure 4.3.: Plotting F-Measure against regularization parameter exponent leads to similar looking search-curves. The used kernel parameter is the one, which leads to the reported best results. The coarse search is colored in blue, the fine search is colored in red. Note that the curves can be regarded as nearly continuous.



Small kernel parameter 2

Larger kernel parameter 12

Figure 4.4.: Depiction, how the dimension of the feature space influences search-curves for regularization parameters (also see text). Parameter search curve using the accuracy. Based on dataset A with the spectrum kernel.

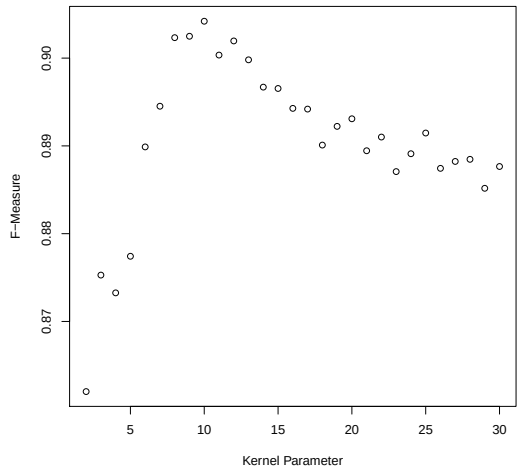
C-SV classification leads to results, which are similar to the ones, gained by using hard-margin classification. This is an interesting observation, since the phenomenon implies a kind of "saturation" of a problem regarding dimension of the feature space.

4.4.3. Kernel Parameters of Different Kernels

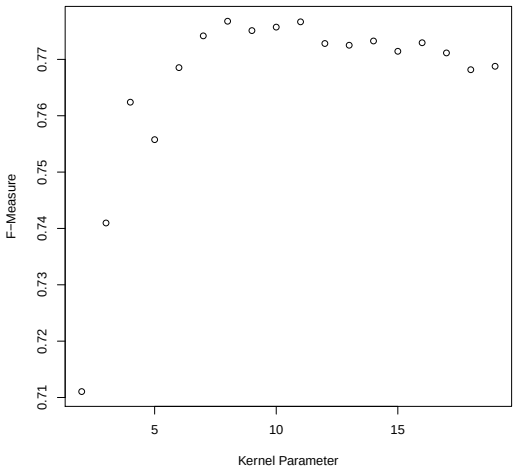
Figure 4.5 shows curves, generated by plotting kernel parameter against classifier performance for datasets A and B. The first observation is that both kernels lead to best classifiers when using a similar kernel parameter, which seems natural because of the similar nature of both kernels. In addition, the curves share certain qualitative attributes: A certain dimension of the feature space, and though a certain size of the kernel parameter, is required for high quality classifiers. At the same time, as the dimension increases further, performance gets worse.

Yet, this is a very coarse view as there are also differences: The Spectrum kernel extremely suffers from larger kernel parameters in contrast to the DS-kernel. To see that, compare the upper and the lower plots of figure 4.5. The results of the Spectrum kernel reach the worst possible result for larger kernel parameters, while the DS-kernel looses some of its quality, but still performs good. To explain this behavior, structure of both kernels has to be compared.

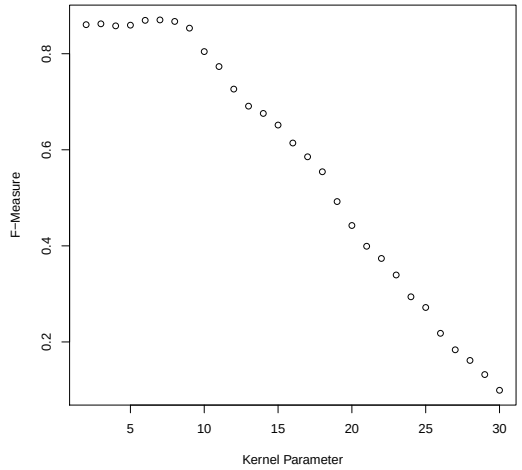
The Spectrum kernel counts subsequences of the length given as kernel parameter. When this length grows larger, it is more unlikely finding two elements in the underlying dataset that share subsequences of that length. Consequently the resulting feature



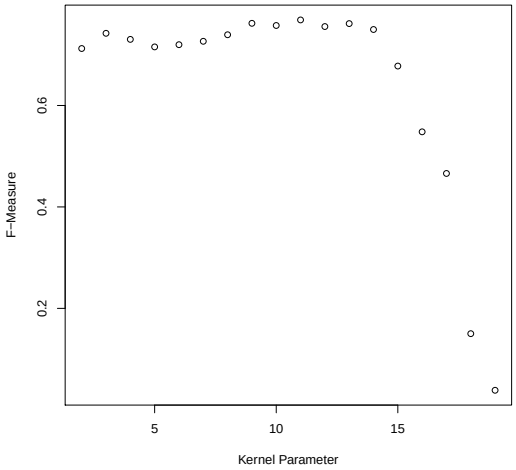
Dataset A, DS-Kernel



Dataset B, DS-Kernel



Dataset A, Spectrum Kernel



Dataset B, Spectrum Kernel

Figure 4.5.: Curves, generated by plotting kernel parameter against F-measure of the resulting best classifier. Note that every point in the plot includes a search for best regularization parameter. The performance of the Spectrum kernel is extremely weak when using large kernel parameters. To see that, note that the vertical axis of the lower plots is scaled differently than the vertical axis of the upper plots.

vectors become more and more sparse while their dimension increases, resulting in a dot-product that converges to zero. The kernel then is not anymore able to "perceive" any structure in the dataset. This is a dilemma: A larger dimension of the feature space is *needed* to model relations in complex data while increasing the dimension, by increasing the kernel parameter, results in worse performance caused by nature of the kernel.

In contrast, the DS-kernel uses subsequences *up to* a certain length. When the kernel parameter is increased, all subsequences, which are shorter, are still represented in the underlying feature vectors. However, as the dimension increases, performance also gets worse. The explanation here is more likely to be over-fitting of data. The risk of over-fitting increases with increasing dimension of the underlying feature space.

4.4.4. Interim Summary

Search curves for the regularization parameter are *nearly* continuous and only have *one* distinctive maximum. All observed search-curves for regularization parameters share these attributes. In addition, these observations are supported by several theoretical reasons. As a consequence, the bisection based search technique as described in section 3.6 may be applied to search for regularization parameters to improve its quality and to reduce its costs. The results for this are described later.

From a certain (large) regularization parameter, any further increment does *not* change the resulting classifier's quality anymore. Consequently, from this points on, no patterns are situated on the margin and therefore hard- and soft-margin do not lead to different results. This phenomenon happens "earlier", meaning at smaller values of regularization parameters, when the dimension of the underlying feature space is larger. The classification problem is earlier "saturated" regarding the dimension of the feature space.

All observed search-curves of kernel parameters share attributes: A certain size of kernel parameters is needed to build classifiers of good quality. In contrast, when the kernel parameter is overly large, performance gets worse. Consequently, the resulting curves have only one distinctive maximum. Unfortunately, since they are not continuous, the bisection based search technique cannot be applied without further modifications, which are out of scope of this work.

The kernel parameters, which lead to best classifiers are not equal, even-though simliar for both kernels. In contrast, their results behave differently when the parameter grows: The Spectrum kernel extremely suffers from large parameters, while requiring a certain size of the latter to perform well. This is a dilemma, since these requirements are in conflict. In contrast, the DS-kernel does not suffer from larger kernel parameters, except for over-fitting problems in large dimensional feature spaces.

4.5. Different Kernels, Different Datasets

In the following, results of the best classifiers, which are found both with the Spectrum and the DS-Kernel, are given. The expectation is that the DS-kernel outperforms the

	A	B	C
Spectrum Kernel			
Kernel Par.	7	11	3
Reg. Par.	$2^{1.6} = 3.03$	$2^{0.5} = 1.41$	$2^{-0.3} = 0.81$
F-Measure	$\mathbf{0.8693} \pm 0.0004$	$\mathbf{0.7656} \pm 0.0016$	$\mathbf{0.9968} \pm 0$
Accuracy	0.9626 ± 0.0004	0.8751 ± 0.0009	0.9969 ± 0
AUC	0.9705 ± 0.0004	0.9180 ± 0.0010	0.9983 ± 0
Sensitivity	0.8230 ± 0.0010	0.7348 ± 0.0018	0.9937 ± 0
Specificity	0.9875 ± 0.0004	0.9289 ± 0.0011	1 ± 0
$\frac{\#SV}{P+N}$	$\frac{400}{205+1151} = \frac{400}{1356} = 0.27$	$\frac{155}{43+112} = \frac{155}{155} = 1$	$\frac{514}{648+661} = \frac{514}{1309} = 0.39$
Distant Segments Kernel			
Kernel Par.	10	8	3
Reg. Par.	$2^{2.4} = 5.28$	$2^{0.4} = 1.32$	$2^{-0.1} = 0.93$
F-Measure	$\mathbf{0.9032} \pm 0.0004$	$\mathbf{0.7764} \pm 0.0016$	$\mathbf{0.9976} \pm 0$
Accuracy	0.9715 ± 0.0001	0.8786 ± 0.0009	0.9977 ± 0
AUC	0.9769 ± 0.0002	0.9084 ± 0.0020	0.9988 ± 0
Sensitivity	0.8800 ± 0.0007	0.7597 ± 0.0020	0.9953 ± 0
Specificity	0.9878 ± 0.0001	0.9242 ± 0.0009	1 ± 0
$\frac{\#SV}{P+N}$	$\frac{371}{205+1151} = \frac{371}{1356} = 0.27$	$\frac{114}{43.112} = \frac{114}{155} = 0.74$	$\frac{371}{648+661} = \frac{371}{1309} = 0.28$

Table 4.3.: Best obtained classifiers for both kernels and datasets A, B and C. With a 10-fold cross-validation, maximized using F-Measure and sharpened by 500 repetitions. 95-% confidence intervals.

Spectrum Kernel, as described in (Boisvert et al., 2008). In the first instance, all results are simply stated.

What follows is an analysis and comparison of selected results regarding different datasets. It is out of scope of this work to analyze all collected data, therefore only partial results, which are of interest in the context of this work, are considered.

A comparison of dataset B with dataset D, with special attention on common and uncommon attributes being the cause for different performance quality, is made. A similar comparison is made regarding dataset E and dataset F, since these lead to different results as the first two. Finally, as it leads to excellent results, a brief description of dataset C's results is given.

4.5.1. Best Found Classifiers

Tables 4.3, 4.4 and 4.5 list results, which are obtained on all used datasets with both kernels.

	D	E	F
Spectrum Kernel			
Kernel Par.	2	3	4
Reg. Par.	$2^{0.5} = 1.41$	$2^{2.3} = 4.92$	$2^{3.8} = 13.93$
F-Measure	$\mathbf{0.9766} \pm 0.0002$	$\mathbf{0.9611} \pm 0.0002$	$\mathbf{0.8023} \pm 0.0007$
Accuracy	0.9683 ± 0.0003	0.9625 ± 0.0002	0.8031 ± 0.0004
AUC	0.9938 ± 0.0002	0.9882 ± 0.0001	0.8880 ± 0.0004
Sensitivity	0.9667 ± 0.0004	0.9501 ± 0.0004	0.8119 ± 0.0004
Specificity	0.9718 ± 0.0007	0.9741 ± 0.0003	0.7945 ± 0.0010
$\frac{\#SV}{P+N}$	$\frac{102}{179+82} = \frac{102}{261} = 0.39$	$\frac{162}{368+387} = \frac{162}{755} = 0.21$	$\frac{329}{309+319} = \frac{329}{628} = 0.52$
Distant Segments Kernel			
Kernel Par.	4	3	3
Reg. Par.	$2^{2.8} = 6.96$	$2^{2.9} = 7.49$	$2^{3.5} = 11.31$
F-Measure	$\mathbf{0.9861} \pm 0.0001$	$\mathbf{0.9605} \pm 0.0002$	$\mathbf{0.8000} \pm 0.0006$
Accuracy	0.9810 ± 0.0002	0.9619 ± 0.0002	0.8003 ± 0.0006
AUC	0.9985 ± 0.0001	0.9878 ± 0.0001	0.8888 ± 0.0003
Sensitivity	0.9832 ± 0.0002	0.9504 ± 0.0002	0.8117 ± 0.0008
Specificity	0.9763 ± 0.0005	0.9728 ± 0.0003	0.7892 ± 0.0009
$\frac{\#SV}{P+N}$	$\frac{171}{179+82} = \frac{171}{261} = 0.66$	$\frac{151}{368+387} = \frac{151}{755} = 0.20$	$\frac{339}{309+319} = \frac{339}{628} = 0.54$

Table 4.4.: Best obtained classifiers for both kernels and datasets D, E and F. With a 10-fold cross-validation, maximized using F-Measure and sharpened by 500 repetitions. 95% confidence intervals.

	A	B	C	D	E	F
Better Kernel	DS	DS	DS	DS	BS	BS
Difference	0.0330	0.0108	0.0007	0.0095	0.0006	0.0023

Table 4.5.: Best kernels of each dataset and differences to the results of the other kernel. Note that the differences are not overly large. Based on tables 4.3 and 4.4.

4.5.2. Kernel Comparison

As described in (Boisvert et al., 2008), the Distant Segments Kernel usually performs better than the Spectrum kernel. The difference is statistically significant with a 95% confidence level in all cases.

However, except for one case, numerical differences between reached results are never larger than 0.011 regarding F-Measure and 0.02 regarding Sensitivity/Specificity. The DS-Kernel does about 1% to 2% more correct predictions than the Spectrum Kernel. This difference comes at some minor computational costs since time complexity to compute the DS-Kernel is quadratic regarding string length. Time complexity of the Spectrum kernel is linear. However, both are polynomial.

As mentioned above, an exception is dataset A where the difference regarding F-Measure is 0.03. The reason for this is mainly the 5% Sensitivity difference, meaning that the Spectrum Kernel does more FN. This is no surprise since the class label ratio of dataset A is $\frac{205}{1151} = 0.18$, meaning there is little information regarding positive patterns. It it noteworthy that dataset A contains the same kind of data as these, which were used in (Boisvert et al., 2008), in which the DS-kernel is introduced. This allows the suspicion that it is especially suitable for similar kinds of classification tasks.

Dataset E and dataset F form an exception: The Spectrum kernel leads to better results. The numerical difference is also minor, however, this illustrates that the DS-kernel is *not* a better choice *in general*. One explanation might be that the kernel parameter, which leads to best results in both datasets, is small. The structural differences of the DS-kernel and the Spectrum kernel are less distinctive when using a small kernel parameter. In particular, when the kernel parameter is very small, the impact of single letters to the DS-kernel gets larger. From a bioinformatics perspective, single letters are *not* very relevant for function of a sequence, so these single letters distort the DS-kernel. When the kernel parameter grows larger, the impact of this distortion becomes minor.

When performing classification on sequence based data, multiple kernels should be considered and compared, if possible. If the latter is not possible, this does not lead to serious problems, as the area, where results are situated is similar for both kernels.

The reason why the DS-kernel performs better in most cases is likely to be caused by biological reasons: The three-dimensional structure of a protein is strongly responsible for its function. For example, bonds to other proteins depend on a suitable "shape" of parts, which are responsible for connections. A protein's three-dimensional structure is influenced by the underlying sequence. The so called *fold* is the spatial arrangement of an amino-acid chain. Proteins tend to fold in such way that the overall energy caused by all involved physical and biochemical processes is minimal. In certain proteins, the amino-acid chain forms a loop and a certain part of the chain is situated closely to another certain part. Consequently, the distance of two parts, which are responsible for a loop is an important factor for a protein's three-dimensional structure, therefore, also for its function and though for its class label.

The DS-kernel takes the distance of subsequences into account and is though able to

	B	D	B	D
	Distant Segments Kernel		**Spectrum Kernel**	
Kernel Par.	8	4	11	2
Reg. Par.	$2^{0.4} = 1.32$	$2^{2.8} = 6.96$	$2^{0.5} = 1.41$	$2^{0.5} = 1.41$
F-Measure	0.7764	0.9861	0.7656	0.9766
Accuracy	0.8786	0.9810	0.8751	0.9683
AUC	0.9084	0.9985	0.9180	0.9938
Sensitivity	0.7597	0.9832	0.7348	0.9667
Specificity	0.9242	0.9763	0.9289	0.9718
$\frac{\#SV}{P+N}$	$\frac{114}{43+112} = 0.74$	$\frac{161}{179+82} = 0.66$	$\frac{155}{43+112} = 1$	$\frac{102}{179+82} = 0.39$

Table 4.6.: Comparison of Dataset B and D. Based on tables 4.3 and 4.4.

model sequence similarities, which may have an impact on three-dimensional structure. These are "out of sight" for the Spectrum kernel.

4.5.3. Datasets B and D: Different Disorder and Classification Severity

In the following, results of dataset B and dataset D are compared, since gained results comply with dataset attributes, which are kind of "inverses" of each other.

Table 4.6 shows the results for dataset B and D. These are the smallest tested datasets. Dataset B, which contains HIV sequences, is very homogeneous: The sequence length difference is only one character. In addition, the sequence based entropy minimum of the dataset equals 0.71, which is much lower than 0.94 of dataset D. Note that the class label ratio of both datasets is kind of "mirrored". Dataset D, which contains two protein super-families, has a huge range of different sequence lengths. Vividly, this results in a more or less disjunct distribution of dataset D's feature vectors in the underlying feature space, in contrast to dataset B, where the distribution is more overlapping. This is supported by the number of support vectors needed to express the class relations of both datasets: Dataset B needs more feature vectors than dataset D, especially when using the Spectrum Kernel. Intuitively, when data is distributed overlapping, a larger number of support vectors is needed to build a separating hyperplane.

Both datasets are classified best with the DS-Kernel, however, results do not numerically differ a lot. The best classifier for dataset B has a Specificity of 92% to 93%. The Sensitivity is at 76% to 73%, which means that the classifier tends to output "positive" too often. A reason for this could be the small number of positive examples in dataset B, only 43. However, the results for dataset D interferes with this explanation, since the number of negative examples is also small, while the Sensitivity is at 92%. Dataset D is classified nearly perfect. About 97% to 98% of answers are correct for both classes.

The optimal kernel parameter for both datasets differs. Dataset B leads to a larger kernel parameter than dataset D and therefore needs a feature space of larger dimen-

	E	F
	Spectrum Kernel	
Kernel Par.	3	3
Reg. Par.	$2^{2.3} = 4.92$	$2^{3.8} = 13.93$
F-Measure	**0.9611**	**0.8023**
Accuracy	0.9625	0.8031
AUC	0.9882	0.8880
Sensitivity	0.9501	0.8119
Specificity	0.9741	0.7945
$\frac{\#SV}{P+N}$	$\frac{162}{368+387} = 0.21$	$\frac{329}{309+319} = 0.54$

Table 4.7.: Comparison of Dataset E and F. Based on tables 4.3 and 4.4.

sion for good results. This leads to the assumption that dataset B contains more complex class relations. The number of needed support vectors support this assumption: Dataset B needs the maximum number of SV. This means that data is memorized as strongly as possible. Normally, such behavior leads to poor generalization ability, however, the results on unknown data are still good, so the extend of memorization has not reached a level where classification quality gets worse. This corresponds with the regularization parameter, which leads to the latter results: It is quite small, so the margin of the separating hyperplane is large and so is the classifier's generalization ability.

The homogeneity of dataset B and the divergence of dataset D are candidates to cause differences of the presented results. In addition, the latter support the supposition that the extend of disorder of data, represented by the minimum of the dataset's sequence based entropy and its sequence length variance, is a first indicator how well a string-kernel equipped SVM performs on such data.

4.5.4. Datasets E and F: Similar Disorder and Classification Severity

In the following, datasets E and F are compared, since they lead to different results, while at the same time, their attributes are similar.

Table 4.7 shows the results. These seem to be similar: Both are nearly perfect symmetric and have a similar size. In contrast, the length of individual sequences differs: The mean sequence length is 99 for dataset E and 240 for dataset F. The length deviation of both datasets is about 1% to 2% of the mean length, so both datasets are homogeneous regarding their sequence length. The minimum of the sequence based entropy is also similar with ~ 0.7, which means that both datasets contain certain "motifs" that are used multiple times.

However, in contrast to their attributes, results are different to the rest of the performed experiments in various ways. The first difference is that the Spectrum Kernel performs better than the Distant Segments Kernel on both datasets. The difference is numerically small, but still significant with 95% confidence. Both datasets

contain drug resistance information of HIV. Note that dataset B also contains this kind of information but the DS-Kernel works better there.

Second, the datasets' individual results strongly differ numerically, although both datasets are quite homogeneous regarding their extend of disorder. The F-Measure difference is about 0.16, caused by about 14% less Sensitivity and about 18% less Specificity on dataset E. Since none of the introduced attributes of the datasets is able to explain these differences regarding classifier quality, no statement may given in this work. The difference is likely caused by biological reasons. However, during discussions with the advisors of this work, no obvious explanation could be found. A consequence is that if a dataset is homogeneous regarding its sequence based entropy minimum, the underlying classification task is compelled to be severe.

Another observation is that the best classifier for both datasets leads to a small quotient of support vectors and number of training patterns. Especially for dataset F, this is surprising, since the performance is rather bad, while at the same time not all resources (ratio of SVs) are used. An explanation might be that a larger number of SVs would better fit the data, but at the same time, generalization ability of the classifier would be reduced, resulting in more errors in classification and therefore to its rejection during the search for best parameters. This leads to a possible reason for the bad performance on dataset F: The distribution of the data is strongly overlapping in the underlying feature space and more accurate fitting would result in more classification errors.

Another resource, which may be used to increase classifier quality, is the dimension of the underlying feature space. However, when working with dataset F, the dimension, which leads to the best results, is small. An explanation might be the best found regularization parameter, which is quite large with ~ 14. Large regularization parameters cause a thinner margin and therefore less generalization ability. A features space with larger dimension (and also a larger number of SVs) would cause stronger fitting of data. But already in a feature space with a *low* dimension, a *large* regularization parameter leads to best results. If the dimension of the underlying feature space would be larger, the generalization ability would be even worse.

4.5.5. Dataset C: Nearly Perfect Results

Since dataset C is classified nearly perfect, a view on these results and the dataset's attributes is given in the following. Table 4.8 shows the results.

The dataset is strongly divergent and was selected with this attribute in mind. The minimum of its sequence based entropy is the largest of all datasets with 0.96. It also has the largest deviation of individual sequence lengths (see also histogram in appendix D). Consequently, resulting feature vectors are distributed extremely afar in the underlying feature space. These kind of classification tasks are "easy" to solve, since a hyperplane with a large margin may be found. Parameters of the best found classifier support this: The regularization parameter is very small, which suggests that there are only few margin errors, resulting in a large margin and therefore strong generalization ability of the classifier. In addition, the best found kernel parameter is

	C
	Distant Segments Kernel
Kernel Par.	3
Reg. Par.	$2^{-0.1} = 0.93$
F-Measure	0.9976
Accuracy	0.9076
AUC	0.9977
Sensitivity	0.9988
Specificity	0.9953
$\frac{\#SV}{P+N}$	$\frac{514}{648+661} = 0.39$

Table 4.8.: Results of Dataset C. Based on table 4.3.

small. It is already possible to model the dependencies in the data in a comparatively low dimensional feature space.

These results support the supposition that datasets with a large extend of disorder form the easiest classification problems for SVM.

4.5.6. Interim Summary

Since, one of the intends of this work is to test the introduced approach on real-life data, the first statement is of general nature: Using the approach on real-life data leads to very good results. Some datasets are classified nearly perfectly (dataset C and D) or very accurate (dataset A and E). The worst result (dataset B and F) is still of practical relevance, since still $\sim$ 75% to 80% of its content is classified correctly. Since used datasets cover a wide range of problems, the approach qualifies itself as a candidate for further research.

The expectation that the DS-kernel is better than the Spectrum kernel is "half fulfilled". On the one hand, the DS-kernel leads to better results on more datasets, on the other hand, the numerical differences are not very distinctive. An exception for this is dataset A, where the difference is larger.

Theoretically, both kernels become more similar as the used kernel parameter gets smaller. In addition, when the kernel parameter is very small, the DS-kernel is distracted by single letters, which then have a large impact on the results, while not being relevant for sequence function. In general, the DS-kernel is more flexible than the Spectrum kernel, since it does not have problems with large kernel parameters. Also, it was argued that distance of certain subsequences, as taken into account by the DS-kernel, has an impact on three-dimensional structure and therefore on function of an amino-acid chain.

A large value of the sequence based entropy, which is a measure of a dataset's disorder, is an indicator for an "easy" classification problem on all examined datasets. The inversion of that argument is not true in general, as there are cases where two datasets with a similar entropy value form classification tasks of different severity

(datasets E and F).

4.6. Bisection Based Search for Regularization Parameters

In section 3.6, an algorithm was described, which is guaranteed to find an exclusive maximum of a continuous function in a given interval and with a given accuracy. The assumptions made regarding the function, which describes the dependency between the regularization parameter and the quality of the classifier are fulfilled in the setting of this work: All presented (and also all observed) curves have only one distinctive maximum and are nearly continuous except some small part of noise (see figures 4.3, 4.4 or 4.1 for examples). Note that this is an information, which is gained empirically and is not valid in general. However, it was argued that there are theoretical reasons for the resulting curve to have these attributes.

The described bisection based optimization of the grid-search was developed at a later point of this work, so there only was time to test the procedure in context of searching for regularization parameters with a fixed kernel parameter.

Figure 4.6 shows a bar plot of the number of classifier evaluations, which were done in the search for the best classifier and depicts that the introduced method is about twice as fast as the naive grid-search (with naive parameter selection $\epsilon = 0$, $tolerance = 0.5$). Note that this cost reduction factor is reached when using the introduced method for searching for *one* parameter. When it is applied to searching for multiple parameters, this factor is multiplicated with itself for every additional parameter, becoming k^n, where k is the mentioned factor for one parameter and n are the number of parameters. A further examination of the process and optimizations to algorithm 3.1 could lead to better factors. However, this is out of scope of this work.

Figure 4.7 compares search curves of the two introduced techniques. Note that the results are nearly the same, while the faster method's curve consists of less "evaluation points". All observed results were nearly the same but are omitted due to limited space.

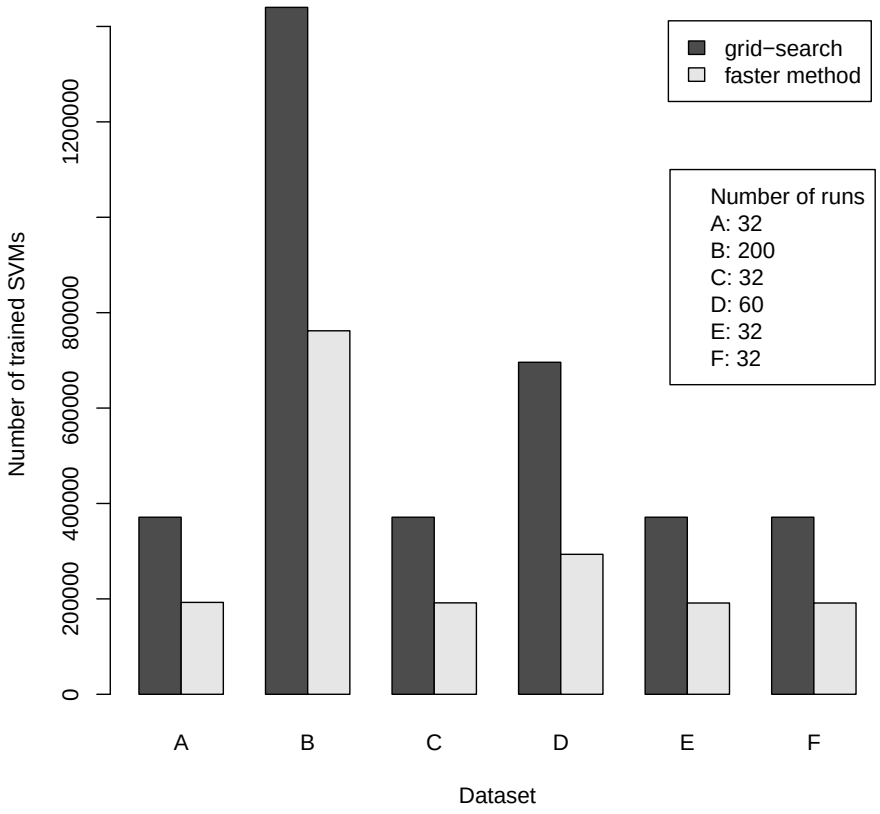
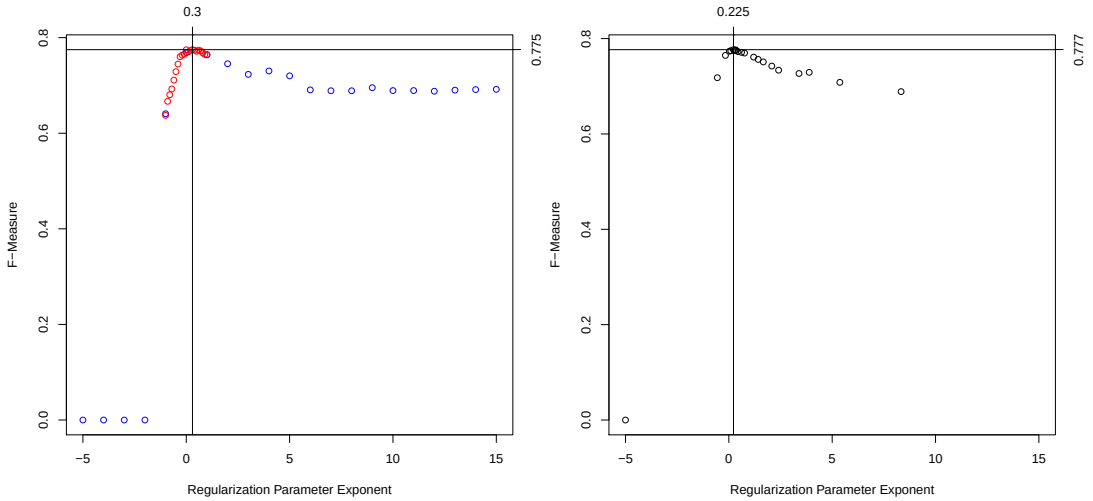
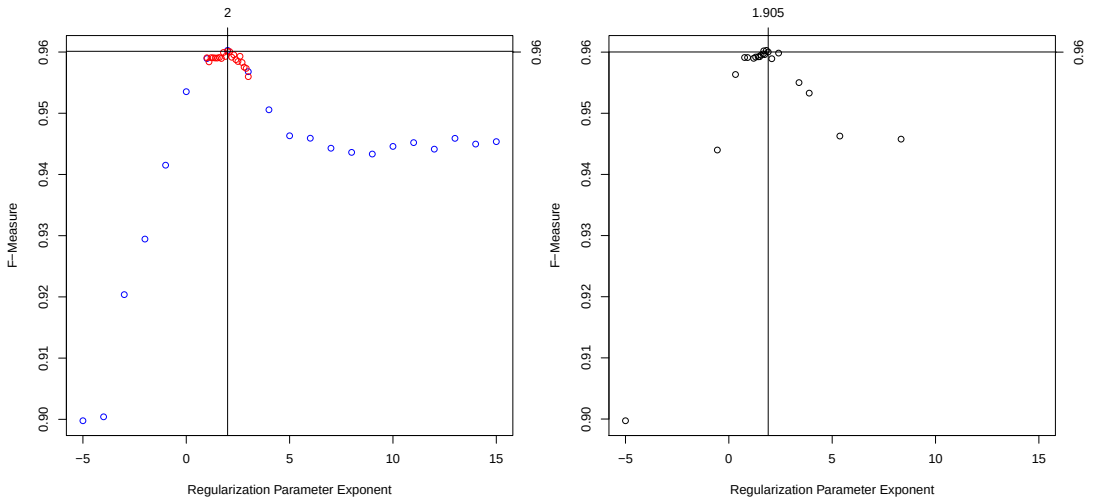


Figure 4.6.: Bar plot, which shows the number of classifier evaluations during the search for the best classifier. Note that only the search for regularization parameters is performed with the introduced bisection based method. The individual height of bars for different datasets differs because the number of repetitions(runs) is chosen differently for each dataset.



(a) Dataset B, kernel parameter 7, grid-search (b) Dataset B, parameter 7, bisection based search



(c) Dataset E, kernel parameter 5, grid-search (d) Dataset E, kernel parameter 5, bisection based search

Figure 4.7.: Classical grid-search compared to the introduced method. The number of points in the plot equal to the number evaluations needed to find the best regularization parameter using a specific, fixed kernel parameter. Note that there are less evaluation points on the right side, while the results are nearly the same. Left is the grid-search. Both with DS-kernel.

5. Summary and further work

In this last chapter, all results of this work are collected. Then, further possible fields of work are described.

5.1. Summary

In this work, an approach for binary classification of sequence based, biological data using kernel methods was described. The approach operates on top of a strong and well researched theoretical fundament, Support Vector Machines. State-of-the-art string-kernels were used, namely the Spectrum Kernel and the Distant-Segments Kernel.

Different methods for measuring performance of a classifier were introduced. To only report results, which are statistically relevant, it was argued how to shrink experimental variance, how to average results and how to obtain tight confidence intervals. These averaged results were used for parameter search. There were two kinds of parameters to search for: Regularization parameters concerning SVM-margin and kernel parameters concerning lengths of subsequences. The classical grid-search was described and shown to have some deficiencies regarding computational costs and accuracy. A bisection based, faster and more accurate method for maximizing an unknown function was described, shown to be correct and applied to search for regularization parameters. To measure disorder of a sequence based dataset, along with length distribution of a dataset's individual sequences, an entropy based measure, the sequence based entropy, was described.

A software was implemented on the base of SHOGUN. It performs all described search techniques on given datasets and produces a large amount of data on the fly. The architecture is layered, so any component may be replaced or used in another context.

In general, usage of string-kernel equipped SVM was reported to be a practical method for performing classification tasks on sequence based, biological data. Most of the results were of very good or even near perfect quality. There were two exceptions where the approach only led to "good" results. However, the latter are still usable from a practical perspective. In addition, the following results were obtained:

- Of the examined standard single-scalar performance measures, the F-measure proved to be the most robust one to compare classifiers. Consequently, it was maximized during the search for best parameters.
- Experimental variance, which expresses in noisy parameter search-curves, was successfully reduced by repeating single evaluations multiple times. Results of

these repetitions were empirically shown to be normal distributed, consequently, using their arithmetic mean does not distort the final result.

- While searching for regularization parameters, an interesting phenomenon was observed regarding dimension of the underlying feature space: As its dimension grows larger, from a certain point on, larger regularization parameters do not affect performance of a classifier anymore. The larger a feature space dimension is, the earlier this happens. Consequently, since the used regularization parameter affects margin-sharpness, at this point, no feature vectors are situated on the margin and therefore hard-margin and soft-margin SVM lead to similar results.

- The two kernels that were compared regarding conceptual differences and their impact on experimental results: The Spectrum kernel suffers from large kernel parameters on the one hand, but since kernel parameters are responsible for the dimension of the underlying feature space, it needs a kernel parameter of a certain size to perform well on complex data. From this can be concluded that the Spectrum kernel should not be used when complexity of data is presumed to be overly large.

  In contrast, the DS-kernel does not conceptually suffer from large kernel parameters. Its performance also gets worse as the dimension of the feature space grows larger, but the reason for this is more likely to be over-fitting, which rather happens in a feature space of larger dimension.

- In contrast to the Spectrum kernel, the DS-kernel takes the distance between subsequences into account. Since in biological data, the distance between these subsequences has an impact on three-dimensional structure of a sequence, the DS-kernel is better suited for biological applications. Results were given for all datasets and kernels. Except for two exceptions, the DS-kernel performed better than the Spectrum kernel. However, the numerical differences of the reached results are not large, except for one case.

  The numerical difference also was minute in the cases where the Spectrum kernel led to better results. In addition, in these cases, the kernel parameter, which led to best results was very small. Since the DS-kernel takes subsequences *up to* a certain length into account, when small kernel parameters are used, single letters form a larger part of all considered subsequences. However, single letters are biologically not this relevant and therefore they "distract" the DS-kernel, when a small kernel parameter is used.

- A look a dataset's attributes gives a first hint, how well the described approach performs on it. The disorder of a dataset correlates with high performance quality. Datasets, which are strongly divergent are easier to classify than datasets, which are homogeneous. Measures of order, which were applied included: The distribution of individual sequence lengths (see histograms in appendix D), including their mean, minimum, maximum and deviation. In addition, high values of the minimum of the described, biologically motivated sequence based entropy

of a dataset are an indicator for disorder and therefore for an "easy" classification task. Low values of this minimum indicate a more homogeneous and though harder to classify dataset. However, similar entropy minima did not compulsory lead to similar results.

- Grid-search was shown to have drawbacks regarding correctness of its results and computational costs. A new method, which is based on bisection, was described and shown to be totally correct when certain requirements are met. In addition, it reduces the linear computational costs of grid-search to logarithmic costs.
- Search for best regularization parameters resulted in search-curves, which contain one distinctive maximum and which are at least *nearly* continuous. (Continuity cannot be shown empirically.) Therefore, all requirements for applying the introduced bisection based method to search for best regularization parameters, are met.
- In practice, the bisection based method was empirically shown to be around twice as fast as naive grid-search for one parameter. In addition, parameter choice of grid-search drops and therefore the risk that a maximum is "overseen", or that computational costs explode. In practice, the bisection based method's advantages regarding accuracy, did not affect the results, since grid-search parameters were chosen in such way that the step size was small.
- The bisection based method may be extended to be robust against small local maxima, or noise. Its total correctness then becomes partial correctness, since termination is not guaranteed anymore. However, this may still be useful, when a suitable uncertainty bound is given as parameter and therefore the method is *likely* to terminate. Besides, any non-termination may be overcome by only allowing a maximum number of iterations.

5.2. Further work

The following areas may expose interesting or better results upon further examination:

Implementation:

- Since the implemented software is only a proof-of-concept, a more sophisticated implementation could be thought of. To the author's knowledge, there does not exist a software or software framework, which directly allows to easily classify different kinds of sequential data using different kinds of kernels, advanced evaluation techniques and advanced parameter determination techniques.

  SHOGUN does offer a lot of the described techniques but lacks for example a flexible cross-validation framework. As SHOGUN is free software and actively developed, this could be implemented. Still, SHOGUN is rather a software-framework than being an end-product. A software that is less technical to use would be nice for educational and research purposes.

- Many of the applied techniques are computational expensive. Off-line classification of biological data ist not time critical, however, many of the computations are independent from each other and could be parallelized. For example the grid-search for determining best parameters or repetitions of single evaluations would extremely benefit from parallelization. In addition, in the last years, computer hardware more and more offers the possibility of benefiting from parallelization (e.g. multiple-core processors, GPUs). This would allow to process more complex data, or to increase the number of used parameters.

Parameter search:

- The described approach for maximizing an unknown function, which is continuous and has one maximum is yet simple, but already performs better than the classical grid-search. This technique could be extended, to work more complex situations, for example in multiple dimensions.

- The algorithm was modified to be robust against small local maxima. The guaranteed termination vanishes this way. A further modification could be added in such way that the search interval converges to the smallest possible search interval around a maximum with a certain uncertainty bound (see figure 3.6). This way, the total correctness could be restored.

- The problem of determining best parameters for a certain method is not limited to SVM and kernels, but forms an own field of research. The examination of methods for determining parameters with alternate approaches, for example using evolutionary algorithms, and application of these to the current context, could be considered.

- The function, which describes dependencies between regularization parameter of a C-SV classifier and its performance was *empirically* shown to be nearly continuous. If it could *theoretically* be shown to be continuous, this would increase the value of the introduced bisection based method for parameter search even more, since then, it would be *guaranteed* to work instead of being a heuristic.

More data: The classification of sequence based data is, as already mentioned, also used in other fields than biology. The performance of the described approach could be tested on other problems that involve data based on sequences of characters, for example in context of computer-vision.

Appendices

A. Implementation

The following section describes implemented software. Despite from stated third-party software, all components were written by the author of this work.

Since the intend of this work is to test the described approach of amino-acid classification, *not* to implement a software, the latter is more a proof of concept than a stable system. The system is built with a layered architecture, therefore single components may be removed, replaced or used in another context. Interfaces are documented in the source code. New datasets or kernels may easily be integrated, which allows to use the software in other situations.

Since the goal was to *evaluate* different kernels, methods and datasets, a large amount of data is produced on every run. This data may be used to build optimal classifiers for certain situations. Still, focus is mainly on gaining insight into underlying problems.

A.1. Used Soft- and Hardware

There exit a vast number of tools that are basically suitable for the desired experiments. The following were used:

R The main parts of the software are written in an interpreted, script based programming language for statistical data analysis available under the GNU-GPL[1], which is called R[2]. It allows comfortable data processing and has powerful plotting functions. Since R is a interpreted language it suffers from performance problems when using loops. Therefore, time critical tasks are implemented in C++ and are embedded into R, which is easily possible via the package Rcpp[3]. ROCR (Sing et al., 2005) is a package, which is used in context of ROC-curves. In Addition, the following packages were used.

Kernlab (Karatzoglou et al., 2004) is a package for R, which provides machine learning tools. However, it is only used for computation of kernel matrices, since there was found a bug in the SVM implementation, which caused infinite loops in some situations.

[1] http://www.gnu.org/licenses/gpl.html
[2] http://www.r-project.org
[3] http://cran.r-project.org/web/packages/Rcpp

Shogun (Sonnenburg et al., 2006) is also a machine learning package for R, which has the focus on large-scale SVM learning. It has bindings for many languages, among them R. Among others, it implements the widespread LIBSVM (Chang and Lin, 2001) SVM implementation and, more importantly, allows usage of strings as input data and to import custom kernel matrices.

Thanks to the Essen Bioinformatics[4] group, the computations were performed on a node of their cluster *Knecht*, which consists of a debian[5] system that runs on the top of a 2.6 GHz dual-core AMD Opteron processor and 16 GB of memory.

A.2. Implemented Software

The software's structure is *layered*. Below is a hierarchically ordered list of its main components, along with a brief description of used input, generated output and other used components. Note that *only* parameters, which are actually used by components are listed. In the actual implementation, the main script is called with *all* parameters, which are used later on. These parameters are forwarded to all called components.

Search for kernels parameter

Used Input	**Output**
Kernel	Best kernel parameter
Kernel parameter search range	Search curves & stats
Performance Measure to use for comparison	Performance measures of best classifier

This is the main script of the software. Here, the grid search, which is described in section 3.5.1 starts iterating linearly over desired kernel parameters as described in section 3.5.2. For each kernel parameter of the given kernel, the corresponding gram matrix of the provided dataset is loaded (details are given later). Then, the best regularization parameter is searched for, by using the next component in the hierarchy. The best kernel parameter is selected by maximizing the provided desired performance measure for comparison. Parameters of the best found classifier, information concerning search and quality of the selected classifier are output.

The following two components are on the same hierarchical level, because the bisection based search is a replacement for the grid-search for regularization parameters.

[4] http://www.uni-due.de/bioinformatik
[5] http://www.debian.org

Naive (grid-)search for regularization parameter

Used Input	Output
Coarse search range and step width	Best regularization parameter
Fine search range and step width	Search curves & stats
Performance Measure to use for comparison	

Performs a one-dimensional exponent based search for regularization parameter C of the SVM as described in section 3.5.2. The step width input parameter determines the difference between two used parameters in the search, for coarse and fine search. Every parameter is evaluated using the the next component in the hierarchy. The best regularization parameter is selected by maximizing the provided desired performance measure for comparison and it is returned along with information concerning the search for it.

Bisection based search for regularization parameter

Used Input	Output
Tolerance for termination	Best regularization parameter
Uncertainty bound ϵ	Search curves & stats
Performance Measure to use for comparison	

Same as above, but with the bisection based approach and its parameters, as described in section 3.6.

Classifier evaluation

Used Input	Output
Number of runs	Averaged performance measures

Calls the component which is next in the hierarchy the provided number of times. Based on the resulting predicted class labels, performance measures, which are described in section 3.3 are calculated, averaged and reported back.

Cross-validation

Used Input	Output
Class labels	Estimated class labels of all tested partitions
Number of partitions	

Cross-validation partitions data regarding their class labels as described in section 3.2. Then these partitions are used for training and testing, as described in the mentioned section using the next component in the hierarchy. The resulting predicted class labels are reported back for every partition.

SHOGUN back-end

Used Input	Output
Data for training and test	Estimated class labels of test data
Kernel matrix (includes kernel parameter)	
Regularization parameter	

Using gram matrix and regularization parameter, the SHOGUN back-end trains a SVM with the provided training data and class labels. Then, it performs class label prediction on test data and reports back the results. SHOGUN always uses its LIBSVM implementation here.

A.3. Tools

A.3.1. Kernel Matrix Generation

Normally, when using SVM implementations, the underlying kernel(-function) is evaluated for every pair of input data whenever the dot-product of the corresponding feature vectors is needed. Then, after the program has finished, these values are lost. This results in a runtime problem when large datasets and complex kernels are used, as in the context of this work. Consequently, the process of evaluating the kernel values is moved to a separate part of the software. The gram matrix (see equation 2.21) of the underlying kernel and the underlying dataset is evaluated and saved on hard-disk *once*. Then it is reloaded on demand. While calculating kernel matrices, the fact that the latter are symmetric is used.

Note that the calculation of a gram matrix presumes that a kernel parameter is already chosen. Since the optimal kernel parameter is to be searched, the gram matrices for pairs of kernels and datasets are calculated and saved for a reasonable range of kernel parameters. Then, choosing a kernel parameter corresponds in loading the corresponding kernel matrix instance.

The kernel matrices of the Spectrum kernel are calculated using kernlab. The matrix of the DS-kernel is calculated using software, which is provided in (Boisvert et al., 2008). Note that both kernel matrices are normalized as described in section 2.23.

A.3.2. Entropy Tools

A set of C++ functions generates curves of the sequence based entropy of datasets (see appendix E and section 3.7). To do this, a bunch of functions is implemented to accomplish the following tasks on a set of strings:

- Extract the set of all substrings of a given length,
- calculate the number of possible substrings of a given length,

- calculate the frequency of occurrence of a substring,
- on the base of the latter, calculate the sequence based entropy.

A.3.3. Single Classifier Evaluation

In addition to the introduced software, a script, which evaluates a single classifier is provided. It is possible to input a set of parameters (regularization parameter, kernel parameter, kernel type, cross-validation parameter, number of runs). The corresponding classifier is evaluated and the performance measures are outputted. The tool is used to obtain tight confidence intervals for classifiers found by the parameter search, as described in section 3.4.

The tool may be used to verify all results, which are given in this work.

B. Proofs

To show the partial correctness of algorithm 3.1, the following two standard theorems of mathematical calculus are needed. Details can for example be found in (Forster, 2008).

Theorem 1 (Intermediate value theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be a continuous function and let $c \in \mathbb{R}$ with $f(a) < c < f(b)$. Then there exists a $p \in [a, b]$ with $f(p) = c$.*

Theorem 2 (Extreme value theorem)**.** *Given a closed and bounded interval $[a, b] \subset \mathbb{R}$, every function $f : [a, b] \to \mathbb{R}$, which is continuous in $[a, b]$, attains a maximum and a minimum in $[a, b]$, each at least once. That is, there exist numbers $g, h \in [a, b]$ with $f(g) \leq f(x) \leq f(h)$ for any $x \in \mathbb{R}$.*

These two theorems can be used to proof the following proposition, which is necessary for algorithm 3.1 to be correct. The proof is inspired by the standard proof for *Rolle's Theorem* in (Forster, 2008).

Proposition 1. *Given a closed and bounded interval $[a, b] \subset \mathbb{R}$, a function $f : \mathbb{R} \to \mathbb{R}$, which is continuous in $[a, b]$ and has exactly one maximum in $]a, b[$, and $x, y, z \in \mathbb{R}$ with $a \leq x < y < z \leq b$ such that $f(x) < f(y) > f(z)$, then* $\operatorname{argmax} f(x) \in]x, z[$.

Proof. Since $f(x) < f(y) > f(z)$, f is *not* constant in $[a, b]$. Theorem 1 gives the existence of $z' \in \mathbb{R}$ with $y < z' < z$ and $f(z') = f(x)$. Because f is continuous in $[a, b]$ and therefore also continuous in $[x, z']$, theorem 2 gives the existence of a maximum in $[x, z']$, which is the only maximum of f. Since $y \in]x, z'[$ and $f(x) < f(y) > f(z')$, f has *no* maximum at the interval borders x and z'. That is, $\operatorname{argmax} f(x) \in]x, z'[$ and because $z' < z$ it holds $\operatorname{argmax} f(x) \in]x, z[$.

□

Now, the partial correctness of the algorithm can be shown. This is done by showing the correctness of the following proposition:

Proposition 2. *Let $f : \mathbb{R} \to \mathbb{R}$ be a function, which is continuous in $[a, b]$ and has exactly one maximum in $]a, b[$. Given the input tolerance, a, b and f, any output o of algorithm 3.1 fulfills $|\operatorname{argmax} f(x) - o| < tolerance$.*

Proof. First is shown that $\operatorname{argmax} f(x) \in]a, b[$ is an invariant of the algorithm. Then it is shown that the algorithm only terminates when the assumptions are met.

Using the assumptions, $\operatorname{argmax} f(x) \in]a, b[$ holds before the algorithm starts. The only place where $]a, b[$ ist changed is in the inner *if*-statement (lines 14 and 15). Using

proposition 1 and the assumptions, the invariant holds if $f(x) < f(y) > f(z)$ and $a \leq x < y < z \leq b$. The first inequality holds because the inner *if*-statement (line 13). $x < y < z$ holds because $a < b \Rightarrow step > 0$ and for any $i \in \mathbb{N}$, it holds

$$a + (i) \cdot step < a + (i+1) \cdot step < a + (i+2) \cdot step \Rightarrow x < y < z.$$

The variable in the *for*-loop runs in $0, ..., n-2$, where $n \geq 3$ throughout the algorithm. For $i = 0$ it holds

$$x = a + (i) \cdot step = a,$$

for $i = n - 2$, it holds

$$z = a + (i+2) \cdot step = a + n \cdot \frac{b-a}{n} = b.$$

For any $i > 0$, it holds

$$x = a + (i) \cdot step > a,$$

for any $i < n - 2$, it holds

$$z = a + (i+2) \cdot step < b.$$

Together this gives the complete inequality $a \leq x < y < z \leq b$. So after the *if*-statement $\operatorname{argmax} f(x) \in]a, b[$ still holds, which qualifies it as invariant.

Because of the *if*-statement containing the *return*-statement (line 3), the algorithm's output fulfills proposition 2, since it is only entered if

$$b - a < tolerance \Rightarrow \frac{b-a}{2} < tolerance$$

Together with the invariant follows $|a + \frac{b-a}{2} - \operatorname{argmax} f(x)| < tolerance$. □

Now, to show the total correctness of the algorithm, the following proposition is shown to be correct:

Proposition 3. *Given the same requirements as in proposition 2, algorithm 3.1 terminates after a finite number of iterations.*

To proof this, it is shown that the inner *if*-statement (line 13) is entered after a finite number of iterations of the main loop (line 2 to line 24).

Proof. Let e be the position of the only maximum in $]a, b[$. Then, there exists a neighborhood α, β with $a < \alpha < e < \beta < b$, such that for any x, x' with $\alpha < x < x' \leq e$, it holds $f(x) < f(x')$ and any y, y' with $e \leq y' < y < \beta$, it holds $f(y) < f(y')$.

In the algorithm, n increases by one every iteration if the inner *if*-statement (line 13) is not entered (trivial). Consequently n may grow such large that

$$2 \cdot step < \min(e - \alpha, \beta - e) \Leftrightarrow n > 2 \cdot \frac{b-a}{\min(e - \alpha, \beta - e)},$$

and

$$3 \cdot step > \min(e-\alpha, \beta-e) \Leftrightarrow n < 3 \cdot \frac{b-a}{\min(e-\alpha, \beta-e)},$$

which is fulfilled by an n with

$$2 \cdot \lceil \frac{b-a}{\min(e-\alpha, \beta-e)} \rceil \leq n \leq 3 \cdot \lfloor \frac{b-a}{\min(e-\alpha, \beta-e)} \rfloor.$$

Because $b-a > \min(e-\alpha, \beta-e)$, the distance between these borders is at least one, so there may always be found such an n.

When the counter variable i in the *for*-loop (line 8) that runs over $i = 0, ..., n-2$ becomes

$$i = \lfloor \frac{e-a}{step} \rfloor - 2,$$

which always happens at some point since

$$\frac{e-a}{step} - 2 = n \cdot \frac{e-a}{b-a} - 2 \leq n - 2$$

and

$$\frac{e-a}{step} - 2 = n \cdot \frac{e-a}{b-a} - 2 \geq 2 \cdot \frac{b-a}{\min(e-\alpha, \beta-e)} \cdot \frac{e-a}{b-a} - 2 \geq 0,$$

it holds

$$\alpha < a + (i) \cdot step < a + (i+1) \cdot step < e,$$

and

$$e < a + (i+2) \cdot step < \beta.$$

Using the fact of the beginning of the proof, when i becomes the above value, it holds

$$f(a + (i) \cdot step) < f(a + (i+1) \cdot step) > f(a + (i+2) \cdot step).$$

So, the inner *if*-statement (line 13) is entered and the search interval becomes smaller by the factor $\frac{b-a}{2 \cdot step}$. Consequently, the search interval becomes smaller by some factor after a finite number of iterations. □

C. Amino-acids

Amino-acids	**3-letter-code**	**1-letter-code**
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamic acid	Glu	E
Glutamine	Gln	Q
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

Figure C.1.: All 20 amino-acids and their letter representations.

D. Sequence Length Histograms

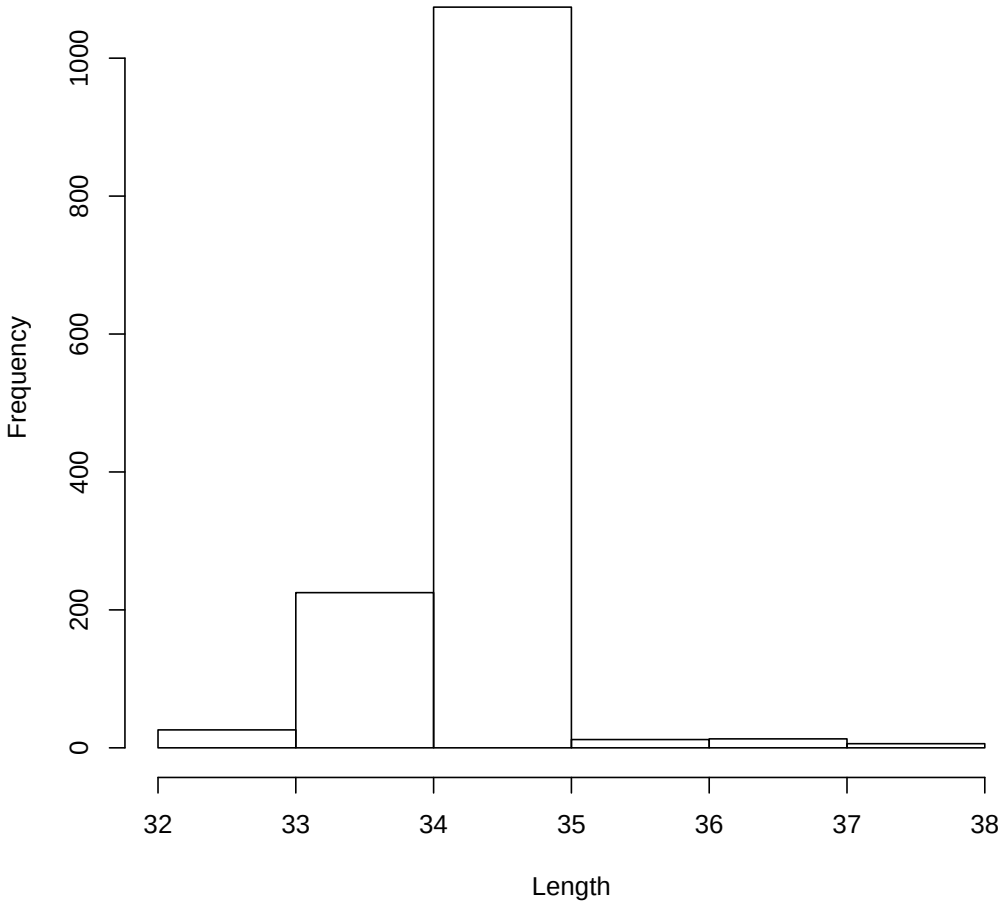


Figure D.1.: Sequence length distribution of dataset A. Individual sequence lengths quite homogeneous. Overall sequence length is short.

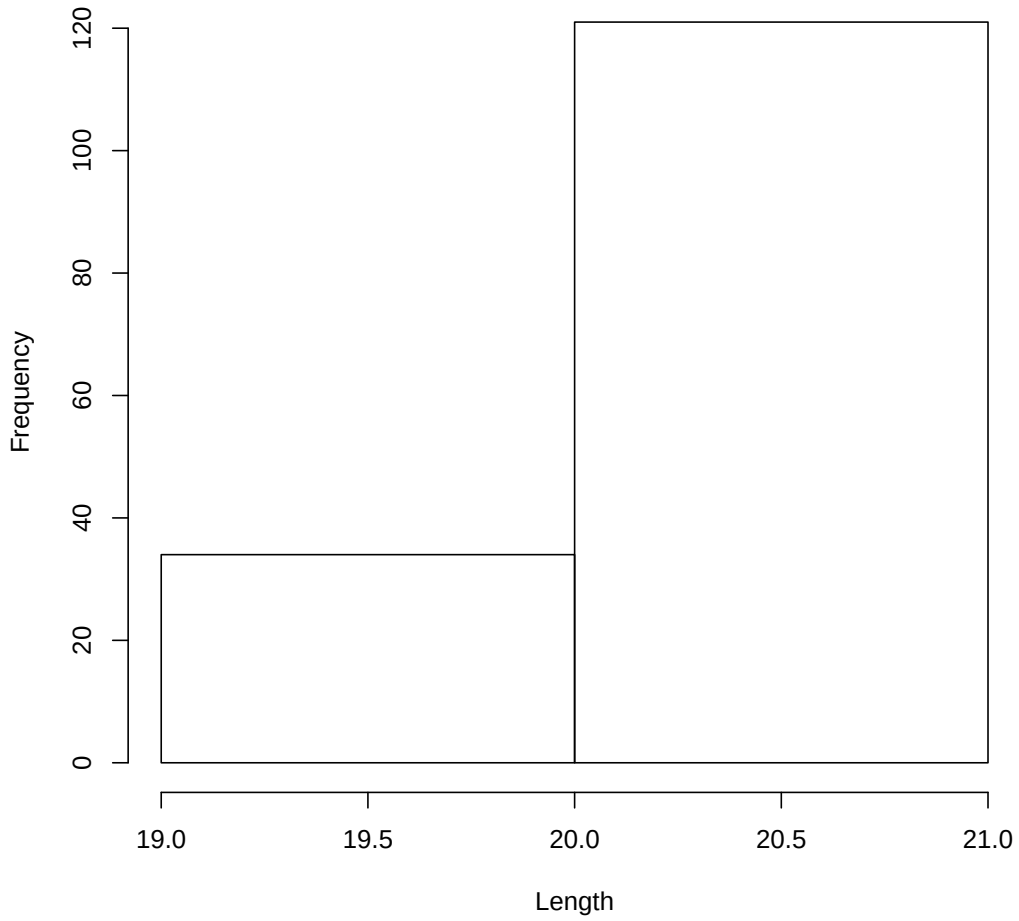


Figure D.2.: Sequence length distribution of dataset D. Individual sequence lengths are extremely homogeneous. Overall sequence length is very short.

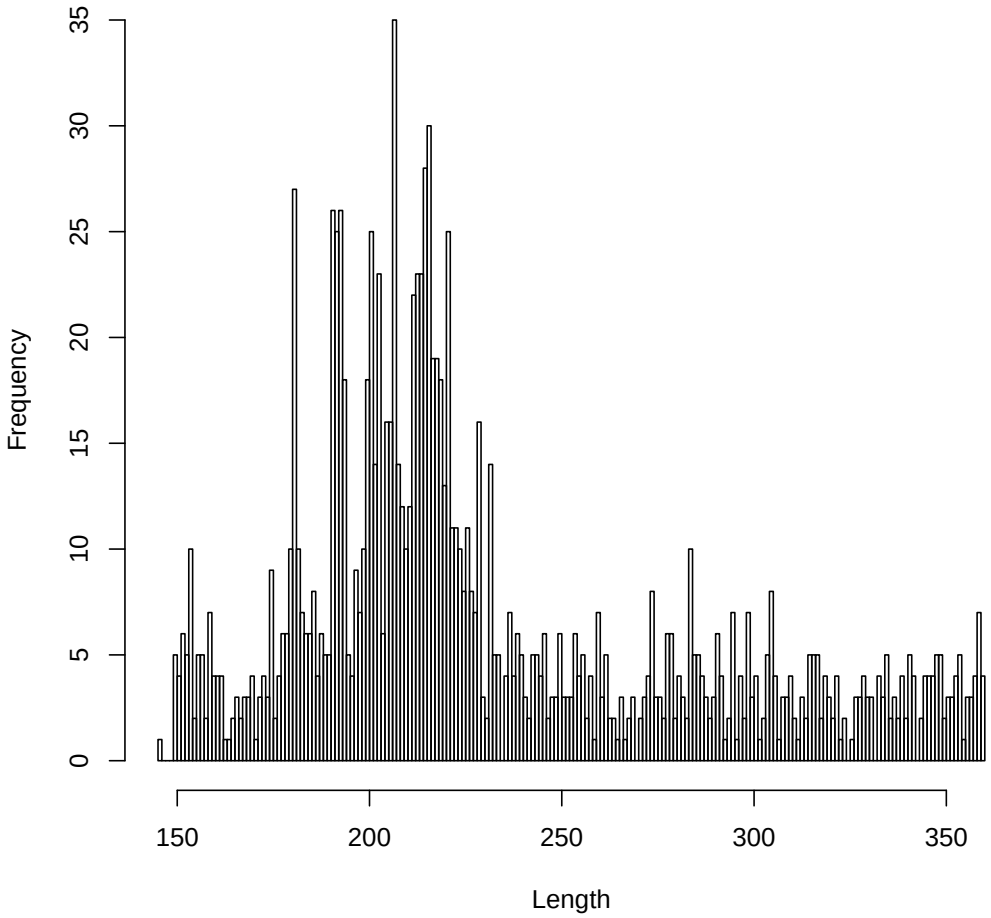


Figure D.3.: Sequence length distribution of dataset C. Individual sequence lengths are extremely divergent. There are sequences of large to extremely large lengths.

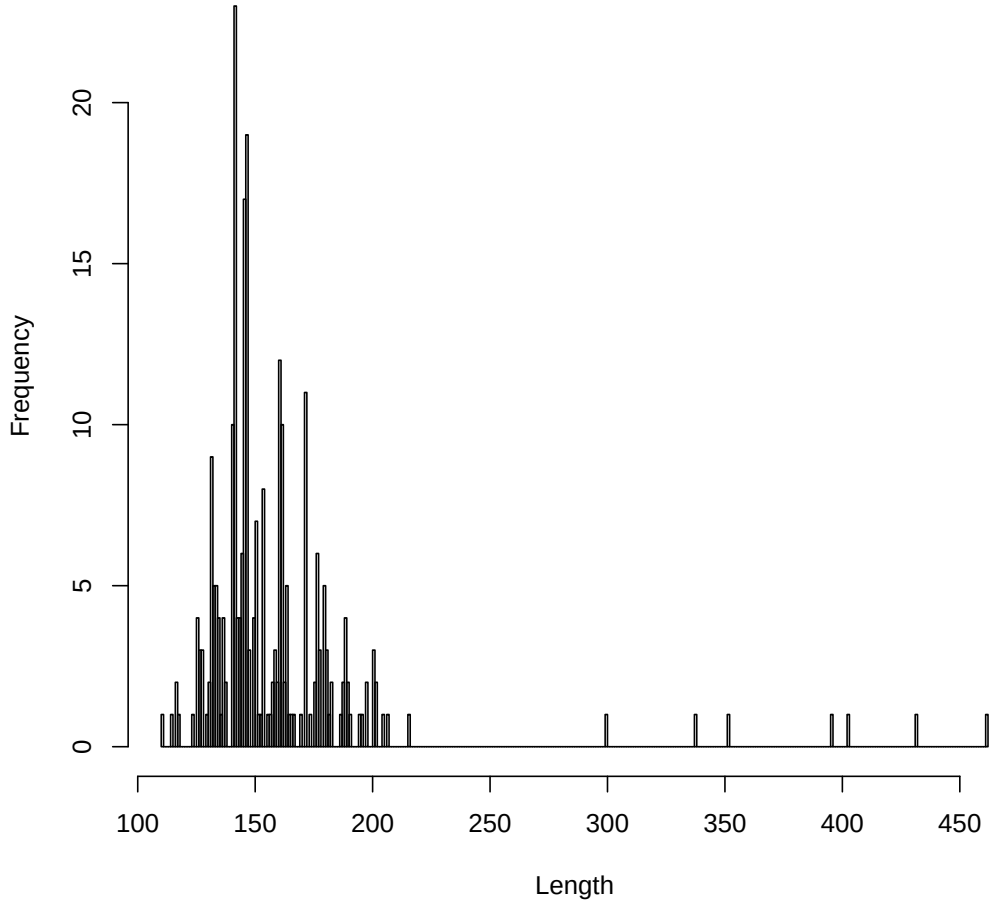


Figure D.4.: Sequence length distribution of dataset D. Individual sequence lengths are strongly divergent. There are sequences of large to extremely large lengths.

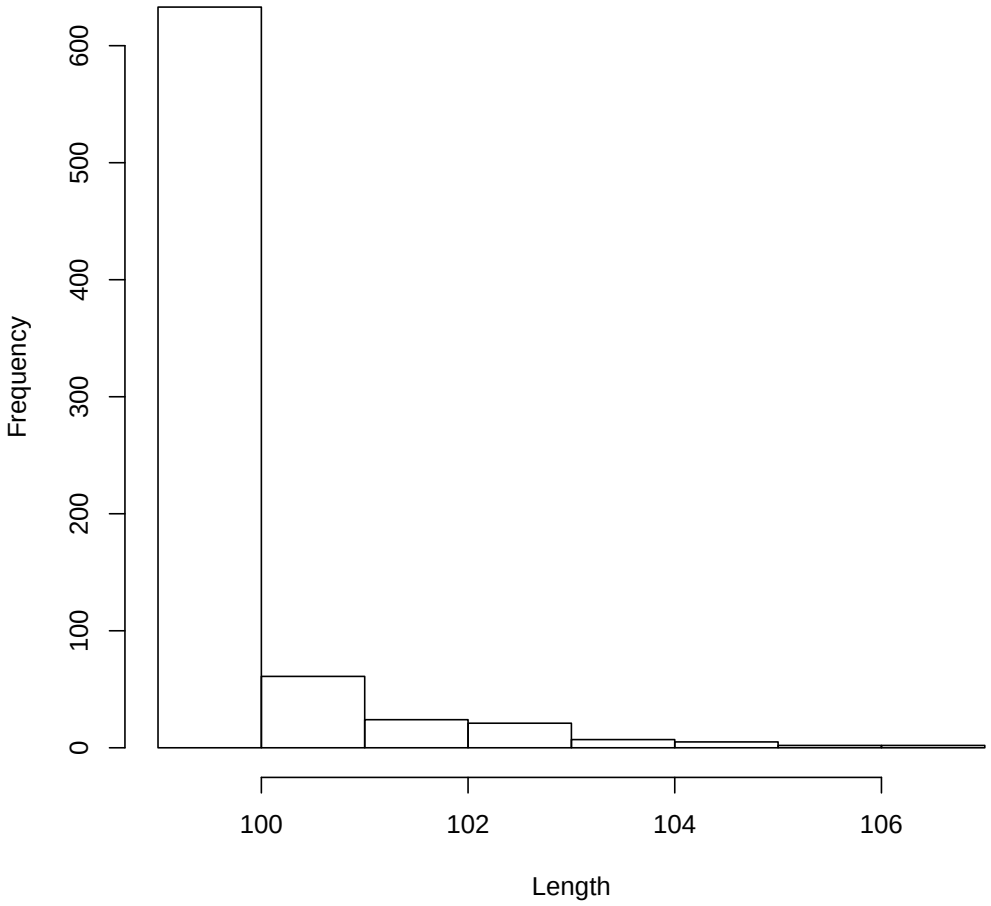


Figure D.5.: Sequence length distribution of dataset E. Individual sequence lengths do not differ a lot. Overall length is large.

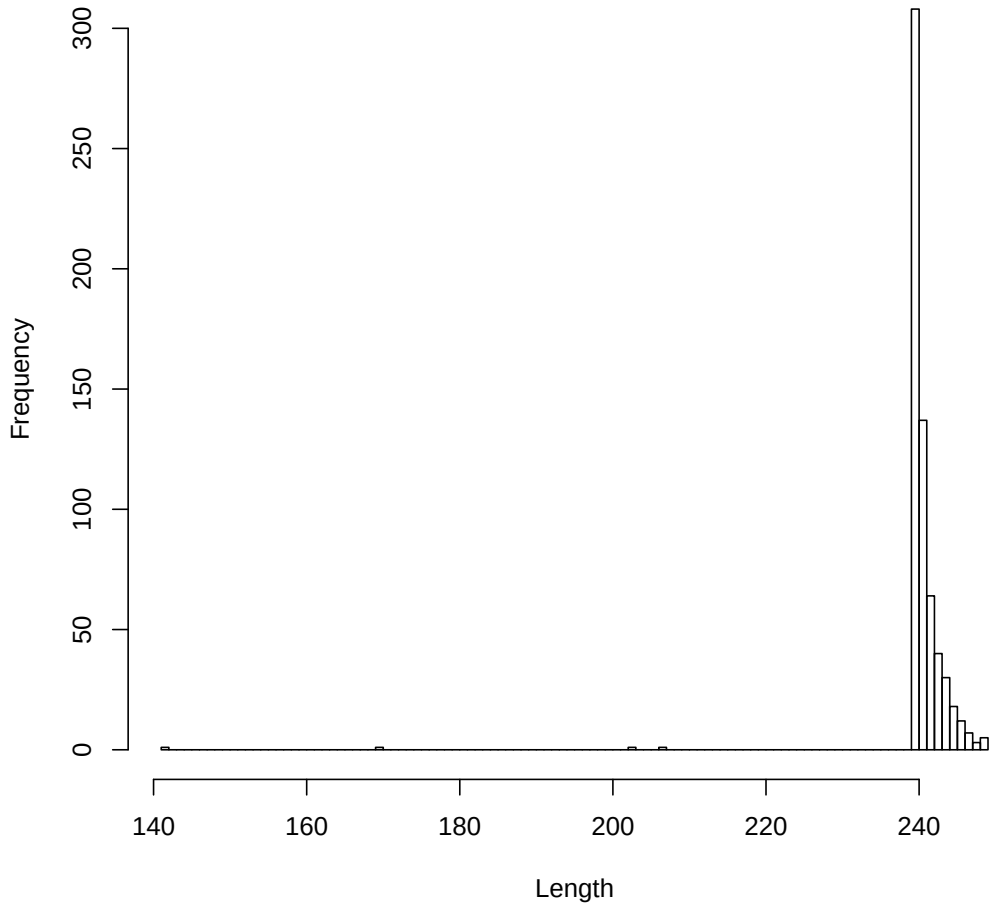


Figure D.6.: Sequence length distribution of dataset F. Apart from few outliners, individual sequence lengths do not differ a lot. Overall length is very large.

E. Curves of Sequence Based Entropy

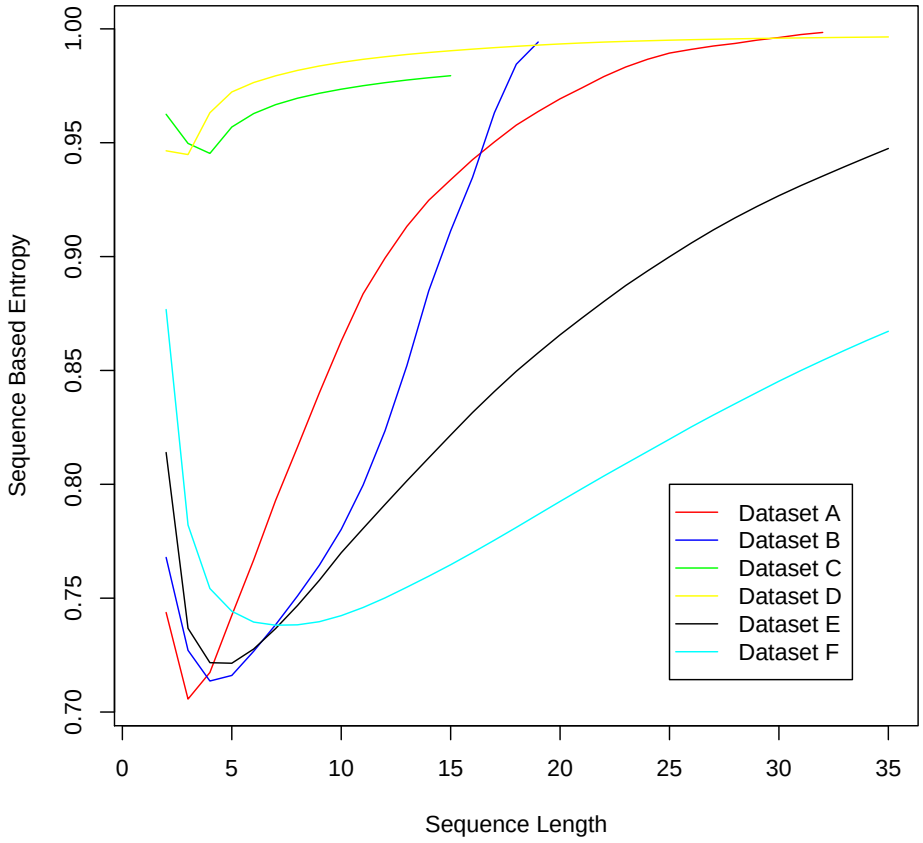


Figure E.1.: Entropy Curves of all datasets as described in 3.7. All curves share the property of having a minimum at a lower sequence length and asymptotic convergency to one as the sequence length becomes larger. Dataset C and dataset D are more divergent than the others and though have a higher minimum.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelor-Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle wörtlich und sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommenen Stellen sind als solche kenntlich gemacht.

Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und noch nicht veröffentlicht worden ist.

Ort, Datum	Unterschrift

Bibliography

Ben-David, S. and Simon, H. (2001). Efficient learning of linear perceptrons. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, page 189. The MIT Press.

Bishop, C. et al. (2006). *Pattern recognition and machine learning*. Springer New York:.

Boisvert, S., Marchand, M., Laviolette, F., and Corbeil, J. (2008). HIV-1 coreceptor usage prediction without multiple alignments: an application of string kernels. *Retrovirology*, 5:110.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge Univ Pr.

Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision, ECCV*, volume 1, page 22. Citeseer.

Fawcett, T. (2003). ROC graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories.

Forman, G. and Scholz, M. (2009). Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement. Technical report, HP Laboratories.

Forster, O. (2008). *Analysis 1: Differential-und Integralrechnung einer veränderlichen*. Springer.

Graepel, R. (2001). A PAC-Bayesian Margin Bound for Linear Classifiers: Why SVMs work. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, page 224. The MIT Press.

Heider, D., Appelmann, J., Bayro, T., Dreckmann, W., Held, A., Winkler, J., Barnekow, A., and Borschbach, M. (2009). A computational approach for the identification of small GTPases based on preprocessed amino Acid sequences. *Technology in cancer research & treatment*, 8(5):333.

Heider, D., Verheyen, J., and Hoffmann, D. (2010). Predicting Bevirimat resistance of HIV-1 from genotype. *BMC bioinformatics*, 11(1):37.

Hsu, C., Chang, C., Lin, C., et al. (2003). A practical guide to support vector classification.

Jaakkola, T., Diekhans, M., and Haussler, D. (1999). Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20. Software available at `http://cran.r-project.org/web/packages/kernlab`.

Leslie, C., Eskin, E., and Noble, W. (2002). The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 564–575.

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions Royal Society London*.

Merkl, R. and Waack, S. (2009). *Bioinformatik Interaktiv: Algorithmen und Praxis*. Wiley-VCH.

Platt, J. (1999). Advances in large margin classifiers, chapter Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods.

Rhee, S., Taylor, J., Wadhera, G., Ben-Hur, A., Brutlag, D., and Shafer, R. (2006). Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences*, 103(46):17355.

Rojas, R. and Feldman, J. (1996). *Neural networks: a systematic introduction*. Springer.

Saigo, H., Vert, J., Ueda, N., and Akutsu, T. (2004). Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682.

Schölkopf, B. and Smola, A. (2001). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge Univ Pr.

Sing, T., Sander, O., Beerenwinkel, N., and Lengauer, T. (2005). ROCR: visualizing classifier performance in R. *Bioinformatics*.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Sonnenburg, S., Rätsch, G., Schäfer, C., and Schälkopf, B. (2006). Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1565.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley-Interscience, New York.

Vapnik, V. (2000). *The nature of statistical learning theory*. Springer Verlag.

Bibliography

Ben-David, S. and Simon, H. (2001). Efficient learning of linear perceptrons. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, page 189. The MIT Press.

Bishop, C. et al. (2006). *Pattern recognition and machine learning*. Springer New York:.

Boisvert, S., Marchand, M., Laviolette, F., and Corbeil, J. (2008). HIV-1 coreceptor usage prediction without multiple alignments: an application of string kernels. *Retrovirology*, 5:110.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge Univ Pr.

Csurka, G., Dance, C., Fan, L., Willamowski, J., and Bray, C. (2004). Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision, ECCV*, volume 1, page 22. Citeseer.

Fawcett, T. (2003). ROC graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories.

Forman, G. and Scholz, M. (2009). Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement. Technical report, HP Laboratories.

Forster, O. (2008). *Analysis 1: Differential-und Integralrechnung einer veränderlichen*. Springer.

Graepel, R. (2001). A PAC-Bayesian Margin Bound for Linear Classifiers: Why SVMs work. In *Advances in neural information processing systems 13: proceedings of the 2000 conference*, page 224. The MIT Press.

Heider, D., Appelmann, J., Bayro, T., Dreckmann, W., Held, A., Winkler, J., Barnekow, A., and Borschbach, M. (2009). A computational approach for the identification of small GTPases based on preprocessed amino Acid sequences. *Technology in cancer research & treatment*, 8(5):333.

Heider, D., Verheyen, J., and Hoffmann, D. (2010). Predicting Bevirimat resistance of HIV-1 from genotype. *BMC bioinformatics*, 11(1):37.

Hsu, C., Chang, C., Lin, C., et al. (2003). A practical guide to support vector classification.

Jaakkola, T., Diekhans, M., and Haussler, D. (1999). Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20. Software available at `http://cran.r-project.org/web/packages/kernlab`.

Leslie, C., Eskin, E., and Noble, W. (2002). The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 564–575.

Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions Royal Society London*.

Merkl, R. and Waack, S. (2009). *Bioinformatik Interaktiv: Algorithmen und Praxis*. Wiley-VCH.

Platt, J. (1999). Advances in large margin classifiers, chapter Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods.

Rhee, S., Taylor, J., Wadhera, G., Ben-Hur, A., Brutlag, D., and Shafer, R. (2006). Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences*, 103(46):17355.

Rojas, R. and Feldman, J. (1996). *Neural networks: a systematic introduction*. Springer.

Saigo, H., Vert, J., Ueda, N., and Akutsu, T. (2004). Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682.

Schölkopf, B. and Smola, A. (2001). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge Univ Pr.

Sing, T., Sander, O., Beerenwinkel, N., and Lengauer, T. (2005). ROCR: visualizing classifier performance in R. *Bioinformatics*.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

Sonnenburg, S., Rätsch, G., Schäfer, C., and Schälkopf, B. (2006). Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1565.

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley-Interscience, New York.

Vapnik, V. (2000). *The nature of statistical learning theory*. Springer Verlag.