Domain Adaptation in Sequence Analysis

Diplomarbeit im Studiengang Bioinformatik

vorgelegt von

Christian Widmer

Erstgutachter: Prof. Dr. Daniel Huson[1,2]
Zweitgutachter: Dr. Gunnar Rätsch[3]

Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften
Universität Tübingen

Tübingen, im September 2008

[1]Wilhelm-Schickard-Institut für Informatik
[2]Zentrum für Bioinformatik Tübingen
[3]Friedrich-Miescher-Labor der Max-Planck-Gesellschaft, Tübingen

Domain Adaptation in Sequence Analysis

Christian Widmer

Abstract

The machine learning approach to problems from computational biology is to learn models from labeled training examples. A common problem in supervised learning is the lack of sufficient training data. Often, however, there exists a large body of data on a different, but related problem. In order to boost performance on our problem of interest, we would like to utilize information from this related domain. The field of *domain adaptation* formalizes this scenario and provides intelligent algorithms to transfer information between domains. The aim of this work is to compare, extend and develop domain adaptation algorithms suited for problems in computational biology.

A central part of this work is the evaluation of domain adaptation methods in large scale computational experiments. To our knowledge, this work constitutes the first thorough experimental comparison of available domain adaptation algorithms in a well controlled experimental framework. Furthermore, we are one of the first to apply such methods to problems in computational biology, which appears to provide a wealth of potential applications.

Representative for the class of sequence-based classification problems we chose the problem of splice site prediction - an important subproblem in computational gene finding. In splice site prediction, challenges arise from an extremely high dimensional feature space, unbalanced data sets and the fact that there is very little labeled data available for newly sequenced organisms. To improve performance on the organism of interest, we turn to other organisms as additional sources of information. Experiments were conducted for a set of target organisms of varying phylogenetic distance to the source organism and for multiple data set sizes. Comparing the methods over a range of conditions allows us to draw conclusions under what circumstances which methods perform well.

We observed that the methods that learn solutions on all data sources simultaneously perform best. Furthermore, methods based on simple convex combinations provide the most cost-effective solution to the domain adaptation problem. The largest improvements over baseline methods can be observed if the organisms are only distantly related. Using the example of splice site prediction, we can show that domain adaptation algorithms are well suited to considerably boost predictive performance on sequence based classification problems.

Acknowledgment

First, I want to thank Gunnar Rätsch and Daniel Huson for giving me the chance to conduct this research project in an excellent scientific environment. I wish to address special thanks to Gabriele Schweikert, who supervised my thesis, provided code for some domain adaptation methods, performed parts of the large scale experiments and provided me with many helpful comments on my manuscript and guidance throughout this project.

Furthermore, I am very grateful for the helpful discussions and good advice from Cheng Soon Ong, Petra Phillips and Alexander Zien. I would also like to express appreciation to Bernhard Schölkopf for showing interest in this work and for contributing ideas and encouragement. Further, I want to acknowledge Regina Bohnert, Sebastian Schultheiss, Timo Sachsenberg and Mee Joo Bärenstrauch for reviewing my manuscript. I would like to address thanks to my colleagues at the FML for a friendly and supportive working environment.

Lastly, I would like to thank my family and friends for supporting me during this challenging time.

Contents

Abstract iii

Acknowledgment v

1 Introduction 1
- 1.1 Domain Adaptation in Computational Biology . . . 1
- 1.2 Application to Gene-finding . . . 1
- 1.3 Aims of the Thesis . . . 3
- 1.4 Thesis Structure . . . 3

2 Background 5
- 2.1 Biology . . . 5
  - 2.1.1 Gene structure . . . 5
  - 2.1.2 Splicing . . . 7
- 2.2 Kernel Methods . . . 8
  - 2.2.1 Similarity Measure . . . 8
  - 2.2.2 Kernel Trick . . . 8
  - 2.2.3 Encapsulation of data and non-linearity . . . 10
  - 2.2.4 String Kernels . . . 12
- 2.3 Support Vector Machines . . . 12
  - 2.3.1 Motivation . . . 12
  - 2.3.2 Optimization Problem . . . 13
- 2.4 Evaluation . . . 16
  - 2.4.1 Cross-validation . . . 16
  - 2.4.2 Performance Measures . . . 17

3 Domain Adaptation 19
- 3.1 Overview . . . 19
- 3.2 Distributional Point of View . . . 19
- 3.3 Methods and Extensions . . . 20
  - 3.3.1 Baseline Methods . . . 20
  - 3.3.2 Instance Weighting . . . 21
  - 3.3.3 Change of Representation . . . 23
  - 3.3.4 Methods using Priors . . . 24
  - 3.3.5 Multi-task Learning . . . 31
  - 3.3.6 Ensemble Methods . . . 32
  - 3.3.7 Combination of Several Sources . . . 33
- 3.4 Method Summary and Contribution . . . 34

3.4.1 Contribution 36

4 Implementation **39**
4.1 Implementation of Domain Adaptation Methods 39
4.2 Evaluation Pipleline 41
4.3 A Grid Computing Tool: Pythongrid 44
4.4 Serialization 47

5 Application to Splice Site Prediction **51**
5.1 The Problem of Splice Site Prediction 51
5.2 Domain Adaptation Setting 53
5.3 Comparison of Domains 54
5.4 Experimental Setup 58
5.5 Experimental Results 58
5.5.1 Multiple Sources 61

6 Discussion **65**
6.1 Outlook 66

7 Conclusion **69**

References **71**

Appendix **75**

1 Introduction

1.1 Domain Adaptation in Computational Biology

Learning models from data lies at the very heart of computational biology. Advances in biotechnology have led to considerable amounts of diverse biological data which reside in databases all over the world. Instead of attempting to decipher data manually to understand the underlying biological mechanisms, machine learning enables us to learn models from data in an automated and mathematically well-defined manner. In supervised learning, this is done by estimating parameters of a model from labeled training examples.

The more complex the model is, the more data is required to obtain a good predictor. In supervised learning, the limiting factor is often lack of sufficient training data. In fact, in machine learning, we frequently face situations where only little data is available for the problem that we would like to investigate, while there exists a large body of data on a different, but related problem. These data could serve as valuable sources of information, which we would like to exploit for the purpose of improving the predictive performance on our domain of interest. This gives rise to the need for intelligent *domain adaptation* algorithms, which help to transfer knowledge between domains in a sensible way.

The field of computational biology provides plenty of problems for which domain adaptation algorithms are relevant. In principle, every family of related problems for which more data could be helpful constitutes a possible application. Consider metabolic networks for a set of organisms, gene expression data from multiple tissues or promoter detection on several families of genes. Only recently, the problem of domain adaptation has started to attract attention in the machine learning community (EP04; DIM06; BMP06; JZ07; BdBCP07). Yet, very little research has investigated the application of domain adaptation algorithms to the field of computational biology (KKSA05). The focus of this work is to analyze domain adaptation algorithms for biological sequence analysis. This is done with the problem of gene finding in mind, which heavily relies on sequence signal prediction.

1.2 Application to Gene-finding

In the face of considerable advances in sequencing technologies (Met05; HI07), the genomes of an increasing number of organisms have been sequenced. Whereas the number of available genomes was limited to representants of the main branches of the tree of life (see Figure 1.1) in the past, the availability of genomes has improved dramatically over the last years and will continue to become even more fine grained in the future.

To fully take advantage of the genomic sequence, annotations are created by assigning



Figure 1.1: Tree of life. This figure shows an artistic illustration of the tree of life. One can see the early split between procaryotes and eukaryotes. Furthermore, the eukaryotic branch is illustrated with examples for several kingdoms. [http://www.tolweb.org]

labels (e.g. intron, exon) to sequence segments. These annotations are of great importance, as they form the basis for subsequent biological experiments. Due to the sheer amount of data from various sources, such as EST (expressed sequence tag) or cDNA (complementary DNA) alignments, protein alignments, conservation, gene expression data or epigenetical data, the manual curation of sequence annotations renders extremely labour intensive. All this motivates the demand for reliable gene finding algorithms to create high-quality sequence annotations.

The problem of domain adaptation arises when the genomes of organisms are newly sequenced that do not have sufficient EST coverage. In fact, this is a common problem as the sequencing of genomes proceeds at a greater pace than the creation of EST libraries. Thus, we have to work with extremely incomplete annotations, which in turn results in the lack of labeled training data. This motivates the idea of considering related organisms as additional sources of information.

One way of realizing an information transfer between organisms is the use of conservation of the DNA sequence. Here, one aligns the genomes of two or more organisms and considers the matching regions. The assumption is that introns are significantly less conserved between species than exons, because there is lower evolutionary pressure towards the conservation of non-functional sequence. A potential problem is that we will only be able to benefit from whole genome alignments if the respective organisms are at an intermediate phylogenetic distance (e.g. human and mouse). If the organisms are too close, introns as well as exons might be conserved. On the contrary, if they are too far apart, one will simply obtain bad alignments, where only a relatively small percentage of the genome is aligned. Most useful are alignments that preserve the correspondence between coding exons, while showing divergence in intronic regions. Another way of transferring information between organisms is to align protein sequences from related organisms to the genomes of the newly sequenced organisms.

Lastly, one could simply train a gene finder on a related, well-annotated organism and use it to predict on a newly sequenced organisms. This has the disadvantage that we discard the potentially little but valuable information we have for the organism of interest.

In this work, we investigated an alternative approach to the problem of information transfer between organisms. Instead of relying on alignments, we employ domain adaptation algorithms from the field of machine learning. The domain adaptation algorithms presented in this work are not meant to replace alignment-based methods, but rather constitute a complement that approaches the problem of information transfer from a different direction.

Ultimately, we would like to use domain adaptation algorithms to enhance the gene finder *mGene*, which is a SVM (support vector machine) based machine learning approach to the gene finding problem and is actively developed by the *Machine Learning in Biology* group at the FML[1]. *mGene* employs a two-layered design. On the first layer, sequence signals such as splice sites or transcription start sites are recognized using a binary classifier. All outputs from the first layer are subsequently combined into a state model to predict a complete gene annotation. Because of this two-layered structure, it is possible to treat the layers separately with respect to domain adaptation. The main focus of this work lies on the improvement of sequence signal recognition on the first layer. Therefore, special attention was paid to domain adaptation methods that readily integrate into the framework for sequence signal prediction (SVMs using string kernels).

1.3 Aims of the Thesis

Generally speaking, we attempt to investigate domain adaptation algorithms for genomic sequence signal prediction in the context of gene finding.

The main contributions of this thesis are:

- A survey of domain adaptation algorithms
- Efficient implementations of domain adaptation algorithms based on new ideas
- Empirical comparison of the above methods on a problem relevant to gene finding

1.4 Thesis Structure

Chapter 2 will provide an overview of the biological background and explain the machine learning techniques that form the building blocks of the employed domain adaptation algorithms.

Subsequently in Chapter 3, we turn our attention to the field of domain adaptation. Here the idea of domain adaptation will be discussed in depth, including a detailed description of the methods used in this thesis. Special attention will be given to methods that were developed or modified in the course of this work.

[1] Friedrich Miescher Laboratory of the Max Planck Society, Tübingen, Germany, where this work was created.

In Chapter 4, we will briefly go over implementational aspects, explain some design choices and outline the development of missing computational infrastructure for performing large scale *in silico* experiments.

After having introduced the relevant methods and their practical realization, Chapter 5 will discuss the application to splice site prediction. The key difficulties of the splice site prediction problem are outlined, then the experimental setup is explained and lastly results are presented.

In Chapter 6, we discuss our findings and give an outlook on future research.

The thesis concludes with Chapter 7, where the central aspects of the thesis are summarized.

2 Background

2.1 Biology

2.1.1 Gene structure

As this work is motivated by the requirements of *ab-initio* gene finding, it makes sense to first define the structure of a gene. It is indeed non-trivial to come up with a general definition of a gene, when taking into account functional RNAs, pseudogenes and the like, as they have no overall conserved structure. For simplicity, I will therefore restrict myself to the definition of a eukaryotic protein-coding gene and its elements, based on the dogma of molecular biology. A simplified illustration of the dogma is displayed in Figure 2.1.

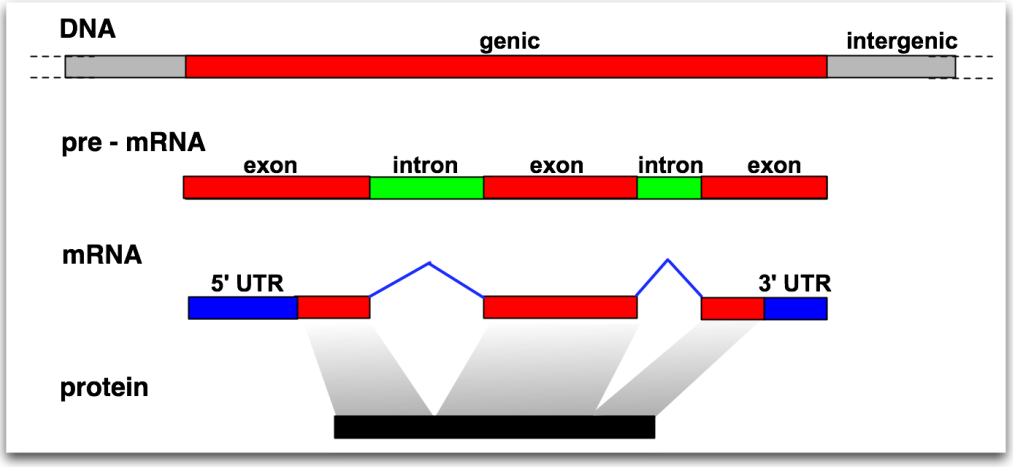


Figure 2.1: Schematic overview of the dogma of molecular biology. The dogma applies to protein-coding genes only. Segments of the DNA that get transcribed are termed genic, non-conding regions intergenic. DNA is transcribed to pre-mRNA, which carries introns and exons. Splicing removes introns from the pre-mRNA, leading to mature mRNA. Lastly, an mRNA is translated to protein. Segments at the beginning and end of the mRNA that are not translated are referred to as untranslated region (UTR). This Figure was created by Gabriele Schweikert.

To obtain a better understanding of the structure of a gene, the steps involved in gene expression will be briefly explained, as most sequence signals are relevant to one of these steps. In a simplified view, there are three steps on the way from a gene to its product, a protein: Transcription, splicing and translation.

Transcription In the first step, DNA is transcribed to a primary transcript, or pre-mRNA (pre-messenger RNA). Transcription gives rise to an exact[1] RNA copy of the (coding strand of the) genome sequence. An important difference between RNA and DNA is the fact that RNA is single stranded, while DNA forms a double helix. Transcription is initiated by the recognition of sequence features or signals on the DNA located immediately upstream of a gene (the promoter region) by so-called transcription factors. Transcription factors are proteins that recognize these TFBS (transcription factor binding sites). Once they have bound to the DNA, an RNA polymerase is attracted and transcription starts. When a signal for cleavage is encountered the transcript is cleaved and a poly-A tail (chain of several adenine nucleotides) is appended. The product of this first process is pre-mRNA, which will be further processed in the subsequent steps.

Splicing Pre-mRNA contains internal non-coding sequences which are excised from the RNA by subsequent enzymatic reactions. These regions are called introns, the ones that remain are called exons. The excision involves the recognition of sequence features by so-called splicing factors and leads to mature mRNA (see Section 2.1.2 for more details). It should be noted that the choice of splice sites is not always deterministic, but subject to regulation. This is referred to as alternative splicing.

Translation In the last step (called translation) proteins are synthesized. Triplets of nucleotides (called codons) code for one amino acid. Translation is initiated at another sequence signal and continues until a stop codon (ATG) is encountered. The region upstream of the translation start is referred to as 5' UTR (untranslated region), the one downstream of the stop codon is called 3' UTR. What follows then is post-processing of proteins, as well as transport to their destination. After this brief overview of gene expression, sequence signals and regions are summarized in Figure 2.2.

The recognition of the following signals is involved in gene expression:

- Promoter
- Transcription Start
- Donor
- Acceptor
- Cleavage
- Translation Start
- Translation Stop

With the above signals it is possible to give a precise definition of the structure of a protein coding gene. Furthermore, the recognition of the above signals, plus the analysis of the composition of the various regions are exploited to construct high quality gene finders, such as *mGene*.

[1] Thymin in DNA is replaced by uracil in RNA.

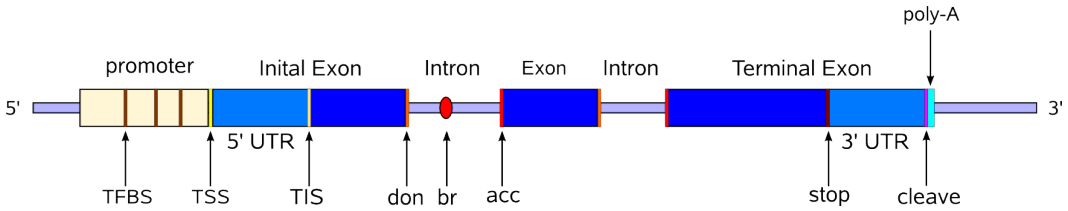


Figure 2.2: Simplified gene structure and positions of sequence signals relevant to gene finding. The promoter is a potentially long region upstream of the transcription start (TSS). The region contains transcription factor binding sites (TFBS). Not everything that is transcribed is also translated. The region between TSS and translation start (TIS) is referred to as 5' untranslated region (UTR). Eukaryotic genes often contain sequences that are excised (introns), which are bounded by the donor (don) on the 5' end and the acceptor (acc) on the 3' end. In between is the branch point (br), which is relevant for lariat formation. Translation is terminated at translation stop (stop), the region between translation stop and the cleavage site is called 3' UTR. After transcription, the poly-A tail is appended to the mRNA. It plays a role in mRNA stability as well as transport.

2.1.2 Splicing

In this work, special attention is given to the recognition of splice sites. Thus, the underlying process will be explained in more detail. Splicing plays an important role in gene expression, gene regulation, and protein diversity in eukaryotes.

The Mechanism After transcription of RNA precursors, a complex of splicing factors catalyzes the removal of introns from the pre-mRNA. Splicing can be roughly divided into two stages. In the first stage, sequence signals are recognized and the complex assembles at the splice sites. Then, after all components have assembled, cleavage and ligation reactions lead to the excision of the intron. The complex of splicing components is referred to as spliceosome. It consists of several proteins and five snRNPs (small nuclear ribonucleoprotein), U1, U2, U5, U4, and U6, which contain a small RNA bound by proteins. To assure the precise excision of introns, a number of signals located in both the intron and the exon have to be recognized. These signals interact with various parts of the spliceosomal complex. In particular, splicing is directed by the following main splicing signals, the 5' splice site or donor, the branch site, the polypyrimidine tract and the 3' splice site or acceptor, as shown in Figure 2.3. These sequence signals serve as binding sites for splicing factors. We utilize the presence of those signals to identify splice sites, with the help of machine learning techniques (see Section 2.2.4).

The assembly of the spliceosome is initialized by a number of binding reactions of splicing factors to the respective signals. U1 snRNP binds to the donor, the protein SF1 to the branch site, the U2 snRNP auxiliary factor U2AF large subunit to the polypyrimidine tract at the 3' end of the intron, and the U2AF subunit to the 3' splicesite. Subsequent recruitment of additional factors that interact with the ones already bound to the pre-mRNA leads to the

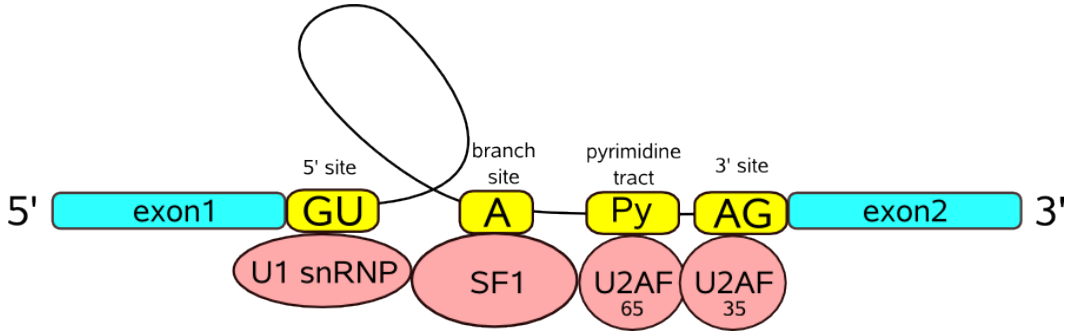


Figure 2.3: Assembly of the initial spiceosomal complex. Recognition of the 5' splice site by an RNA-RNA binding reaction of U1 snRNP, followed by a recruitment of U2AF to the polypyrimidine tract and the 3' splice site. SF1 interacts with the branch site sequence and bridges U1 with U2AF, making the first connection between the two splice sites (Lew00).

formation of the mature spliceosome (KRM05). The spliceosome brings together the 5' and 3' splice site, before any reaction can occur. The reaction with the 5' splice site involves formation of a lariat that joins the GU end of the intron via a 5'-2' linkage to the A at the branch site (see Figure 2.3). Then the 3' OH end of the exon attacks the 3' splice site, the exons are ligated and the intron is released as a lariat (Lew00).

2.2 Kernel Methods

One of the central building blocks of this work are kernels. In the following section we therefore give an intuitive explanation of the underlying concepts, as well as a few formal definitions. For a thorough, rather theoretical introduction to kernels, the reader is referred to (SBS99; SS02). Alternatively, (STV04; BHOS$^+$08) provide a comprehensive introduction to kernel methods with a focus on computational biology.

2.2.1 Similarity Measure

In machine learning, a kernel can be intuitively understood as a similarity measure, which must satisfy certain mathematical conditions. A simple example of a function that fulfills these conditions is the scalar product, or dot product. In the context of kernel methods, the dot product is referred to as *linear kernel*. Thus, an intuitive way of thinking of kernels is as a generalization of the dot product.

2.2.2 Kernel Trick

Consider a set $\{x_1, ..., x_n\} \in \mathcal{X}$ of arbitrary, possibly non-vectorial objects. A natural way of processing such objects in an algorithm would involve finding a vectorial representation. Therefore, a first step would be to come up with a function $\phi : \mathcal{X} \rightarrow \mathcal{R}^d, d \in \mathbb{N}$ that maps

these objects to a vector space, which we will refer to as feature space. The successive steps of the algorithm could then deal with the images $\phi(x_1), ..., \phi(x_n)$ rather than the objects themselves.

The problem with this approach is, that such a mapping can lead to very high dimensional feature spaces. In particular, when attempting to capture all traits of a complex object, such as a sequence, by a vectorial representation, the feature space becomes often extremely high-dimensional, which leads to an unacceptable increase in memory usage and computation time.

Kernel methods employ a different strategy. To circumvent the problem of high-dimensional feature spaces, kernels avoid an explicit mapping ϕ by providing a way to *compare* two arbitrary objects directly with a real-valued comparison function k. Also, it is sometimes easier to define a comparison function between two objects than to find an explicit representation in a vector space. For instance, consider sequence data, where a vectorial representation is not obvious, while plenty of methods for sequence comparison exist.[2] With the use of kernels, the mapping ϕ is solely implicit. Kernels can be formally defined as follows:

Given a mapping $\Phi : \mathcal{X} \to \mathcal{H}$, which maps elements x from input space $\mathcal{X}$ to a dot product space $\mathcal{H}$, a kernel corresponds to the dot product in feature space (SS02). In the context of machine learning, $\mathcal{H}$ is often referred to as Reproducing Kernel Hilbert Space (RKHS). For an exact definition, the reader is referred to (SS02).

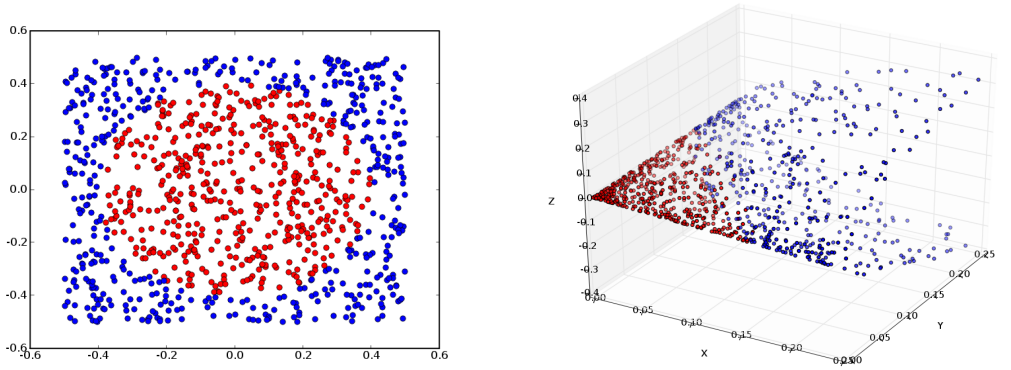
$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle \tag{2.1}$$

Example Consider the toy example in Figure 2.4. We have points of two classes (red, blue) which live in a two dimensional input space. Say, we would like to classify the points by their class, ideally with a simple rule. Clearly, there exists straight line that separates the two classes. Thus, we define a mapping $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$, $\Phi : (x_1, x_2) \to (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ which maps our points from two dimensional input space into a three dimensional feature space. In this space, we can now separate the two classes by a plane.

One can show, that a polynomial kernel of degree two can be used to compute the dot product between two points in the three-dimensional feature space, without explicitly performing the mapping ϕ, if we define the kernel as follows:

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^2$$

[2] However, not all comparison algorithms give rise to valid kernels.



(a) not linearly separable in $\mathbb{R}^2$ (b) linearly separable after mapping to $\mathbb{R}^3$

Figure 2.4: Toy example illustrating the polynomial kernel

Proof.

$$\begin{aligned}\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle &= \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}^T \begin{pmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1'x_2' \end{pmatrix} \\ &= x_1^2x_1'^2 + x_2^2x_2'^2 + 2x_1x_1'x_2x_2' \\ &= (x_1x_1' + x_2x_2')^2 \\ &= \langle \mathbf{x}, \mathbf{x}' \rangle^2\end{aligned}$$

□

Our kernel k is simply the square of the dot product in input space. Thus, we can compute similarities between points in feature space, without having to perform an explicit mapping Φ. This allows us to compute similarities between objects efficiently, which would quickly become computationally infeasible for non-trivial problems.

2.2.3 Encapsulation of data and non-linearity

The use of kernels has two important consequences. In case of a vectorial input space, they can be used to implicitly map examples from input-space to another vector space of potentially very high dimension. Therefore, an important aspect of kernels is that they provide a convenient way of introducting non-linearity into otherwise linear algorithms. An example where this could be useful was presented in Figure 2.4. Furthermore, in case of non-vectorial data, the kernel avoids an explicit mapping to a vector space, by directly comparing objects and thus defining a similarity measure. In either case, the data can now be represented as a set of pairwise comparisons. For a set of n objects, $n \times n$ comparisons represents our dataset. This matrix of comparisons is referred to as Kernel matrix, or Gram matrix.

Definition: Kernel Matrix Given a function $k : \mathcal{X}^2 \to K$ (where $K = \mathbb{C}$ or $K = \mathbb{R}$) and patterns $x_1, ..., x_n \in \mathcal{X}$, the $n \times n$ matrix K with elements

$$K_{ij} := k(x_i, x_j)$$

is called the Kernel matrix (or Gram matrix) of k with respect to $x_1, ..., x_n$.

Definition: Positive Definite Matrix A complex $n \times n$ matrix K satisfying

$$\sum_{i,j} c_i c_j K_{ij} \geq 0$$

for all $c_i \in \mathbb{C}$ is called positive definite (SS02).

Definition: Positive Definite Kernel Let $\mathcal{X}$ be a non-empty set. A function k on $\mathcal{X} \times \mathcal{X}$ that gives rise to a positive definite Gram matrix for all $n \in \mathbb{N}$ and all $x_1, \ldots, x_n \in \mathcal{X}$, is called a positive definite kernel.

The above definitions shed light on the question which conditions have to be satisfied such that a pairwise comparison function k is considered to be a valid kernel.

From the viewpoint of software engineering, kernels allow an elegant encapsulation of arbitrary objects into machine learning algorithms. If the algorithm knows about the kernel matrix $k_{i,j}$, it can be agnostic to whether the underlying objects are vectors, sequences, images, graphs, or of any other structure. This point is worth being stressed in the context of computational biology, as one encounters objects of diverse structure, such as DNA and protein sequences, chemical compounds in graph representation, or high-dimensional vectorial data from microarrays, to name a few.

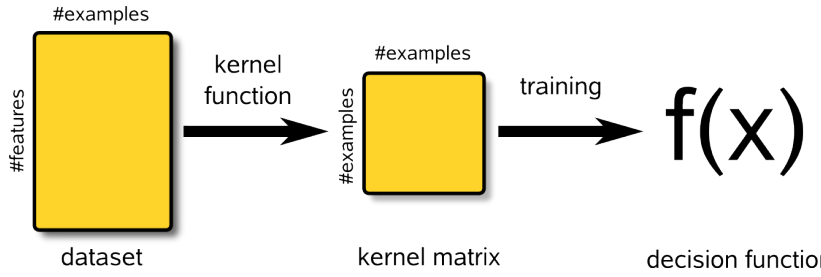


Figure 2.5: Encapsulation of data from the algorithm. The kernel encapsulates data from the algorithm and thus allows a modular design of machine learning software.

As shown in Figure 2.5 kernel methods consist of two components. The first component, the kernel, is responsible for embedding the dataset in feature space, while the second component, the learning algorithm, attempts to discover linear patterns in that space. One speaks of *kernelization*, when a formerly linear algorithm (i.e. formulated in terms of the dot product) is generalized by the use of kernels. A prominent example of a classic algorithm, that has been *kernelized* is the kernel-principle component analysis (kernel-PCA) (SSM99).

2.2.4 String Kernels

The main idea of string kernels is to define a similarity measure between two strings, based on the substrings they have in common. In the following, strings of length k will be referred to as k-mers. Clearly, the more k-mers occur in both sequences at corresponding position, the more similar the sequences are. Of particular interest to this work is the the so-called *Weighted Degree* kernel (RS04), which defines a similarity between two example sequences $\mathbf{x}$ and $\mathbf{x}'$ of fixed length L by counting co-occurring substrings in both sequences at the same position:

$$k_\ell^{\mathrm{wd}}(\mathbf{x}, \mathbf{x}') \quad = \quad \frac{1}{L} \sum_{l=1}^{L-l+1} \sum_{d=1}^{\ell} \beta_d \mathbf{I}\left(\mathbf{x}_{[l:l+d]} = \mathbf{x}'_{[l:l+d]}\right) \tag{2.2}$$

where I is an indicator function, $\mathbf{x}_{[l:l+d]}$ is the substring of length d of $\mathbf{x}$ at position l and $\beta_d = 2\frac{\ell-d+1}{\ell^2+\ell}$ is the weighting of the substring lengths. The weights compensate for the fact that longer k-mer are less likely per se.

2.3 Support Vector Machines

2.3.1 Motivation

Many machine learning algorithms have been studied during the last decades, amongst them prominent algorithms like Neural Networks that have played a role since the 1970s. However, during the 90s a machine learning method called *Support Vector Machine* became popular. An SVM is a typical kernel method, where an otherwise linear algorithm can be extended by the use of non-linear kernels. Several reasons were responsible for the success of the SVM. One could argue that one of the central reasons was that statistical learning theory (Vap00) provides probabilistic bounds, which guarantee a certain performance of the SVM. In a nutshell, (Vap00) were able to show that pursuing the large margin approach of the SVM would lead to good generalization performance. Another important reason was that the training of the SVM corresponds to solving a convex optimization problem, for which well-understood solutions exist.

Supervised Learning The SVM is a supervised learning approach.[3] In the supervised learning scenario, samples are drawn from a joint distribution $P(X, Y)$ of objects and labels. The goal is to learn a predictor $f : \mathcal{X} \to \mathcal{Y}$ from a set S of labeled training examples. In this work, we focus on a special case: binary classification, where each object belongs to one of two classes, indicated by the label $y \in \{-1, +1\}$. The SVM attempts to separate two classes by a linear (but kernelizable) discrimination function of the form $f(x) = sign(\mathbf{w}^T\mathbf{x} + b)$. The question arises *how* f can be learned from data.

Generative vs. Discriminative In supervised learning one can distinguish generative and discriminative methods. While discriminative approaches such as the SVM focus solely on

[3] However, the *one-class SVM* allows for density estimation.

the discrimination boundary between classes, generative models try to estimate the underlying distributions, which would allow the *generation* of data from the learned model. This means that, whereas generative classifiers learn a model of the joint probability $P(X, Y)$, a discriminative classifier focuses on the posterior $P(Y|X)$ directly (see Figure 2.6).

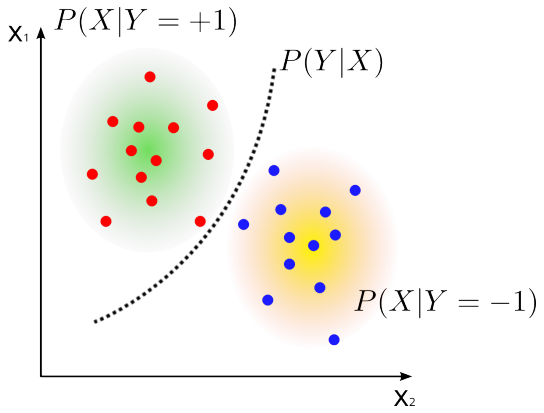


Figure 2.6: Generative vs. discriminative. Discriminative models focus on the separation boundary $P(Y|X)$. Generative classifiers learn a model of the joint probability $P(X, Y)$, which is shown class-wise in this Figure.

One can now argue that, given the prior $P(X)$ one can always obtain the posterior $P(Y|X)$ from the joint probability $P(X, Y)$ by using Bayes' Rule: $P(Y|X) = P(X, Y)/P(X)$. Thus, if we are only interested in classification, generative models take a costly detour. In such cases, (Vap00) recommend to prefer discriminative classifiers over generative classifiers, in order to avoid solving a more general problem as an intermediate step to solving a simpler problem. Intuitively speaking, discriminative models perform better if a limited amount of data is available, because finding the optimal separator generally has a "*lower degree of freedom*" than estimating the underlying distributions.

2.3.2 Optimization Problem

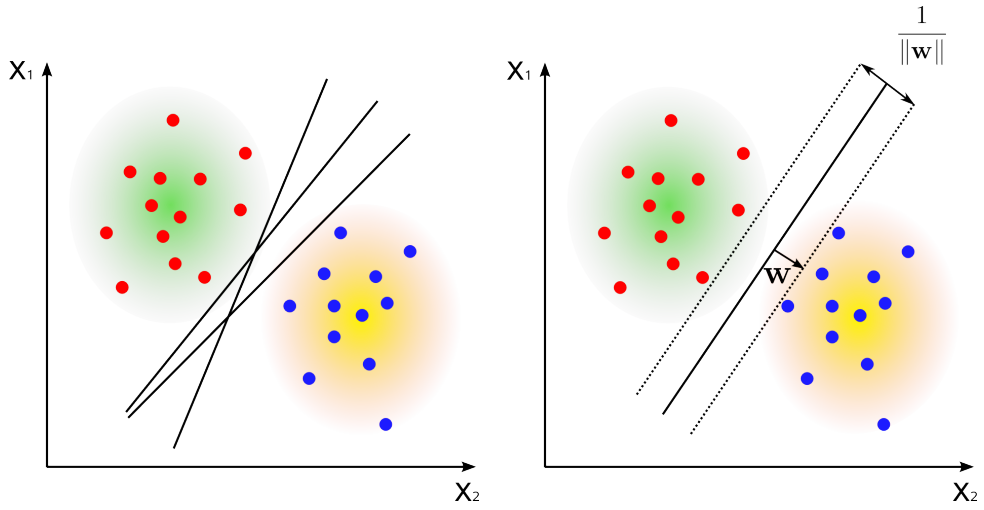
First, a few definitions are necessary. The SVM attempts to infer a separation boundary to discriminate examples according to their class. This boundary is commonly referred to as hyperplane. A hyperplane is a $(n-1)$-dimensional subspace of a n-dimensional space. A hyperplane is defined by a normal vector $\mathbf{w}$ that determines the orientation of the plane and a constant b that determines the distance from the origin. In the following, $\mathbf{w}$ and b will be referred to as *primal variables* of the SVM.

The training algorithm of the SVM can be formulated as a convex optimization problem. This is advantageous because there exists one global optimum[4] and we know how to obtain it. The ideas that motivate the formulation of the SVM will be briefly discussed.

[4]Neglecting saddle points which could lead to arbitrarily many optima of the same value.

Large margin

Other than for instance the perceptron, we are not looking for just any hyperplane that separates our two classes, but the hyperplane that does so with a *large margin*. This is not only in agreement with our intuition, but is also motivated by statistical learning theory (Vap00).



(a) Perceptron: Any hyperplane that separates the data is an optimal solution

(b) Large margin classifier: The optimal solution has the largest possible margin

Figure 2.7: Example illustrating the difference between the perceptron algorithm and the large margin approach.

It can be shown that the "thickness" of the margin equals $\frac{1}{\|\mathbf{w}\|}$. Thus, the SVM can be formulated as the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ &\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0 \ \forall i \in [1, n] \end{aligned} \tag{2.3}$$

The term $\mathbf{w}^T\mathbf{w}$ that enforces the large margin is referred to as regularizer. From a functional point of view, it limits the complexity of the function class that is taken into account. An intuitive explanation of Equation 2.3 is the following. We attempt to find the hyperplane with the largest margin, given that all training points are to be located on the correct side (in the half-space that corresponds to their class label) of the hyperplane. The above formulation is generally referred to as *hard-margin SVM* because a violation of the constraints is prohibited under all circumstances.

Soft margin

In practice, however, most problems are not linearly separable. Therefore one needs to relax the formulation of the SVM, such that a solution exists, even for problems with overlapping classes. This is done by introducing so-called *slack terms* ξ_i. In principle a penalty is introduced into the objective of the optimization problem, for points that lie on the wrong side of (or within) the margin. This term is referred to as *loss*-term. In particular, the SVM incorporates the so-called *hinge* loss. For an in depth discussion of loss functions, including the *hinge* loss, the reader is referred to (CBL06). The hardness of the margin (i.e. slope of the cost function) can be set using a hyperparameter C.

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i \\ \text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) + \xi_i - 1 \geq 0 \ \forall i \in [1,n] \\ \xi_i \geq 0 \ \forall i \in [1,n] \end{aligned} \tag{2.4}$$

An abstract view

We have introduced all ingredients necessary to formulate the SVM in an abstract manner. It has become clear that a solution to the optimization problem constitutes a trade-off between obtaining a large margin (which is enforced by the regularization term Ω) and minimizing the cost induced by the loss term L, if constraints are violated.

Thus, the SVM can be written as follows (Cha07):

$$\min_{\mathbf{w},b} \Omega(\mathbf{w}) + C\sum_{i}^{N} L(\mathbf{w}, b; \mathbf{x}_i, y_i)$$

Either term represents a potential target to introduce conditions relevant to *domain adaptation* into the formulation of the SVM. This will be discussed in Chapter 3.

Solving the SVM

Given the above optimization problem, one can derive the *dual* formulation of the SVM by first forming the *Lagrangian* using *Lagrange multipliers* α_i and then eliminating the primal variables $(\mathbf{w}, b, \xi)$. Details will be skipped at this point, a thorough discussion of the derivation of the dual can be found in (SS02). As a result, one obtains the following formulation:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \gamma \text{ for all } i = 1, \ldots, n \end{aligned} \tag{2.5}$$

It is important to point out that we only have to deal with the dual variables α_i in this formulation. Furthermore, the dual only depends on the dot product and can thus be generalized using the *kernel trick*, by setting $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = k(\mathbf{x}_i, \mathbf{x}_j)$. The solution of an SVM yields a vector of α_i, assigning a weight to each training example x_i. Examples x_i which are assigned a non-zero weight $\alpha_i \neq 0$ are termed *support vectors*. The normal vector $\mathbf{w}$ of the resulting separation boundary (a hyperplane) is entirely defined by these support vectors, as follows (note that b can also be obtained from the dual solution):

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

where y_i denotes the label of example $\mathbf{x}_i$. As mentioned before, this gives rise to a predictor function f, which can be used to classify unseen examples $\mathbf{x}$:

$$f(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x} + b)$$

2.4 Evaluation

2.4.1 Cross-validation

To evaluate a trained predictor, a part of the dataset is set aside before training. This part will be referred to as evaluation set. It is of great importance that none of the data points in the evaluation set are used in training, because an overfitted predictor will report a small error on its training data, while it will most likely perform poorly on unseen data. Because we are ultimately interested in the ability of our classifier to perform well on unseen data, we measure performance on the evaluation set to estimate how well our trained predictor generalizes.

A common technique that formalizes the splitting of data into training and evaluation sets is the n-fold cross-validation. Here, the dataset is split into n parts. The predictor is trained on $n - 1$ parts, and performance is measured on the remaining part. This is repeated until each part has been used as evaluation set once.

If we use cross-validation to perform hyperparameter (i.e. C in case of an SVM) optimization, the hyperparameters are selected to achieve optimal performance on the evaluation set. Thus, the hyperparameter are "fitted" to the evaluation set as well. Therefore an additional

part of the dataset has to be set aside that will not be used in hyperparameter tuning. This set is called *test set* and will be used to evaluate performance only after hyperparameters have been optimized.

2.4.2 Performance Measures

We employ two common performance measures for evaluation purposes. The most common one in machine learning is the *area under the Receiver Operator Characteristic curve* (auROC). Due to its problems with unbalanced datasets, another related measure called *area under Precision Recall curve* (auPRC) is used.

	actual positive	**actual negative**
predicted positive	true positive (TP)	false positive (FP)
predicted negative	false negative (FN)	true negative (TN)

Table 2.1: Confusion matrix of correct and incorrect predictions (DG06).

The following metrics are relevant to the performance measures that were used.

Precision Precision is defined as the fraction of actual positive examples among all positive predictions (see also Table 2.1).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.6}$$

True Positive Rate The TPR is defined as the faction of correctly identified positive examples among all positively labeled examples. It is also referred to as sensitivity, or recall.

$$\text{TPR} = \text{Recall} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

False Positive Rate The FPR is the fraction of examples that were incorrectly predicted as positives, among all negatives. The FPR is often referred to as $(1 - \text{Specificity})$

$$\text{False Positive Rate} = (1 - \text{Specificity}) = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{2.8}$$

auROC

A ROC curve is obtained by plotting sensitivity (or TPR) versus $1 - \text{Specificity}$ (or FPR). This is done for varying cutoffs. The auROC score is then obtained by calculating the area under the ROC curve.

auPRC

The area under Precision Recall Curve is a related measure. Because the datasets used in this work are quite unbalanced (see Chapter 5), the auPRC is better suited to obtain a

good estimate of the prediction accuracy (DG06). The auPRC score is obtained by plotting precision versus recall (as defined above). Again, this is done for varying cutoffs.

3 Domain Adaptation

3.1 Overview

One of the central assumptions in statistical learning theory is that training and test data are drawn from the same underlying distribution. However, in many cases this assumption does not hold. In supervised learning, the bottleneck in terms of predictive performance is often the lack of sufficient labeled training data. Let us assume, that we have a large body of data available on some domain, while we are really interested in a different but related domain, where only little data is available. Following (BdBCP07), the domain with sufficient training data will be referred to as source domain, whereas the domain with none or little labeled training data will be called target domain. The domain adaptation scenario is encountered in numerous applications of machine learning. Consider the following examples:

- Spam filter: A spam filter is trained on data from a large body of users. In case of a new user, the predictor has to be adapted to a similar but possibly different distribution of emails.
- Acoustic models: In speech recognition, an acoustic model that is trained by one speaker but used by another has to be adapted to a new domain.
- Face recognition: A predictor is trained on data that was obtained under particular lighting conditions, but will be used under different conditions as well.
- Splice site prediction: For newly sequenced organisms, nearly no labeled training data exists, whereas a large body of data is available for related organisms.

One can distinguish two cases of domain adaptation. The first is a purely supervised learning scenario, where a large corpse of labeled data is available on the source domain and a limited amount of labeled data is available on the target domain. Following (Jia08), this case will be termed *supervised domain adaptation*. Conversely, the case where we do not have any labeled training examples in our target domain will be referred to as *unsupervised domain adaptation*. Unless stated otherwise, the supervised case will be assumed.

3.2 Distributional Point of View

In the following, the domain adaptation scenario is analyzed from a distributional point of view. In a supervised learning scenario, samples are drawn from a joint distribution $P(X, Y)$ of examples X and labels Y. Given a source domain, and a target domain, the question arises in which way $P_S(X, Y)$ can differ from $P_T(X, Y)$, where $P_S(X, Y)$ is the joint distribution with respect to the source domain and $P_T(X, Y)$ is the joint distribution with respect to the

target domain. The following factorization gives the answer:

$$P(X, Y) = P(X) \cdot P(Y|X)$$

It becomes clear that the distributions $P_S(X, Y)$ and $P_T(X, Y)$ can vary in different ways.

- $P_S(X) \neq P_T(X)$: Here, we assume that the underlying mechanism, which generated labels to examples remains unchanged (i.e. $P_S(Y|X) = P_T(Y|X)$). However, marginal distributions $P(X)$ might differ between source and target domains. This scenario is commonly known as *covariate shift*. For a discussion of the *covariate shift* scenario (including problematic aspects of the assumptions) see (Shi00).

  For instance, consider the spam filter example. The distribution of words between the emails of, say, a journalist, who is signed up to several political mailing lists, and a computer scientist, who is signed up to various tech-related mailing lists, are likely to be quite different. Some words will be found more frequently in the inbox of one, others will be more often encountered in the email account of the other. However, in the *covariate shift* scenario one assumes, that both individuals agree on which emails are considered to be spam, and which are not. That way, the underlying mechanism $P(Y|X)$ is identical for both domains.
- $P_S(Y|X) \neq P_T(Y|X)$: In this case one also allows for differences in the underlying mechanisms $P(Y|X)$ between domains. Again, in terms of the spam example, this would mean that one person considers some emails to be spam that are interesting to another individual.

The above observations not only help giving a better understanding of the domain adaptation problem, but also help to organize the methods that will be discussed in the following section.

3.3 Methods and Extensions

In the following, we will present methods from literature, discuss some extensions and present own developments. Methods are explained in the context of their respective category to facilitate their understanding and to expose their relations to each other.

3.3.1 Baseline Methods

As baseline methods, we considered three obvious methods to compare against.

- *SourceOnly*: Only examples from the source dataset are used for training, ignoring data from the target domain.
- *TargetOnly*: Accordingly, the classifier is only trained on target examples, ignoring data from the source domain. This method gives an idea of the achievable performance, when no source data is available.

- *SimpleUnion*: The third baseline method is to train a classifier on the union of the source and target dataset, assuming that we have labeled training data on both.

Note that, if a soft-margin SVM is employed, all of the above baseline methods depend on only one hyperparameter C (see Section 2.4.1).

3.3.2 Instance Weighting

A possible way of addressing the domain adaption problem is to assign example-dependent weights. These weights are generally incorporated into the loss function of a training method (see Section 2.3.2).

In this section, three methods for instance weighting are presented. First, we introduce a trivial reweighting method *Combined*, where example weights are set by domain. Furthermore, we present *Instance Pruning* which is suited in case of differing functional relations (e.g. $P_S(Y|X) \neq P_T(Y|X)$) and lastly *Kernel Mean Matching*, which is based on the *covariate shift* assumption.

Weighted Combination (*Combined*)

This method can be regarded a straightforward extension of the baseline method *SimpleUnion*. The idea is to allow a trade-off between source and target domains by assigning domain dependent weights.

The relative importance of each domain is integrated into the loss term of the SVM and can be adjusted by setting domain-dependent cost parameters C_S and C_T for the m and n training examples from the source and target domain, respectively:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C_S \sum_{i=1}^{m} \xi_i + C_T \sum_{i=m+1}^{m+n} \xi_i \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \Phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \quad \forall i \in [1, m+n] \\ & \xi_i \geq 0 \quad \forall i \in [1, m+n] \end{aligned} \tag{3.1}$$

If the cost for one domain is set significantly higher than for the other domain, the resulting solution is clearly dominated by points from the domain with the higher cost.

This method has two model parameters (C_S, C_T) and requires training on the union of the training sets, which means that within-domain interactions (in the sense of comparisons) as well as between-domain interactions will be considered. Since the computation time of most classification methods increases super-linearly with the number of examples and exhaustive model selection requires to consider all combinations of parameters, this approach is computationally demanding.

Instance Pruning (*Pruning*)

If we allow for differing conditionals (i.e. $P_S(Y|X) \neq P_T(Y|X)$), the same observation $x \in X$ might be assigned a different label y, depending on the domain.

To deal with this scenario, (JZ07) have presented a heuristic which removes *misleading samples* from the source dataset (i.e. sets the weights of the respective instances to zero). This is done by first training a classifier on labeled examples from the target domain. The classifier is then used to predict labels on examples from the source domain (of which we know the true labels). In instance pruning, one removes (some) examples from the source sample, if the predicted label does not match the true label. That way, we avoid using contradictory information in our training set (which is in itself not as problematic as it might appear, but also happens in the regular supervised learning case).

However, it should be pointed out that this approach can only be expected to work if a reasonable number of labeled target examples is available. Otherwise, one might end up pruning virtually random examples from the source dataset and thus discard valuable information. This method will not be discussed in the following as initial experiments did not lead to significant improvements. It is still mentioned here, because it constitutes one of the few methods available that explicitly focus on the scenario of differing conditionals.

Kernel Mean Matching (*KMM*)

The idea of Kernel Mean Matching is to reweight examples, such that the mean of distributions of examples become similar for source and target $P_S(X) = P_T(X)$ (GSH$^+$08).

As explained in Section 2.2, kernel methods map the data into a reproducing kernel Hilbert space (RKHS) by means of a mapping $\Phi : \mathcal{X} \to \mathcal{H}$ related to a positive definite kernel via $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$. Depending on the choice of kernel, the space of $\mathcal{H}$ may be spanned by a large number of higher order features of the data. In such cases, higher order statistics (e.g. variance, skewness) for a set of input points can be computed in $\mathcal{H}$ by simply taking the mean (i.e. the first order statistics). In fact, it turns out that for a certain class of kernels, the mapping

$$\mu : (\mathbf{x}_1, \ldots, \mathbf{x}_n) \mapsto \frac{1}{n} \sum_{i=1}^{n} \Phi(\mathbf{x}_i)$$

is injective (GBR$^+$07) — in other words, given knowledge of (solely) the mean (the right hand side), we can completely reconstruct the set of points. For a characterization of this class of kernels, see for instance (FGSS08). Generally speaking, the higher dimensional $\mathcal{H}$, the more information is contained in the mean.

As mentioned before, the goal of KMM is to match means of source and target distributions by reweighting source examples. To obtain weights $\beta_i \in \mathbb{R}^m$ for each source example x_i, the following optimization problem has to be solved (see (GSH$^+$08) for details):

$$\min_{\beta} \ \frac{1}{2} \beta^\top K \beta - \kappa^\top \beta \tag{3.2a}$$

$$\text{subject to } \beta_i \in [0, B] \text{ and } m(1-\epsilon) \leq \sum_{i=1}^{m} \beta_i \leq m(1+\epsilon). \tag{3.2b}$$

where $K_{ij} := k(x_i, x_j)$ and $\kappa_i := \frac{m}{n} \sum_{j=1}^{n} k(x_i, x'_j)$. The first constraint limits the scope of discrepancy between source $P_S(X)$ and target distributions $P_T(X)$, while the second constraint

ensures that the reweighted source distribution (i.e. $\beta(x)P_T(x)$) is a probability distribution. Note that a constant B can be used to determine how strongly the source dataset is reweighted.

Other than (GSH+08), we do have a certain amount of labels also for the target distribution. We make use of them by performing the reweighting separately for each class $y \in \{\pm 1\}$. This means that instead of attempting to reweight examples from the source sample, such that $P_S(X) = P_T(X)$, we separately reweight examples from the positive class to obtain $P_S(X|Y = +1) = P_T(X|Y = +1)$ and examples from the negative class to obtain $P_S(X|Y = -1) = P_T(X|Y = -1)$.

The resulting weight vector β can be readily plugged into the loss-term of the SVM. One can either train an SVM on the reweighted source examples only, or use one of the other methods presented in this chapter (e.g. *Combined*, *Convex*) to combine the weighted source examples with labeled target examples. While the former allows for unsupervised domain adaptation, the latter was found to perform better.

3.3.3 Change of Representation

With Structural Correspondence Learning (SCL), (BMP06) presented another approach to the problem of domain adaptation. The idea behind SCL is to change the representation of examples x, such that source and target distributions become similar.

It has been pointed out (Jia08) that a change of representation of X can affect the marginal distribution $P(X)$, as well as the conditional distribution $P(Y|X)$. In short, the goal is to find a representation of examples, such that $P_T(X, Y) = P_S(X, Y)$.

In the following we will introduce a KMM-based method, which performs a translation instead of a reweighting of examples. Furthermore, we will present a method that attempts to apply the idea of SCL to the Weighted Degree Kernel.

Kernel Mean Matching by Translation (*Shift*)

In Section 3.3.2, we introduced KMM and discussed why matching means is beneficial in high-dimensional feature spaces. The same idea motivates this method, however, means are matched using a different approach. In this variant, we do not reweight source examples, but rather we translate each example in the direction of the vector that connects the mean of the source examples and the mean of the target examples:

$$\hat{\Phi}(\mathbf{x}_j) = \Phi(\mathbf{x}_j) - \gamma \left(\frac{1}{m} \sum_{i=1}^{m} \Phi(\mathbf{x}_i) - \frac{1}{n} \sum_{i=m+1}^{m+n} \Phi(\mathbf{x}_i) \right) \qquad \forall j = m+1, \ldots, m+n.$$

This also leads to a modified source distribution that is statistically more similar to the target distribution and can thus be used to improve performance when training the target task. Again, we make use of labeled target data by performing the shift separately for each class $y \in \{\pm 1\}$:

$$\hat{\Phi}(\mathbf{x}_j) = \Phi(\mathbf{x}_j) - \gamma \left(\frac{1}{m_y} \sum_{i=1}^{m} [\![y_i = y]\!] \Phi(\mathbf{x}_i) - \frac{1}{n_y} \sum_{i=m+1}^{m+n} [\![y_i = y]\!] \Phi(\mathbf{x}_i) \right)$$

for all $j = m+1, \ldots, m+n$ with $y_j = y$, where m_y and n_y are the number of source and target examples with label y, respectively. Using a parameter γ, we can set how far source examples are translated. Analogously to the options available to *KMM* (see Section 3.3.2), the shifted examples can be used in different ways to obtain a final classifier.

K-mer Reweighting

This approach is specific to sequence signal prediction, as it is directly applied to features used in the Weighted Degree Kernel. In order to achieve a change of representation (BdBCP07; BMP06) in sequence analysis, we propose the method of reweighting k-mers (see Section 2.2.4) such that source and target samples become similar. While the idea of defining a reweighting to achieve distribution similarity is clearly related to KMM, the approach focuses on reweighting *features* rather than *examples*.

Recently, a method was proposed by (SZPR08) that helps the understanding of SVM classifiers that use the WD kernel: Position Oligomer Importance Matrices (POIMs). In a nutshell, POIMs assign an importance score to each k-mer at each position, quantifying how relevant it is for classification.

The idea behind k-mer reweighting is to use POIMs to assign a higher weight to features that are reported to be important in the POIMs of both domains. In the first step, POIMs are computed for source and target, resulting in two score vectors. In the next step, POIMs from source and target are multiplied. If we obtain a large positive product, we know that the respective k-mer is important in both source and target.

The computed weight will be used to reweight the importance of the individual k-mers in the Weighted Degree Kernel. The idea for this method arose towards the end of this project, so there was in-sufficient time to fully implement a working version of it. It is still mentioned here because it constitutes an interesting way of approaching the problem and is a possible direction of future research.

3.3.4 Methods using Priors

The idea of prior methods is to integrate prior information about the model $\mathbf{w}$ when seeking an optimal model parameter $\mathbf{w}^*$. In domain adaptation we can draw prior information from the source domain. Most research concerning this approach utilizes a Bayesian framework where a Bayesian prior $P(\mathbf{w})$ can directly be integrated into the maximum a posterior (MAP) formulation (CA06).

As we focus on SVM classifiers (see Section 2.3.1), prior information will be encoded as the normal vector of the separating hyperplane on a previous solution.

Source-regularized Target (*Regtrick*)

The idea of this method motivated the thorough investigation of the other related methods and it was the first to be implemented. Therefore, special attention is given to this approach.

The geometric interpretation behind the regularization trick is the following: Assume, that we want to train an SVM with only a few labeled training examples available. Furthermore, let the features space be fairly high dimensional (as induced by string kernels). Thus, the resulting solution might not be very robust. To obtain a more robust solution, the idea

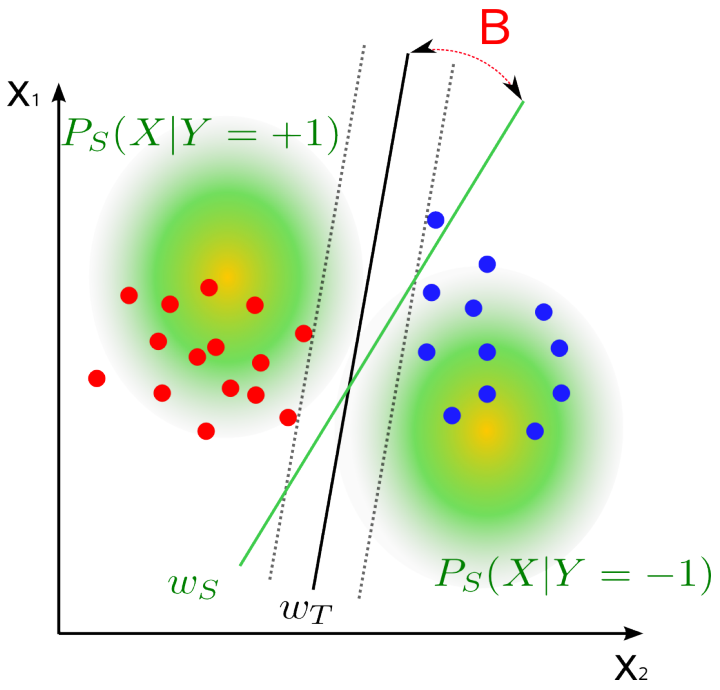


Figure 3.1: Example illustrating the geometric interpretation. The points in the figure denote examples drawn from an unknown target distribution. We assume to have a large amount of source data, such that the empirical distribution approximates the true source distribution $P_S(X)$ well (drawn in green). Furthermore, source and target distributions are similar, but not identical. In the example, a few outliers dominate the resulting solution $\mathbf{w}_T$. By using the classifier from source $\mathbf{w}_S$ as a prior, one can obtain a more robust solution. The influence of the prior solution $\mathbf{w}_S$ can be set via a parameter B.

is to use the normal vector of a previously learned solution with sufficient training data to *stabilize* our normal vector. The new solution should not only have a large margin, but should also result in a normal vector that is similar to the previously learned solution on a related problem (see Figure 3.1). One can include this criterion into the regularization term of the optimization problem (see Section 2.3.2).

This leads to the following modified formulation of the SVM:

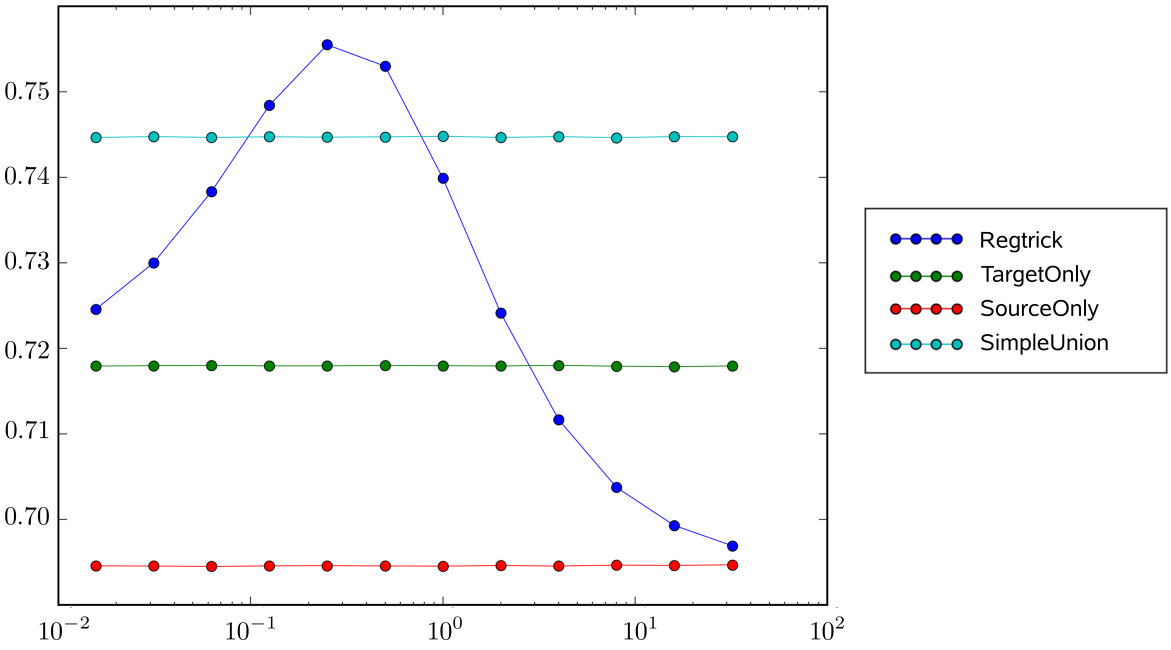


Figure 3.2: To illustrate the dependency on the hyper-parameter B, this plot shows classification performance (auROC) on the y-axis plotted versus different values of B on the x-axis.

$$\mathbf{\Omega}_{\text{reg}}(\mathbf{w}_T) = \frac{1}{2}\|\mathbf{w}_T\|^2 - B \cdot s(\mathbf{w}_T, \mathbf{w}_S) \tag{3.3}$$

where s is a similarity measure and B denotes a hyper-parameter that regulates the trade-off between a large margin and similarity to the old solution $\mathbf{w}_S$, which carries our *prior knowledge* about the solution. In the following, we will employ the dot product as similarity measure. Note, however, that other choices of s are possible.

In practice the optimal parameter B depends on the distance between the two domains as well as the number of available target data (which also holds true for the trade-off parameters of other methods). In Figure 3.2, we illustrate the dependency of prediction performance upon the hyper-parameter B. If we choose B to be close to zero, then our prior solution $\mathbf{w}_S$ plays no role at all, and the resulting solution will be essentially the same as $\mathbf{w}_T$. Analogously, the solution is dominated by $\mathbf{w}_S$ if B is set to infinity. Clearly, an improvement can only be achieved in between these two extrema, as shown in Figure 3.2.

An Efficient Formulation In order to be able to solve the modified optimization problem with a standard SVM solver, one has to derive the dual formulation of an SVM that employs Equation 3.3 as regularizer. The explanation of the mathematical techniques that were used for the derivation is beyond the scope of this work. The interested reader is referred to (BV04) which contains an excellent introduction to the field.

We start with the primal formulation using the regularization term from Equation 3.3.

Primal:

$$\begin{aligned} \min_{\mathbf{w}_T,\xi} \frac{1}{2}\mathbf{w}_T^T\mathbf{w}_T + C\sum_{i=1}^{n}\xi_i - B\mathbf{w}_T^T\mathbf{w}_S \\ \text{s.t. } y_i(\mathbf{w}_T^T\mathbf{x}_i + b) + \xi_i - 1 \geq 0 \ \forall i \in [1,n] \quad (3.4) \\ \xi_i \geq 0 \ \forall i \in [1,n] \quad (3.5) \end{aligned}$$

The Lagrangian of this is given by:

$$L(\mathbf{w}_T,\xi,\alpha) = \frac{1}{2}\mathbf{w}_T^T\mathbf{w}_T + C\sum_{i=1}^{n}\xi_i - B\mathbf{w}_T^T\mathbf{w}_S - \underbrace{\sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}_T^T\mathbf{x}_i + b) + \xi_i - 1)}_{(3.4)} - \underbrace{\sum_{i=1}^{n}\mu_i\xi_i}_{(3.5)}$$

By expansion, we obtain:

$$L(\mathbf{w}_T,\xi,\alpha) = \frac{1}{2}\mathbf{w}_T^T\mathbf{w}_T + C\sum_{i=1}^{n}\xi_i - B\mathbf{w}_T^T\mathbf{w}_S - \sum_{i=1}^{n}\alpha_i y_i \mathbf{w}_T^T\mathbf{x}_i - \sum_{i=1}^{n}\alpha_i y_i b - \sum_{i=1}^{n}\alpha_i\xi_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\mu_i\xi_i \quad (3.6)$$

Partial differentiation with respect to the primal variables:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}_T} &= \mathbf{w}_T - B\mathbf{w}_S - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i \\ \frac{\partial L}{\partial \xi} &= \mathbf{C} - \alpha - \mu \\ \frac{\partial L}{\partial b} &= -\alpha^T\mathbf{y} \end{aligned}$$

We find the stationary points by setting the partial derivatives to zero:

$$\mathbf{w}_T = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i + B\mathbf{w}_S \tag{3.7}$$

$$\mathbf{C} - \alpha - \mu = 0 \tag{3.8}$$

$$\alpha^T \mathbf{y} = 0 \tag{3.9}$$

Substitute $\mathbf{w}_T$ in equation (3.6) using equation (3.7) :

$$\begin{aligned} L =& \frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i + B\mathbf{w}_S)^T(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + B\mathbf{w}_S) + C\sum_{i=1}^{n} \xi_i - B(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i + B\mathbf{w}_S)^T\mathbf{w}_S \\ & - \sum_{i=1}^{n} \alpha_i y_i (\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + B\mathbf{w}_S)^T\mathbf{x}_i - \sum_{i=1}^{n} \alpha_i \xi_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \mu_i \xi_i \end{aligned}$$

We move transpose inside, where possible:

$$\begin{aligned} L =& \frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T + B\mathbf{w}_S^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + B\mathbf{w}_S) + C\sum_{i=1}^{n} \xi_i - B(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T + B\mathbf{w}_S^T)\mathbf{w}_S \\ & - \sum_{i=1}^{n} \alpha_i y_i (\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T + B\mathbf{w}_S^T)\mathbf{x}_i - \sum_{i=1}^{n} \alpha_i \xi_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \mu_i \xi_i \end{aligned}$$

Terms are expanded, where possible:

$$\begin{aligned} L =& \frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) + B(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T\mathbf{w}_S) + \frac{1}{2}B^2\mathbf{w}_S^T\mathbf{w}_S \\ & + C\sum_{i=1}^{n} \xi_i - B\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T\mathbf{w}_S - B^2\mathbf{w}_S^T\mathbf{w}_S \\ & - \sum_{i=1}^{n} \alpha_i y_i (\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T\mathbf{x}_i + B\mathbf{w}_S^T\mathbf{x}_i) - \sum_{i=1}^{n} \alpha_i \xi_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \mu_i \xi_i \end{aligned}$$

Terms canceled out where possible, simple sums in vector notation:

$$\begin{aligned} L =& \frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) - \frac{1}{2} B^2 \mathbf{w}_S^T \mathbf{w}_S \\ & - (\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) - (B \sum_{i=1}^{n} \alpha_i y_i \mathbf{w}_S^T \mathbf{x}_i) + \sum_{i=1}^{n} \alpha_i + \xi^T \mathbf{C} - \xi^T \alpha - \xi^T \mu \end{aligned}$$

Further summerizations, factor out ξ^T :

$$L = -\frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) - B(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{w}_S) + \sum_{i=1}^{n} \alpha_i + \xi^T(\mathbf{C} - \alpha - \mu) - \frac{1}{2} B^2 \mathbf{w}_S^T \mathbf{w}_S$$

Use equation (3.8) to eliminate $\xi^T(\mathbf{C} - \alpha - \mu)$, drop constant $\frac{1}{2} B^2 \mathbf{w}_S^T \mathbf{w}_S$

$$L = -\frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T)(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) - B(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{w}_S) + \sum_{i=1}^{n} \alpha_i$$

Further simplifications yield the dual Lagrangian:

$$L = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^{n} \alpha_i (B y_i \mathbf{x}_i^T \mathbf{w}_S - 1) \tag{3.10}$$

Hence, we obtain the following dual optimization problem:

$$\begin{aligned} \max_{\alpha} & -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^{n} \alpha_i (B y_i \mathbf{x}_i^T \mathbf{w}_S - 1) \\ & \text{s.t. } 0 \leq \alpha_i \leq C \ \forall i \in [1, n] \\ & \alpha^T \mathbf{y} = 0 \end{aligned} \tag{3.11}$$

Note that this formulation explicitly relies on the normal vector $\mathbf{w}_S$ of the previously trained SVM. As we deal with extremely high dimensional feature spaces, this is often infeasible. The remaining steps address this issue.

We use the representer theorem $\mathbf{w}_S = \sum_{j=1}^{m} \alpha'_j y'_j \mathbf{x}'_j$ to substitute $\mathbf{w}_S$ in equation (3.10), where $\mathbf{x}'_j$ denotes an example from the source dataset, y'_j the corresponding label, and α'_j the

weight that was assigned to example $\mathbf{x}'_j$ with respect to the solution of the previously trained SVM:

$$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n}\alpha_i(B(y_i\mathbf{x}_i^T(\sum_{j=1}^{m}\alpha'_j y'_j\mathbf{x}'_j)) - 1)$$

By rearranging the right term, we obtain:

$$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n}\alpha_i(B(\sum_{j=1}^{m}\alpha'_j y_i y'_j\mathbf{x}_i^T\mathbf{x}'_j) - 1)$$

Use kernel trick:

$$L = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j \underbrace{y_i y_j k(\mathbf{x}_i,\mathbf{x}_j)}_{Q_{ij}} - \sum_{i=1}^{n}\alpha_i\underbrace{((B(\sum_{j=1}^{m}\alpha'_j y_i y'_j k(\mathbf{x}_i,\mathbf{x}'_j)) - 1)}_{p_i} \tag{3.12}$$

We obtain the final optimization problem:

$$\begin{aligned}\max_{\alpha} & -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j k(\mathbf{x}_i,\mathbf{x}_j) - \sum_{i=1}^{n}\alpha_i((B(\sum_{j=1}^{m}\alpha'_j y_i y'_j k(\mathbf{x}_i,\mathbf{x}'_j)) - 1)\\ & \text{s.t. } 0 \le \alpha_i \le C\ \forall i \in [1,n]\\ & \alpha^T\mathbf{y} = 0\end{aligned} \tag{3.13}$$

This directly corresponds to a QP in standard form:

$$\begin{aligned}\min_{\alpha} & \alpha^T\mathbf{Q}\alpha + \alpha^T\mathbf{p}\\ & \text{s.t. } 0 \le \alpha_i \le C\ \forall i \in [1,n]\\ & \alpha^T\mathbf{y} = 0\end{aligned} \tag{3.14}$$

where $p_i = B(\sum_{j=1}^{m}\alpha'_j y_i y'_j k(\mathbf{x}_i,\mathbf{x}'_j)) - 1$. The above formulation can be readily plugged into an SVM-solver which allows an efficient computation. For implementational details see Section 4.1. Note that the above formulation is closely related to the regular SVM, where $p_i = -1\ \forall i \in [1,n]$.

3.3.5 Multi-task Learning

The field of multi-task learning is clearly related to domain adaptation. The difference is that in multi-task learning it is assumed that there is a single distribution $P(X)$, while there might be several output variables $Y_1, \ldots, Y_M$, where M denotes the number of tasks.

However, domain adaptation can be treated as a special case of multi-task learning, as two domains correspond to two tasks (S, T), and it is assumed that the class label sets Y_S and Y_T for these two domains are identical.

Dual-task Learning (*DualTask*)

This method is clearly motivated by the *Regtrick*, which is presented in Section 3.3.4. In the formulation of the *Regtrick*, the solution of a previously solved optimization problem $\mathbf{w}_S$ is held fixed, while we optimize the normal vector $\mathbf{w}_T$ of the current problem. If we allow the simultaneous optimization of both normal vectors $\mathbf{w}_S$ and $\mathbf{w}_T$, while maintaining a trade-off between a large margin of the individual solutions (i.e. minimize $\|\mathbf{w}_S\|^2 + \|\mathbf{w}_T\|^2$) and enforcing similarity between the solutions (i.e. minimize $-\langle \mathbf{w}_S, \mathbf{w}_T \rangle$) we get the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}_S, \mathbf{w}_T, \xi} \quad & \frac{1}{2}\|\mathbf{w}_S\|^2 + \frac{1}{2}\|\mathbf{w}_T\|^2 - B\langle \mathbf{w}_S, \mathbf{w}_T \rangle + C \sum_{i=1}^{m+n} \xi_i \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}_S, \Phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \qquad \forall i \in 1, \ldots, m \\ & y_i(\langle \mathbf{w}_T, \Phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i \qquad \forall i \in m+1, \ldots, m+n \\ & \xi_i \geq 0 \qquad \forall i \in 1, \ldots, m+n \end{aligned} \tag{3.15}$$

Again, note that now $\mathbf{w}_S$ *and* $\mathbf{w}_T$ are optimized. Also, if $B = 1$, then the regularization term becomes $\frac{1}{2}\|\mathbf{w}_S - \mathbf{w}_T\|^2$.

Feature Augmentation (*Augmented*)

In (Dau07) a method was proposed that augments the features of source and target examples in a domain-specific way:

$$\begin{aligned} \Phi(\mathbf{x}_i) &= (\Phi(\mathbf{x}_i), \Phi(\mathbf{x}_i), \mathbf{0})^\top \quad \text{for } i = 1, \ldots, m \\ \Phi(\mathbf{x}_i) &= (\Phi(\mathbf{x}_i), \mathbf{0}, \Phi(\mathbf{x}_i))^\top \quad \text{for } i = m+1, \ldots, m+n. \end{aligned}$$

The intuition behind this idea is that there exists one set of parameters that models the properties common to both sets and two additional sets of parameters that model the specifics of the two domains.

It can be shown that the kernel for the augmented feature space can be computed as

$$k_{AUG}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 2\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle & \text{if } [\![i \leq m]\!] = [\![j \leq m]\!] \\ \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle & \text{otherwise} \end{cases}$$

This means that the "similarity" between two examples is twice as high if the examples were drawn from the same domain as if they were drawn from different domains.

Other than in (Dau07), we parametrized the kernel using a parameter B

$$k_{AUG}(\mathbf{x}_i, \mathbf{x}_i) = \begin{cases} B\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle & \text{if } [\![i \leq m]\!] = [\![j \leq m]\!] \\ \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle & \text{otherwise} \end{cases}$$

which allows for a trade-off between the general component and the domain-specific component. If B is large, domain specific components will dominate the solution.

3.3.6 Ensemble Methods

While the methods presented in the other sections return single classification models, we will lastly turn to methods that combine several classification models into a complex classifier. In particular *mixture of experts* methods are considered where the final classification arises from a weighted vote of underlying classification models.

Convex Combination (*Convex*)

A straight-forward idea for domain adaptation is to reuse the two optimal functions f_S and f_T as generated by the base line methods *SourceOnly* and *TargetOnly* and combine them in a convex manner:

$$f_C(\mathbf{x}) = \alpha f_S(\mathbf{x}) + (1 - \alpha) f_T(\mathbf{x}).$$

Here, $\alpha \in [0, 1]$ is the convex combination parameter that can be tuned in a model selection step. A great benefit of this approach is the fact that the classifier does not have to be retrained when performing model selection, which results in great efficiency.

Advanced Convex Combination (*AdvancedConvex*)

Similar to the convex of combination of *SourceOnly* and *TargetOnly*, this method also performs a convex combination of two previously obtained predictors.

$$f_{AC}(x) = \alpha f_{SU}(x) + (1 - \alpha) f_T(x)$$

First, optimal functions f_{SU} and f_T following the *SimpleUnion* and the *TargetOnly* scheme are obtained, whereas only a one-dimensional hyperparameter optimization has to be performed for each classifier. Then at linear cost (with respect to the number of evaluation examples), the optimal combination coefficient α can be obtained as it does not involve the retraining of the SVM, but can be found by simply taking the maximum over all candidates.

The most important difference to *Convex* is that *AdvancedConvex* combines a method (*SimpleUnion*) that is trained on the union of both datasets and therefore takes interactions (in the sense of comparisons) between all examples into account with a method that is specific to the target domain (*TargetOnly*). That way, *AdvancedConvex* has a general component f_{SU}

and a target specific component f_T, and therefore approximates *DualTask*, which involves a two-dimensional hyperparameter optimization (B,C). To summarize, the main idea of this approach is to perform two separate one-dimensional hyperparameter optimizations and a cheap convex combination to obtain an approximation of a more expensive method that involves searching a two dimensional grid in model selection (*DualTask*).

3.3.7 Combination of Several Sources

Most of the above algorithms can be extended in one way or another to integrate several source domains (see Figure 3.3).

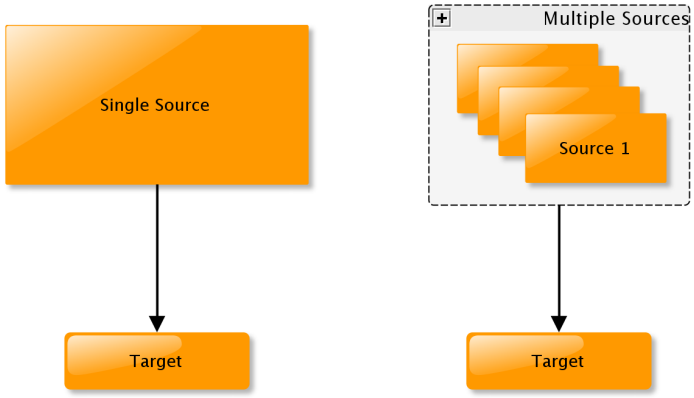


Figure 3.3: Single source domain versus multiple source domain. Left: Data is drawn from one source only. Right: Source dataset is assembled from multiple heterogeneous sources.

In this work we consider only three possible algorithms: (a) convex combinations of several domains, (b) *Combined* on several domains and (c) an extension of the *DualTask* approach to *MultiTask*. We briefly describe these methods below:

Multiple Convex Combinations (*MultiConvex*) The most general version would be to optimize all convex combination coefficients independently. If done in a grid-search-like manner, it becomes prohibitive for more than about three source domains. In principle, one can optimize these coefficients also by solving a linear program. In preliminary experiments we tried both approaches and they typically did not lead to better results than the following combination:

$$f_{MC}(\mathbf{x}) = \alpha f_T(\mathbf{x}) + (1-\alpha)\frac{1}{|\mathcal{S}|}\sum_{S\in\mathcal{S}} f_S(\mathbf{x}),$$

where $\mathcal{S}$ is the set of all considered source domains. We therefore only considered this way of combining the predictions.

Multiple Source Combined (*MultiCombined*) This method is a simple extension to *Combined* (see 3.3.2). Instead of solely considering one source domain, we assemble the source

dataset from several domains. As in *Combined*, a trade-off between source and target domains is adjusted via the cost constants C_T and C_S. Here, C_S sets the weight for all source domains equally.

Multi-task Learning (*MultiTask*) We consider the following version of multi-task learning, which is a straight forward generalization of *DualTask*, which was presented earlier:

$$\min_{\{\mathbf{w}_D\}_{D\in\mathcal{D}},\xi} \quad \frac{1}{2}\sum_{D_1\in\mathcal{D}}\sum_{D_2\in\mathcal{D}}\gamma_{D_1,D_2}\|\mathbf{w}_{D_1}-\mathbf{w}_{D_2}\|^2+\sum_i \xi_i \tag{3.16}$$

$$\text{s.t.} \quad y_i(\langle \mathbf{w}_{D_j}, \Phi(\mathbf{x}_i)\rangle + b) \geq 1-\xi_i \tag{3.17}$$

$$\xi_i \geq 0$$

for all examples $(\mathbf{x}_i, y_i)$ in domain $D_1 \in \mathcal{D}$ and $D_2 \in \mathcal{D}$, where $\mathcal{D}$ is the set of all considered domains. γ is a set of regularization parameters which we parametrized by two parameters C_S and C_T in the following way: $\gamma_{D_1,D_2} = C_S$ if D_1 and D_2 are source domains and C_T otherwise.

3.4 Method Summary and Contribution

So far, we have discussed a number of domain adaptation methods and categorized them into several groups to aid the understanding of their differences and similarities. The following section is intended to provide a high-level summary of the methods that were presented in this chapter.

In Table 3.1, the most important properties of the presented methods are summarized. In the first column, we state to which *category* the respective method belongs. In a nutshell, baseline methods involve no domain adaptation, instance weighting methods perform adaptation by reweighting examples, multi-task methods learn solutions for several domains simultaneously, prior methods exploit prior information about the solution, ensemble methods build a complex classifier from a set of single classifiers and representation based methods attempt to find a representation of examples such that source and target distributions become similar.

The next column in Table 3.1 refers to the size of the kernel matrix (KM) that is involved in training. The methods with the largest KM size are ones from the group multi-task and the method *Combined* which jointly optimize weight vectors for source w_S and target w_T. In the following, these methods will be referred to as *one-step* methods. Methods in this group are generally computationally more demanding than the other methods, as not only dependencies within source and target data sets have to be considered, but also dependencies between source and target data sets. As illustrated in Figure 3.4, this leads to a kernel matrix of size $(n+m)^2$.

The other extreme regarding this criterion are ensemble methods (e.g. *Convex*) that combine domain specific classifiers and which will be referred to as *two-step* methods. Here, each classifier only considers within-domain interactions, leading to total of n^2+m^2 interactions. A compromise between both extrema is the method *Regtrick*. Here, one takes into account

Method	Category	Size KM	Trade-off	Parameters	Unsup
SourceOnly	baseline	n^2	n/a	C	yes
TargetOnly	baseline	m^2	n/a	C	no
SimpleUnion	baseline	$(m+n)^2$	n/a	C	no
Combined	weighting	$(m+n)^2$	Loss	C_S, C_T	no
Pruning	weighting	m^2	Threshold	T	no
KMM	weighting	$n \cdot m$	Reweighting Factor	B	yes
Shift	representation	n/a	Translation Factor	γ	yes
KmerReweighting	representation	$n^2 + m^2$	Threshold	T	no
Regtrick	prior	$n^2 + m^2 + n \cdot sv(m)$	Regularizer	B, C_S, C_T	no
DualTask	multi-task	$(m+n)^2$	Regularizer	B, C	no
Augmented	multi-task	$(m+n)^2$	Regularizer	B, C	no
Convex	ensemble	$n^2 + m^2$	Mixture Factor	α	no
AdvancedConvex	ensemble	$n^2 + m^2$	Mixture Factor	α	no

Table 3.1: Summary of method properties. The column *Category* denotes the group to which a method belongs. *Size KM* refers to the size of the kernel matrix, or in other words, how many examples have to be compared, where m is the number of source examples, n is the number of target examples, and $sv(m)$ is the number of support vectors from source. Column *Trade-off* summarizes how a trade-off between source and target domains is introduced into the learning algorithm. Which hyperparameters are available for each method is displayed in column *Parameters*. Lastly, column *Unsup* shows if methods are suited for the unsupervised domain adaptation scenario.

the intra-domain interactions on target, and the interactions between target examples and the *support vectors* of a previously trained source classifier, leading to a number of interactions of $m^2 + n^2 + n \cdot sv(m)$. Note however, that the interactions between support vectors and target examples lead to a constant term p in the corresponding quadratic program (see Equation 3.3.4), which means that training will effectively have the same time complexity as simply training *SourceOnly* and *TargetOnly* separately. In instance pruning (*Pruning*), an SVM is trained on the target examples, leading to a kernel matrix of size m^2. The solution of the quadratic program to obtain a weight vector β in *KMM* takes into account the similarities between source and target examples, leading to $m \cdot n$ interactions. *Shift* on the other hand simply computes the means for each domain and performs a translation of source examples. While *Shift* itself does not involve computing a kernel matrix, successive training steps on the translated examples might. Finally, in order to perform *KmerReweighting*, an SVM on target and an SVM on source have to be trained as an intermediate step to computing POIMs, resulting in $m^2 + n^2$ interactions.

The column *trade-off* summarizes how the trade-off between source and target domains is incorporated into the respective domain adaptation approach. Let us start with methods that can be formulated similarly to the SVM (i.e. methods from groups multi-task and prior). As discussed in Section 2.3.2, either the loss term or the regularizer can be used to set the trade-off. Multi-task methods employ the latter approach and set the trade-off between domains (or the trade-off between general and domain specific components, to be precise) via

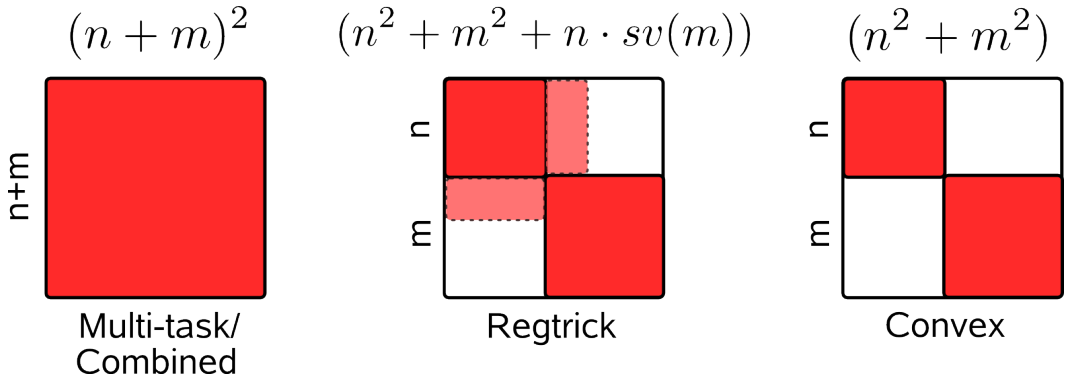


Figure 3.4: Illustrates the number of interactions (in the sense of comparisons) for different classes of methods. Methods in the first category take interactions within domains, as well as interaction between domains into account. This leads to a full kernel matrix of size $(m+n)^2$. On the other extreme are ensemble methods, which combine classifiers that were trained on each domain separately, leading to a kernel matrix of size n^2+m^2. In between is the *Regtrick* that considers within domain interactions, as well as interactions between target examples and the support vectors from source (as can be seen from the dual in Section 3.3.4), leading to $m^2+n^2+n\cdot sv(m)$ interactions., where $sv(m)$ denotes the number of support vectors of the source solution.

the regularizer. The same applies to the *Regtrick*. Only in *Combined* the trade-off is set via the loss-term by using domain specific cost values C_S and C_T. The remaining methods are more diverse regarding trade-off incorporation. *KMM* and *Shift* allow to set how strongly examples from the source dataset are reweighted/shifted. Similarly, a threshold determines which instances are removed in *Pruning* or which features are assigned a zero weight in *KmerReweighting*, respectively. Lastly, a mixture coefficient determines the influence of each expert of *Convex* and *AdvancedConvex*.

Which hyperparameters are used by each method depends on how the trade-off is incorporated, as shown in column *Parameters*. Baseline methods only have one hyper-parameter C that determines the hardness of the SVM margin. Also, methods *Pruning*, *KMM*, *Shift*, and *KmerReweighting* each use only one hyperparameter. *Combined* sets the relative importance of each domain, via domain-specific cost terms, resulting in two hyperparameters C_S and C_T. The multi-task methods use a generic cost term C and an additional trade-off term B. In Table 3.1, *Regtrick* is listed with three hyper-parameters: the trade-off parameter B, the cost terms for the previous solution on source examples C_S, and the cost term for the target examples C_T. Lastly, convex combinations depend on a trade-off term α.

3.4.1 Contribution

Because methods from literature are difficult to separate from own contributions in this chapter, we conclude with a summary of the work that was done. The main methodical contribution was the efficient formulation of the *Regtrick* for which the dual was derived and

the resulting quadratic program was integrated into the SVM-solver *LibSVM*. Some of the methods taken from literature had to be modified in order to be applied to the scenario at hand. For instance, performing class-wise *KMM* constitutes a methodical extension that allowed us to exploit label information.

The *Shift* method, which can be regarded as another flavor of *KMM*, arose in discussions with Gunnar Rätsch, Gabriele Schweikert, and Bernhard Schölkopf and was added to the list of candidate methods. Furthermore, the ensemble method *Advanced Convex Combination* constitutes another contribution which was motivated by some of the multi-task methods (see Section 3.3.5) and by the observation that ensemble methods perform surprisingly well. Finally, the effort relating the wealth of methods and pinpointing their similarities and differences constitutes a large aspect of the work devoted to this chapter.

4 Implementation

The aim of this chapter is to provide an overview of software that was written in the course of this work and to explain the design choices that were made. In the first section, the implementation of domain adaptation methods will be explained, then *pythongrid*, a grid computing tool which was developed to perform experiments in a cluster computing environment will be presented, and lastly modifications to the *shogun* library that allow convenient data persistence, will be outlined.

In scientific computing one commonly uses chimeras of compiled languages such as C or C++ for speed-critical code and scripting languages such as Python or Ruby for high-level programming. That way, one can combine the best of both worlds: the speed of native programs with the flexibility and rapid development of scripting languages.

Following that paradigm, most of the software for this project was written in Python, but was built upon the *shogun* C++ library (SRSS06), which provides implementations of the Weighted Degree Kernel as well as interfaces to various SVM solvers, including *SVMLight* (Joa00) and *LibSVM* (CL01). The *shogun* library can be used from scripting languages, as bindings are available through SWIG (Simplified Wrapper and Interface Generator) (Bea96), which allows an automated generation of wrappers for C++ programs. High-level programming was mostly performed in Python, a modern scripting language with a wealth of libraries for scientific computing (JOP+01; PG07; Hun07). As large parts of the computational infrastructure at the FML are available only in MATLAB, some prototypes as well as a few domain adaptation methods in the final version were implemented in MATLAB.

4.1 Implementation of Domain Adaptation Methods

The implementation of most domain adaptation methods was realized on two layers: the extension of *shogun* in C++ and the design of high-level code in Python. On the Python side, code common to all domain adaptation methods was implemented in a base class called *DomainAdaptationMethod*. Each domain adaptation method was implemented in its own class, each subclassing *DomainAdaptationMethod*, as shown in Figure 4.1.

Note that the methods differ in their training procedure, while most can be evaluated using the same code. Therefore *train* in *DomainAdaptationMethod* solely provides an interface that has to be implemented for each domain adaptation method. To minimize code duplication, the evaluation procedure was split up in *_predict* and *_assess*. That way, *Augmented*, which needs additional domain information during evaluation was implemented by simply overriding the method *_predict*, while the rest of the evaluation code is inherited from the base class and is shared between all methods.

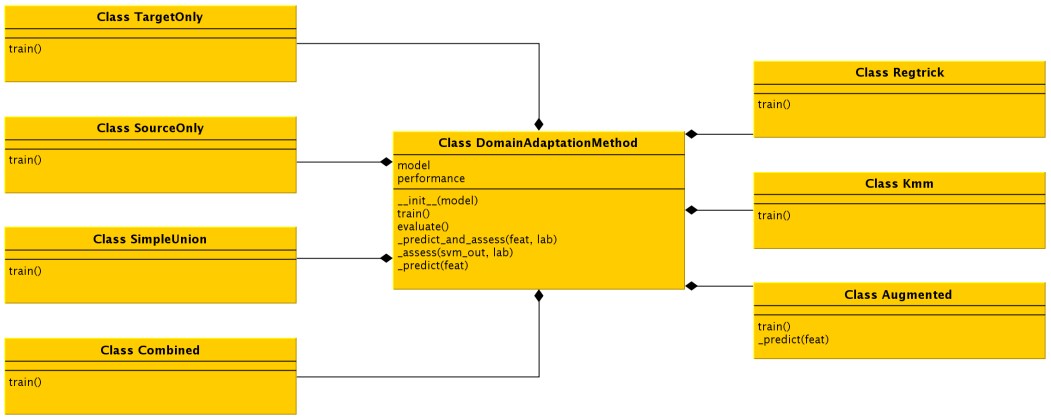


Figure 4.1: Simplified class hierarchy of domain adaptation methods in the Python layer. Common functionality is provided by the base class. Other method classes inherit this functionality and specialize by overriding respective methods (e.g. *train*)
.

Baseline Methods The baseline methods *SourceOnly*, *TargetOnly* and *SimpleUnion* were straight forward to implement, as they do not require a modification of the underlying machine learning toolkit *Shogun*.

Regtrick Having shown that the dual can be expressed as a quadratic program in standard form, the optimization problem was now added to the *shogun* toolbox[1]. A first prototype was developed in Python and was soon replaced by an efficient implementation in C++. For this, the *LibSVM* (CL01) implementation was generalized such that the *Regtrick* as well as the regular SVM could be passed to the same underlying solver. A new SVM class called *RegtrickSVM* was added to *shogun* by subclassing the *LibSVM* class and overriding training and evaluation methods. SWIG-bindings provide a convenient interface to Python and optionally other scripting languages. As for the high-level implementation, only the method *train* had to be implemented in the derived class.

Combined For this method, the *SVMLight* (Joa00) class in *shogun* was modified. It was extended such that a cost vector $\mathbf{C}$ (containing a weight per example) could be passed instead of one scalar C for all examples. This allows the assignment of cost to examples, depending on the domain to which it belongs. Other slight modifications were necessary to incorporate this new weight vector into the training procedure. As the use of the cost-vector $\mathbf{C}$ is relevant to training only, solely the method *train* had to be overridden, while the remaining methods from the base class could be used.

KMM The training of KMM consists of two steps. First, a quadratic program has to be solved to obtain the weights, which are subsequently used in the training of an SVM. While a first prototype was implemented in MATLAB using the commercial solver CLPEX, the

[1] At the moment in a separate branch, but will be merged into trunk in the near future.

quadratic program was then ported to Python using the freely available *cvxopt* toolbox. The resulting weight vector was then incorporated into the SVM solver by means of the generic cost vector **C** (see above), that has been added. As KMM re-weights source samples, it is therefore relevant to training only.

Augmented As has been shown in Section 3.3.5, it is possible to implement this method by using a domain-aware kernel. Hence, such a kernel was added to the *Shogun* toolbox. When computing similarity between examples, not only features are taken into account but also information to which domains the examples belong. The Weighted Degree Kernel was altered to compute weighted similarities $k(i, d_i, j, d_j) = w_{d_i,d_j} k(i, j)$, by extending it such that a weight matrix W is set on construction. Once this was implemented in C++, SWIG bindings were created to make the enhanced functionality available to the Python layer.

Dual/Multi-Task *Dual/MultiTask* was implemented in MATLAB, as it does not rely on *shogun* for training, but on another general purpose solver[2] that was not available in Python. The optimization problem as stated in Section 3.3.5, was approximated by replacing the non-differentiable Hingle loss with the logistic loss, such that it could be solved using a conjugate gradient procedure (enhanced version of gradient descent). As the name implies, gradient decent methods rely on gradient information, which is not available if the objective function is non-differentiable.

Shift Similarly, the prototype of *Shift* was implemented in MATLAB, as the algorithm involves explicit computations in feature space (see Section 2.2.2), and parts of the respective code already existed in MATLAB. Because of its mediocre performance we decided against developing an efficient implementation and retained the prototype.

KmerReweighting The last method to be implemented was *KmerReweighting*. For the computation of POIMs, a library called *easysvm* was used. Furthermore, routines to compute feature weights from POIMs had to be implemented. Once brought into the right format, the resulting weights can be set directly using the standard Weighted Degree Kernel in *shogun*. Unfortunately, only a prototype of this method was implemented - further development of this approach can be considered future research.

4.2 Evaluation Pipleline

After having outlined the implementation of domain adaptation methods, the pipleline for performance evaluation (see Figure 4.2) will be briefly explained. The objective of the pipleline is to first perform a model selection on the training set and then measure the performance of the best model on a test set.

[2]The conjugate gradient solver *minimize* was provided by Carl Rasmussen.

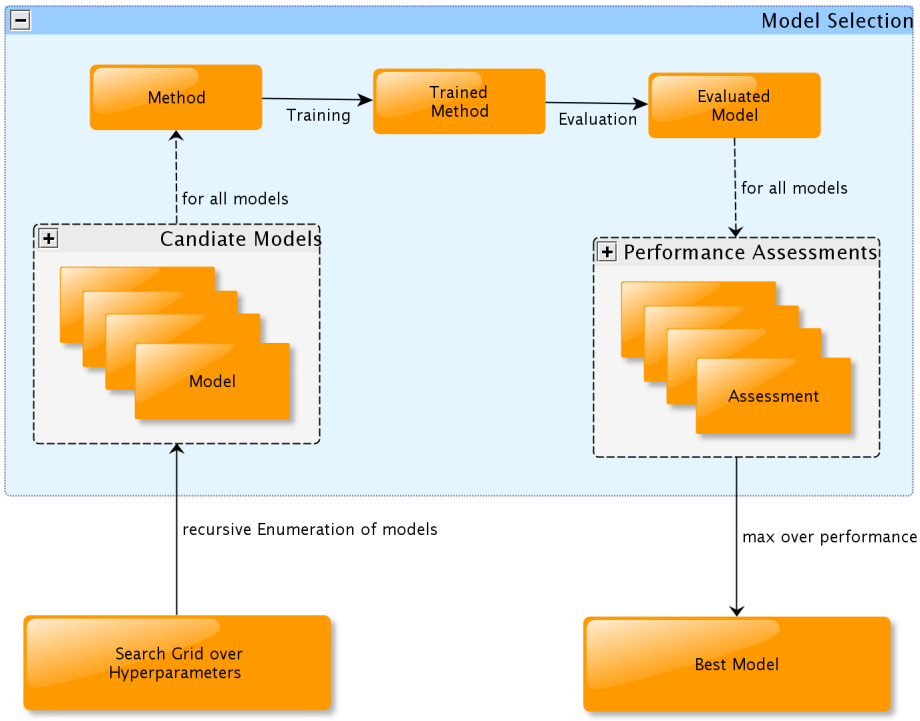


Figure 4.2: Illustration of the model selection pipleline. As input one defines a number of candidate values for each hyperparameter. According to these values, candidate models are recursively enumerated. Each model is benchmarked by training and evaluation. From the list of assessed models, the one with the best performance is selected.

Method Assessment As summarized in Section 3.4, each method has a number of hyper-parameters, which have to be optimized. This is usually done by performing a grid search over all combinations of parameter values. As a first step a set of candidate values for each parameter has to be defined. In our implementation a class called *ModelIterator* then recursively enumerates all candidate models that have to be evaluated. The evaluation of models is performed by a module *CrossValidation* which allows two modes of operation: Performance of a model can be either assessed by using n-fold cross-validation or a simple validation (where dataset partitions are not permuted). In both cases a model is first trained on the training set before performance is assessed on the evaluation set. Thus, for each combination of parameters, we obtain a performance value (e.g. auROC, auPRC, see Section 2.4.2). One can now simply choose the best performing model according to the assessed performance. Using the same routines with slightly different flags, one can re-train the best performing model on the union of training and evaluation set, and assess the final performance on the test set, which was not used for hyper-parameter optimization. This final score is what we are ultimately interested in, as it tells us how well our model generalizes.

Convenient Data Handling To avoid having to re-implement data loading functionality and data subset selection for each of the domain adaptation methods individually, a generic data

handling subsystem was created, as shown in Figure 4.3. It is organized in two layers, a back-end and a front-end. The back-end takes care of the data loading and subset selection. The program in its current state saves and loads data using Python's serialization module *pickle*. It is however straight forward to add back-ends to additional data formats, such as flat-files, XML, and databases.

A central feature of a *DataHandler* is that data is only loaded on demand. This means that in a cluster computing environment, the data handler carries all the meta-data of a dataset, such as size and location of the data. The data handler object itself is light-weight and can be passed onto cluster nodes conveniently (data exchange is handled via a shared file system). Once the actual dataset is requested, it is loaded dynamically.

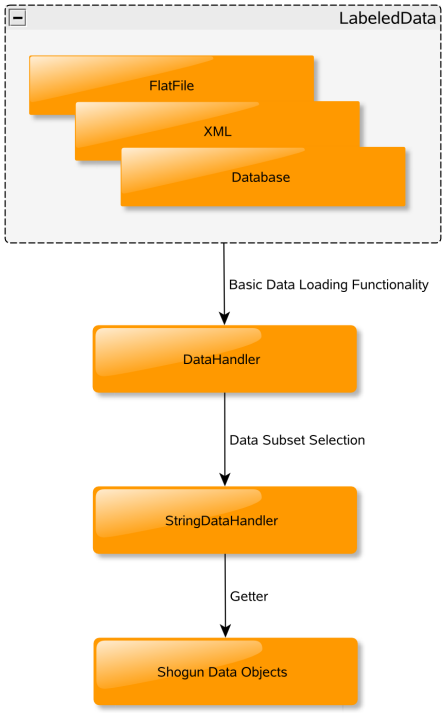


Figure 4.3: Illustration of the data handling sub-system. The back-end (LabeledData) is responsible for data persistence (save/load). Different back-ends could be used interchangeably. Another central feature of the back-end is data subset selection. The class StringDataHandler serves as an adapter to *Shogun* by providing accessors that return *Shogun* data objects.

Another important feature of the back-end is subset selection. For cross-validation, datasets have to be split up into several partitions. For unbiased evaluation it is very important that none of the examples used for training reoccur in the test set. The DataHandler class takes care of dataset splitting on index-basis and provides convenient *getter* methods to fetch training and evaluation data.

While the back-end is agnostic to the type of the data, the front-end *StringDataHandler* provides an adapter to the type of data that is required. The front-end allows to generate

required *Shogun* data objects from a DataHandler. By avoiding code duplication, in particular in data subset selection and by providing a unified way of obtaining the final data objects, the chance of observing error-induced artifacts is minimized.

4.3 A Grid Computing Tool: Pythongrid

The computations that were performed in the course of this work sum up to about 1600 days on a state-of-the-art CPU. As this can hardly be realized on a desktop workstation, a grid computer was used to perform our experiments. Due to the lack of grid computing infrastructure for Python, we developed a tool called *pythongrid* that provides high-level cluster computing functionality for the Python programming language. It communicates with the grid engine (in our case Sun Grid Engine) via the DRMAA protocol, which is an established standard for cluster computing. As a convenience feature, local multi-threading can be used via the same interface as an alternative to cluster computing. For debugging purposes, it is sufficient to pass a simple flag in order to run the program sequentially on the host machine.

This piece of software was written employing the back-end-front-end paradigm. *pythongrid* is based on an open-source SWIG-wrapped DRMAA implementation by Sun Micro Systems. As the back-end is exchangeable with relatively little effort, *pythongrid* can in principle be used as a front-end to other grid engines besides the Sun Grid Engine. The front-end is written in Python and provides a convenient method to execute functions on a grid computer.

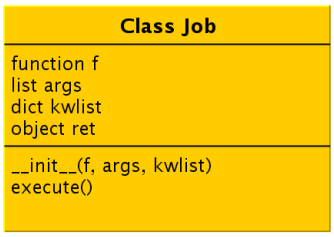


Figure 4.4: Simplified UML scheme of the Job class. On construction, the function *f*, the argument list *args* and the keyword arguments *kwlist* are set. When the method *execute* is invoked, function *f* is executed with the respective arguments *args* and *kwlist*. The return value is written to field *ret*.

The central entity in *pythongrid* is the Job object (see Figure 4.4). A Job object consists of a function[3], its arguments, a method that invokes the execution of the function with its respective arguments, methods for filesystem persistence and some metadata.

Besides the Job class, two important functions are defined in the *pythongrid* module, corresponding to two modes of operation:

- *submit and collect:* Here, a set of jobs is executed on the cluster and the program waits until all jobs have finished, then collects the return values of the executed functions

[3]Note that in Python, functions are treated as objects.

and continues execution. In principle it corresponds to a parallelized for-loop, where execution continues after processing the loop.

- *submit and forget:* In this mode, it is assumed, that each Job object takes care of writing its result to the filesystem or a database, independently. The submission script will terminate, right after the jobs are sent to the queue of the grid engine. In this mode, results are not collected automatically.

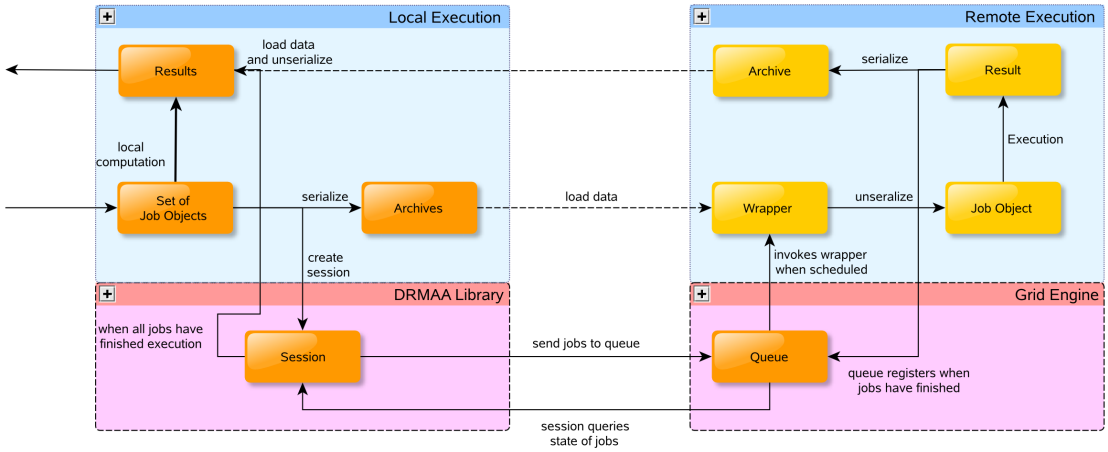


Figure 4.5: Illustration of the cluster computing tool *pythongrid*. A set of *Job* objects is submitted to *pythongrid*. If these jobs are computed locally, results are returned after sequential computation. If the user chooses to submit the jobs to the cluster computer, a new session object is created and *Job* objects are serialized and written to the file system. Each job is submitted to the queue of the grid engine, sending the path to the stored Job file as an argument. When the job is scheduled on the client side, the *Job* object is restored from the file system by a wrapper. The function attached to the *Job* object is executed with the respective arguments and the result is written to the field *ret* in the *Job* object. The *Job* object is written back to the file system and the client terminates, which is registered by the grid engine. Via the DRMAA layer, the Session object can query whether all jobs have terminated. Once that is the case results are restored from the file system and returned, if in *submit and collect* mode. Results can be collected manually, if in *submit and forget* mode.

As shown in Figure 4.5, a list of jobs is passed to one of the submit methods. Depending on which flags are set, the jobs are executed locally (possibly in a multi-threaded manner) or sent to the cluster computer. In case of local computation, one simply iterates over the list of job objects, sequentially invoking their *execute* methods and returning the results, once all job objects are processed. While the local computation is straight-forward, the remote execution demands some explanation.

When one of the submit procedures is invoked, two things happen. First, all jobs are written to the filesystem through their persistence methods (in principle this could also be done via a database). Then, a new session is created via the DRMAA back-end, and each job is submitted to the grid engine queue, with the filename of the persistent object as an argument. When the job is then executed on a cluster node, a wrapper unserializes and restores the persistent *Job* object, which contains the saved function and its arguments. Once the object is restored, the *execute* method is invoked and the return value is written back to the respective field in the *Job* object. Again, using its persistence methods, the evaluated *Job* object is written back to the file system.

When in *submit and collect*-mode, the *pythongrid* back-end keeps querying the grid engine to keep track of whether all jobs have finished. Once this has happened, the local front-end of *pythongrid* collects all processed *Job* objects that have been saved to the file system and restores the results on the host computer. In case the user chose to use the *submit and forget* mode instead, *pythongrid* returns a session identifier that allows the user to collect the results (if desired) at an arbitrary time in the future. This requires that the respective jobs have already finished, which can also be queried manually given the latter session identifier. A function for querying the state of a session manually is provided by *pythongrid*.

The beauty of *pythongrid* is that all functionality is encapsulated and can be used without any detailed knowledge about the underlying implementation. The following is a simple example of how to submit a set of Jobs to the cluster computer.

```
from pythongrid import Job, process_jobs
import scipy

args_list = [[3], [4], [5], [6]]
jobs = []

for args in args_list:
    job = Job(scipy.factorial, args)
    jobs.append(job)

results = process_jobs(jobs)
```

The above example shows the computation of the factorial function on the cluster. Four *Job* objects are created, one for each item in *args_list*, where each object has *scipy.factorial* as function, which is to be executed. Once, the list of *Job* objects is created, the computation on the cluster computer is as simple as a function call. All argument-passing, queuing, and cleaning up temporary files is taken care of, under the hood. *pythongrid* was used throughout this thesis to conduct large scale experiments and has proven to be a suitable tool to perform convenient, high-level cluster computing in Python. It is already being used by other researchers at the FML and will most likely be open-sourced in the near future.

4.4 Serialization

The need for persistent *Shogun* objects became apparent when implementing the domain adaptation method *Regtrick*. Because a RegTrickSVM requires an already trained SVM to be passed as a constructor argument, one would like to be able to save a trained SVM and reuse it later in an automated manner. Therefore, possible solutions to this problem were researched and it was decided to turn to the *boost* C++ Library, which provides a serialization package for C++ programs.

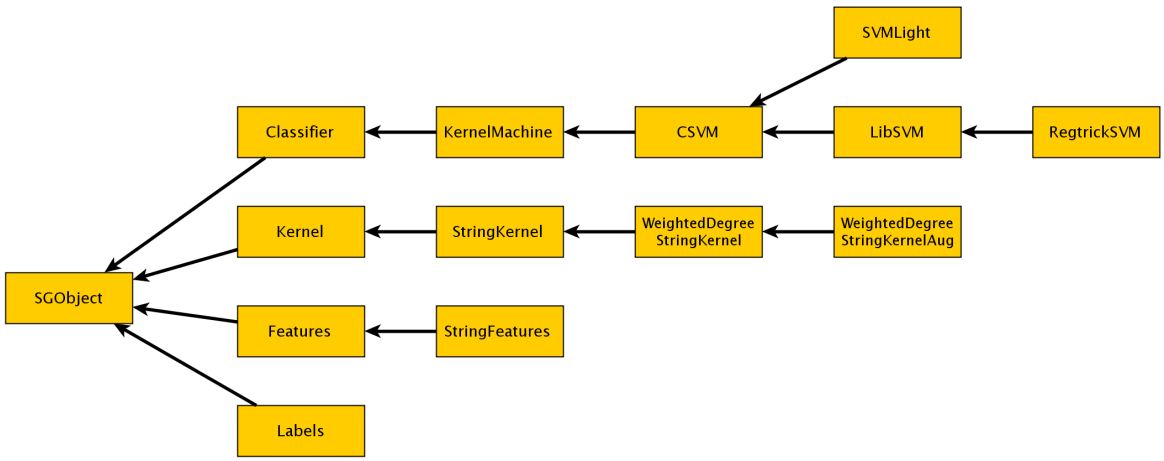


Figure 4.6: Excerpt of the *shogun* class hierarchy. The displayed classes are the ones that are most relevant to this project. Serialization code was added to all of them.

A few lines of serialization code had to be added to each relevant *Shogun* class (see Figure 4.6), such that objects from each class could be saved and restored. As a simple example, the following code was added to the class *KernelMachine*, which has the fields *kernel, use_lin_add, use_batch_computation*.

```
#ifdef HAVE_BOOST_SERIALIZATION
  private:

    friend class boost::serialization::access;
    // When the class Archive corresponds to an output archive, the
    // & operator is defined similar to <<.  Likewise, when the class Archive
    // is a type of input archive the & operator is defined similar to >>.
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version)
    {
      ar & boost::serialization::base_object<CClassifier>(*this);
      ar & kernel;
      ar & use_batch_computation;
      ar & use_linadd;
    }
```

```
#endif //HAVE_BOOST_SERIALIZATION
```

Clearly, the serialization of a simple class is straightforward. Each member variable that is to be serialized should be included in the *serialize* method, using the overloaded & operator. Furthermore, by invoking the serialization method of the base class, fields that are inherited will also be serialized. One can see from the code that *Archive* is a template variable, so different types of archives can be used exchangeably (flat file, binary, XML).

While the serialization of simple classes was straight forward, the following obstacles had to be overcome:

- Serialization across module boundaries (several shared-objects)
- Serialization of pointers to natives (e.g. double*)
- Serialization of classes without default constructor

To be able to perform serialization across module boundaries (e.g. shared objects/dynamic link libraries), the boost library had to be patched to include functionality, which will be part of future boost releases. The problem with serializing C-arrays (i.e. double*), is that boost does not know how many elements the array has. Therefore, the method *serialize* has to be split up into *save* and *load*.

```
#ifdef HAVE_BOOST_SERIALIZATION

    friend class boost::serialization::access;
    template<class Archive>
    void save(Archive & ar, const unsigned int version) const
    {
      ar & boost::serialization::base_object<CSGObject>(*this);
      ar & num_labels;
      for (int i=0; i < num_labels; ++i) ar & labels[i];
    }

    template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
      ar & boost::serialization::base_object<CSGObject>(*this);
      ar & num_labels;
      labels = new double[num_labels];
      for (int i=0; i< num_labels; ++i) ar & labels[i];
     }

    BOOST_SERIALIZATION_SPLIT_MEMBER();
#endif //HAVE_BOOST_SERIALIZATION
```

In the above example, the *Label* class is serialized. Member variables are a C-array of doubles called *labels* and an integer *num_labels* that keeps track of the size of the array. In

save, the first step is to save the number of elements and then each element separately. Later, when *load* is invoked, one first restores the number of elements, then allocates an array of the respective size and subsequently restores the elements of the array. Having to implement serialization this way is quite inconvenient and could be avoided by using containers from the standard template library (STL), such as std::vector.

When unserializing an object, boost first creates an object using the default constructor and then restores the member variables from the archive. For that reason, it constitutes additional effort to serialize classes that do not have a default constructor. To overcome this problem, two additional methods have to be added to these kind of classes: *save_construct_data* and *load_construct_data*, which allow to save data separately, that is required to construct an object. When unserializing an object, these fields are restored first, then an object is created by passing the values to the constructor and lastly the other fields are restored using the regular serialization functionality.

In summary, boost provides a convenient way to serialize C++ libraries. However, we conclude from our experiences of serializing *shogun*, that if one is planning to add serialization support to a C++ program in the future, one should try to avoid using C-arrays and classes without default constructor.

With the serialization support in *shogun* it is now possible to simply save and restore any shogun object (i.e. trained SVMs, computed kernels) conveniently. This functionality has been successfully used for the Python-based experiments performed in this thesis.

5 Application to Splice Site Prediction

In order to conduct a thorough empirical analysis of domain adaptation algorithms for sequence analysis, we selected the problem of splice site prediction. This problem is particularly well suited, as labeled training data is available for numerous organisms and it exhibits several interesting attributes. For one, as splice sites are functional elements, they are relatively well conserved across eukaryotes, which constitutes a fundamental requirement in order to benefit from transferring knowledge from a source organism to a target organism. However, due to evolutionary changes, splice sites and the underlying mechanisms are not identical in the organisms, which gives rise to the need for intelligent adaptation algorithms. Furthermore, splice site detection can be regarded as a representant for the broad class of sequence-based classification problems, which we are ultimately interested in.

5.1 The Problem of Splice Site Prediction

The task of splice site prediction can be formulated as a binary classification problem as follows. We know *a priori* that the vast majority of splice sites contain a certain consensus sequence, namely GT on the donor side, and AG on the acceptor side of an intron. As almost *all* splice sites contain this consensus, we can use this information to obtain an exhaustive list of candidates in a straight forward manner. Given the sequence of a genome, we consider all occurrences of the above consensus sequences and extract a window of a reasonable size, for instance 150 bp (base pairs), around the consensus position (see Figure 5.2). It is assumed that sequence features, which are relevant to the splicing mechanism, are likely to be located in the proximity of the splice site. While this may not hold true for each and every sequence feature involved in the splicing process, a window of the above size is sufficient to achieve a good predictive performance.

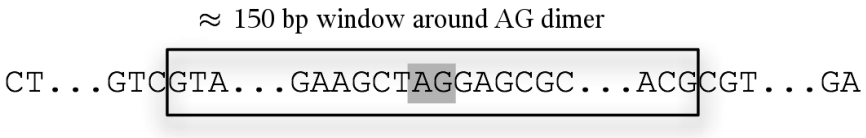


Figure 5.1: In splice site prediction, windows of a reasonable size (here 150 bp) are considered around known consensus sites. This figure shows the consensus AG of a potential acceptor site. Figure created by Gunnar Rätsch.

The problem of splice site prediction can therefore be formulated as a binary classification problem, in which one attempts to categorize the set of candidate sequences into true splice

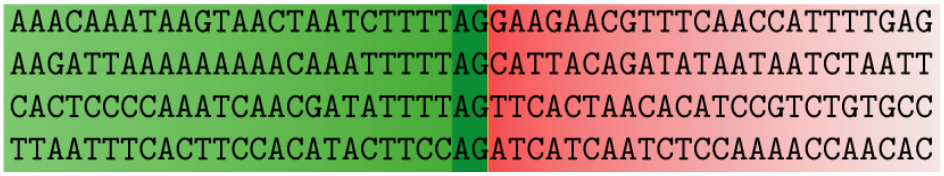


Figure 5.2: Windows around all occurrences of the respective consensus are extracted. Displayed in this figure is the consensus of an acceptor site, whereas the positions in green denote intron positions and the positions in red denote exon positions. Figure created by Gunnar Rätsch.

sites and decoy sites (sequences which exhibit the consensus, but are not real splice sites).

Unbalanced Problem As DNA sequences are based on an alphabet of cardinality four $\{A, T, G, C\}$, every dimer has a chance of $(1/4)^2 = 1/16$ of containing a consensus of length two. For example, consider the genome of the nematode *C. elegans*, which has approximately 100 million bp. According to the above calculation, we have about 30 million candidates for donors or acceptors, respectively. On the other hand, there are roughly 20000 genes (SAB$^+$06) mostly with 4-5 exons on average, so there are roughly 60000 true splice sites of each type. Thus, for every true splice site we observe about 500 decoy sites.

This simple calculation shows that we are facing an extremely unbalanced classification problem, where much fewer positive examples are available than negative examples. As has been shown before (SSP$^+$07), SVMs are well suited for such a task.

Very High Dimensional Feature Space The Weighted Degree Kernel (see Section 2.2.4) was introduced by (RS04) and has been frequently used in sequence analysis, in particular by Sonnenburg et. al. (SRS05; SSP$^+$07; SZPR08). When using a Weighted Degree Kernel with a reasonable degree and window size (e.g. $l = 20$, $w = 150$), feature space becomes extremely high dimensional:

$$dim = 150 \cdot \sum_{i=1}^{20} 4^i$$

It has been shown that in order to achieve the best classification performance, a large number of training examples is essential ((SSP$^+$07) used up to 8 million examples). One can argue that this motivates the use of domain adaption algorithms even for problems where a reasonable body of data is available, as performance can nearly always be improved by using more data.

5.2 Domain Adaptation Setting

For the problem presented above, we investigated a number of related classification problems, varying the degree of similarity between source and target domains, and sizes of datasets.

Similarity of Source and Target Distributions A question of particular interest is the effect of differing degrees of similarity between our source distribution and the target distribution. Clearly, the closer source and target distributions are, the more we can gain from the information transfer. However, the question remains whether some of the presented methods work better under specific circumstances, while others are better in another setting.

Thus, we selected a number of organisms in varying phylogenetic distances to construct such a scenario. We considered a star topology, where always the same organism was used as a source dataset. The well-studied model organisms *C. elegans* seemed to be a reasonable choice due to the large amount of high-quality training data.

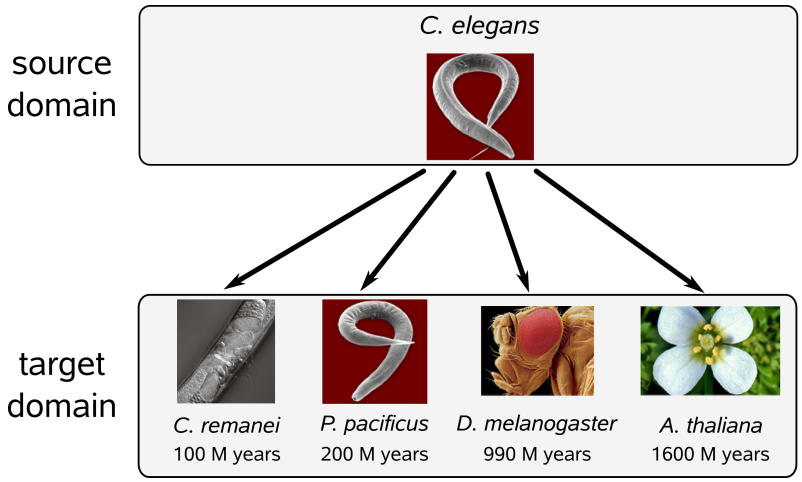


Figure 5.3: This figure shows the organisms we considered in this study. As source domain, we always used the splice site data from the well-studied model organism *C. elegans*. As target domains we considered organisms at different evolutionary distances to *C. elegans*. Pictures of organisms were taken from the website of the MPI for Developmental Biology and the UCSC genome browser.

As target organisms we chose two additional nematodes, namely, the close relative *C. remanei*, which diverged from *C. elegans* 100 million years ago (SBB$^+$03), and the more distantly related *P. pacificus*, a lineage which has diverged from *C. elegans* more than 200 million years ago (PDS04). As a third target organism we used *D. melanogaster*, which is separated from *C. elegans* by 990 million years (UVEB03). Finally, we consider the plant *A. thaliana*, which has diverged from the other organisms more than 1,600 million years ago (HBVS04). It is assumed that a larger evolutionary distance will likely have led to an accumulation of functional differences in the molecular splicing machinery. We therefore expect that the differences of classification functions for recognizing splice sites in these organisms will greater with increasing evolutionary distance.

Various Data Set Sizes Furthermore, we are interested in how different data set sizes of our source and target sample affect domain adaptation. For computational reasons we decided to vary only the size of our target sample, while retaining the size of the source sample. Considering all combinations of evolutionary distance, target sample size and source sample size, including a complete model selection for each experiment would have been too expensive in terms of time and resources. Therefore, we randomly selected a large enough source data set with 100,000 training examples from *C. elegans*, while for all target organisms data set sizes of $2500, 6500, 16000, 40000, 100000$ were considered.

5.3 Comparison of Domains

Knowing the evolutionary distance between organisms in years, we were now interested in the difference of the splice site classification *problems*. For this, we considered a number of measures that take into account how the SVM *"sees"* the data through string kernels.

Thus, the following measures were considered:

- Separability of data from different domains
- Break-even point
- Comparison of POIMs (Positional Oligomer Importance Matrices).

Separability by domain To better understand the distance between these organisms with respect to the specific task of splice site detection, we performed experiments to classify the example sequences according to their origin. For these pairwise discrimination tasks, we used the same method as for the splice site detection task, namely SVMs with the weighted degree kernel ($\ell = 22$). As a performance measure we used the area under the precision-recall curve (auPRC). In this context, a higher auPRC is associated with a greater difference between the functioning of the splice mechanism. We examined the difference between true splice sites, as well as between decoy sites (i.e. other genomic sequences). For each classifier we used $1,000$ examples from each of the two investigated organisms for training (67%) and evaluation (33%). The results are shown in Table 5.1.

We observe that the discrimination performance of discriminating true splice sites (auPRC+) from different organisms correlates well with their evolutionary distance. The discrimination performance for decoy examples (auPRC-) saturates for organisms more than 100-200 million years apart (the rest of the genome is evolving faster than important functional elements such as splice sites).

Break even point The following experiment is in preparation of a larger set of experiments, which includes all organisms, dataset sizes and methods. While we use the same setting, solely the baseline methods, *SourceOnly* and *TargetOnly* are of interest for this experiment.

The question asked here is: How many examples from the target domain are required to

	time (10^6 years)	**auPRC+**	**auPRC-**
C. elegans - C. remanei	> 100	63%	67%
C. elegans - P. pacificus	200	93%	81%
C. elegans - D. melanogaster	990	95%	78%
C. elegans - A. thaliana	$> 1,600$	98%	77%

Table 5.1: Summary of different distance measures between source and target organism. Reported are the times since divergence. Additionally, the performance of classifiers that distinguishes between examples of two organisms. The performance measure is given as auPRC+ for a classifier that distinguished between true splice sites and auPRC- for a classifier that distinguished between decoy sites. This table was published in (SWSR08).

perform equally well (performance is measured on a test set from the target domain) than a large number of examples from the source domain (100,000 in this case)? This point of equal performance of the *SourceOnly* and *TargetOnly* classifier will be referred to as *break-even point*.

Of course, one expects that the more closely related two organisms are, the more target examples are necessary to achieve a superior performance to a large source sample. While the ordering of break-even points can be expected to be identical to the ordering induced by the phylogenetic distance in years, the following experiment can help to find answers to the question of *how different* the problems really are.

Results from this experiment are shown in Figure 5.4. Clearly, the break-even point moves from large target sample size to a small target sample size, as the evolutionary distance increases. We observe that the break-even point is located at $\approx$ 75K for *C. remanei*, 25k for *P. pacificus*, 10k for *D. melanogaster*, and 4k for *A. thaliana*. This measure therefore reveals the same overall picture as encountered when considering the *separability* of data sets (as in Table 5.1), while not converging as quickly (e.g. *P. pacificus* and *D. melanogaster* can be distinguished).

Positional Oligomer Importance Matrices A well-known problem of the SVM is the lack of interpretability. While the use of, say, single-order Markov Chains allows the visualization of learned models in the form of a sequence motif, the respective visualization is not obvious, when using SVMs with string kernels of high degree.

However, (SZPR08) recently proposed an efficient solution to this shortcoming: POIMs. In a nutshell, a POIM assigns each k-mer at each position i an importance score, which is computed by considering the differences between the expectation value, with the k-mer of interest z at position i held fixed, and the unconditioned expectation value.

$$Q(i,k,z) = E[s(X)|x_{i,k} = z] - E[s(X)] \tag{5.1}$$

The rationale behind this is that the above difference will quantify the influence of the respective k-mer.

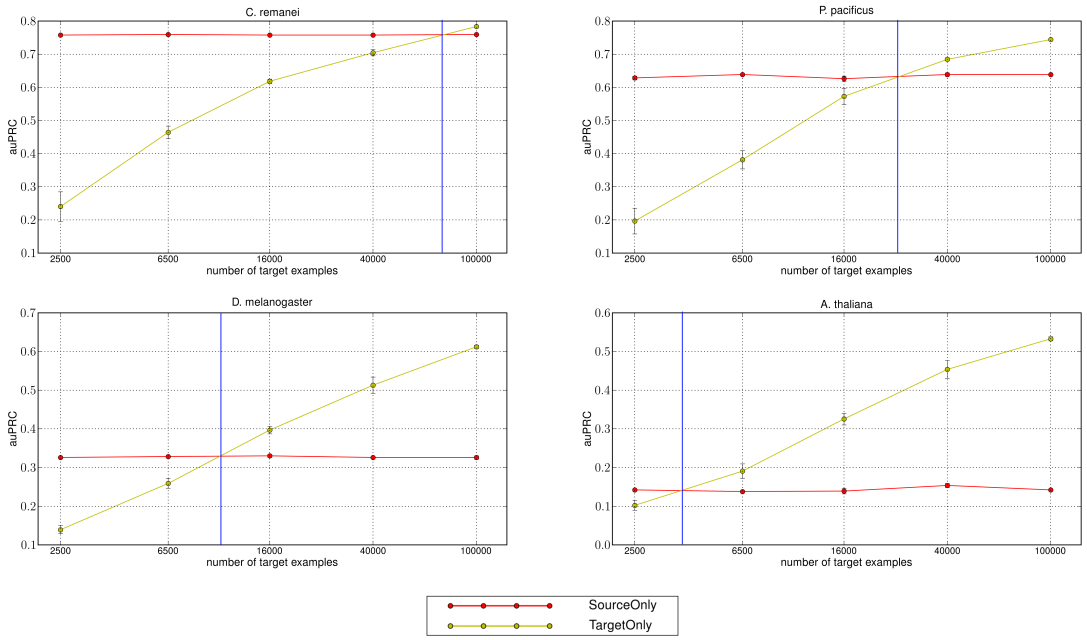


Figure 5.4: The x-axis shows the number of examples from our target domain, the y-axis shows the auPRC (area under precision recall curve), which was measured on a test set from the target domain. The break-even point (illustrated as blue line) shows how many target examples are required in order to perform equally well as a large source sample. The four plots show that this clearly depends on the evolutionary distance of the compared organisms. The closer two organisms are, the more relevant information about the problem in the target domain is contained in the source sample.

Figure 5.5 displays a representation of the POIM of the source organisms *C. elegans* as well as the target organisms. One can clearly see that the patterns are more similar for closely related organisms. Furthermore, we observe that the most important k-mers for classification of splice sites are located next to the intro-exon boundary.

To quantify the similarity between problems, we linearized the POIM matrices Q and assessed the similarity by taking the normalized dot product between *C. elegans* and the target organisms, which corresponds to the following equation.

$$s_{\text{POIM}}(Q_S, Q_T) = \frac{\sum_i \sum_k \sum_z Q_S(i,k,z) \cdot Q_T(i,k.z)}{\sum_i \sum_k \sum_z Q_S(i,k,z) + \sum_i \sum_k \sum_z Q_T(i,k.z)} \tag{5.2}$$

The resulting value is a similarity measure between 0 and 1, whereas a large value denotes a larger similarity. The results are listed in Table 5.2.

Summary In this section we investigated three measures to assess the similarities of domains. For all of the above measures, similarities are measured with respect to functions

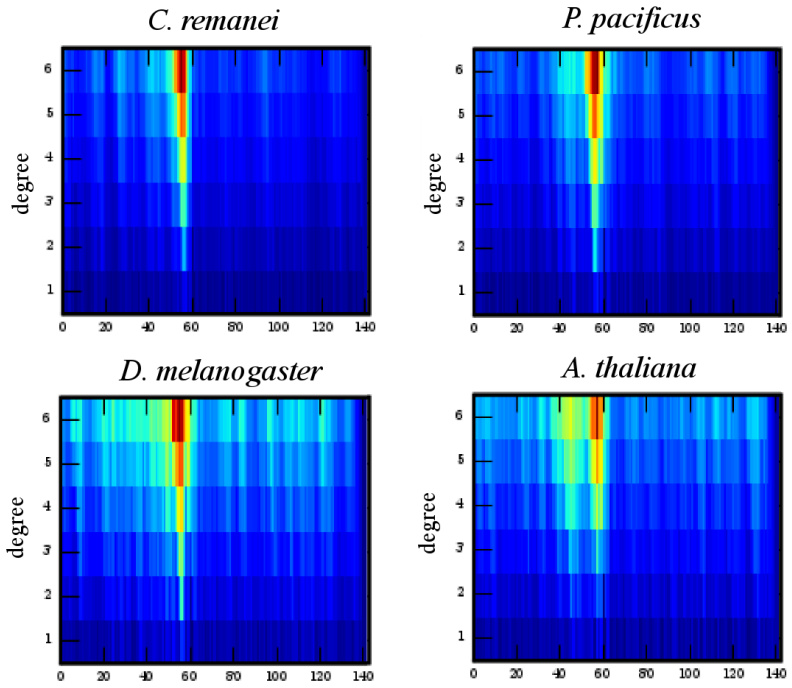


Figure 5.5: POIMs of all target organisms in ascending evolutionary distance. The sequence position i is shown on the x-axis and the degree of k-mers is on the y-axis. The importance of k-mers of degree d at position i is obtained by summing over all respective k-mers z, according to Equation 5.1. High importance is encoded as bright color. While a common pattern can be observed for all organisms, these plots show that the problems do in fact differ to some extend between organisms.

	time (10^6 years)	**POIM similarity**
C. elegans - C. remanei	> 100	0.468
C. elegans - P. pacificus	200	0.459
C. elegans - D. melanogaster	990	0.362
C. elegans - A. thaliana	$> 1,600$	0.296

Table 5.2: Summary of POIM-based distances between source and target organism according to the measure defined in Equation 5.2.

in our hypothesis class, i.e. SVMs using the WD-Kernel. To summarize, the measures presented in this section have three important applications. First, they help to quantify how different domains are and thus aid the understanding of the domain adaptation problem at hand. Second, we will show in the following that domain adaptation methods work most efficiently at the *break-even point*. Lastly, inter-domain distance measures are needed to prove generalization bounds for domain adaptation algorithms (BdBCP07; BMP06).

5.4 Experimental Setup

To simulate the domain adaptation scenario, we randomly selected a dataset of 100,000 examples from *C. elegans*, while data sets of sizes 2,500, 6,000, 16,000, 40,000 and 100,000 were selected for each target organism. It was ensured that the positives to negatives ratio is at 1/100 for all datasets. Two thirds of each target set were used for training, whereas one third was used for evaluation in the course of hyperparameter tuning. Additionally, test sets of 60,000 examples were set aside for each target organism. As some of the methods are explicitly computed in feature space, the number of dimensions had to be limited to a reasonable number. Therefore, we chose to employ WD kernels of degree $l = 1$.

All experiments were repeated three times with different training splits (source and target), except the last one which always used the full data set. Reported will be the average area under the precision-recall-curve (auPRC) and its standard deviation, which is considered a sensible measure for imbalanced classification problems.

5.5 Experimental Results

We considered two different settings for the comparison. For the first set of experiments we assume that there is *one* source domain with sufficient data (100k examples in our case) that should be used to improve the performance in the target domain. In the second setting we analyze whether one can benefit from using several source domains.

Single Source Domain

In Figure 5.7 we compare the methods *Regtrick*, *DualTask*, *Augmented*, *Shift*, *Combined*, *Convex*, and *AdvancedConvex*. For each of these methods, we plot the improvement over the best performing baseline methods *SourceOnly* and *TargetOnly* at a given target data set size. This is done for each of the four target organisms: *C. remanei*, *P. pacificus*, *D. melanogaster*, *A. thaliana*. The absolute performance is shown in Figure 7.1 and the exact numerical values are shown in the appendix. Furthermore, Figure 5.6 shows a summary with a focus on best/worst performing methods.

A first central observation is that all of the compared domain adaptation methods show considerable improvements over baseline methods. The improvements show a clear dependency on target data set size as well as evolutionary distance. For each organism, a *point of maximal gain* can be determined, at which the improvement over baseline methods is maximal. For the closely related nematode *C. remanei*, the maximum is located at a target data set size of 40,000, for *P. pacificus* and *D. melanogaster*, at 16,000 and for the most distantly related organism *A. thaliana* at about 6,500 target examples. The maxima correspond well to the *break-even points* that were determined in Section 5.3. We conclude from this observation, that the application of domain adaptation algorithms improves the performance the most, when performances of *SourceOnly* and *TargetOnly* are similar. This should be kept in mind when facing a novel domain adaptation scenario.

Moreover, we observe that improvements of different magnitude can be achieved depen-

	C. remanei	*P. pacificus*	*D. melanogaster*	*A. thaliana*	Total
DualTask	1.0	3.0	2.2	1.8	2.00
AdvancedConvex	2.4	1.2	5.0	2.8	2.85
Augmented	6.0	4.4	2.4	3.6	4.10
Shift	4.8	4.2	3.8	4.4	4.30
Convex	3.8	3.6	6.2	4.6	4.55
Combined	5.6	5.8	3.4	4.6	4.85
Regtrick	5.2	6.6	5.4	6.4	5.90
SourceOnly	7.4	8.0	8.6	8.8	8.20
TargetOnly	8.8	8.2	8.0	8.0	8.25

Table 5.3: Average rank of methods. This table displays the mean rank over target data set sizes for each organism and method. Additionally, the mean rank over all target data set sizes and organisms is shown in the column *Total*. Methods are ordered by their total rank.

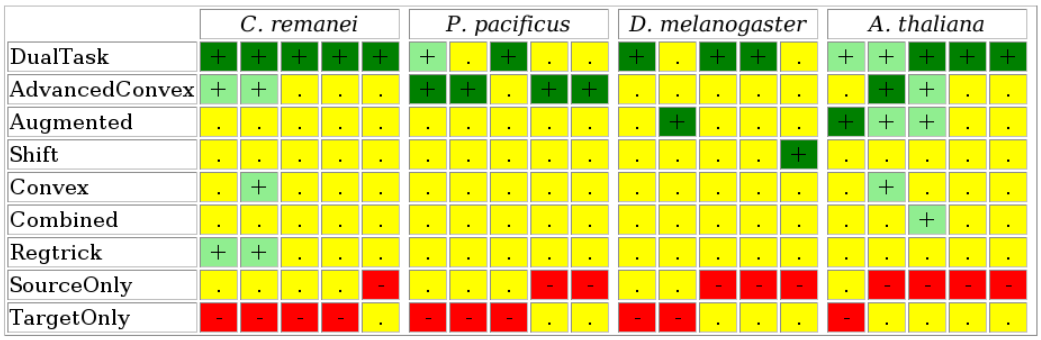


Figure 5.6: Summary of performances of each presented method. Each column contains 5 sub-columns, which correspond to ascending target data set sizes (2,500, 6,500, 16,000, 40,000, 100,000). The method with the highest auPRC score for a given organism and target data set size is assigned a dark green background. Methods that do not perform significantly worse (within a confidence interval of $\sigma/\sqrt{n}$) are shown in light green. Accordingly, the worst performer is shown in red, while methods which perform similarly bad in the sense defined above are assigned an orange color (does not occur in this plot, but in Figure 5.8 that uses the same color scheme). The remaining methods that do not fall in any of the mentioned categories are shown in yellow. This figure summarizes results from the single-source domain experiments.

dent on the distance between domains (i.e. evolutionary distance between organisms). The results indicate that domain adaptation methods show the largest improvement if domains are relatively far apart (note the different scales for the subplots in Figure 5.7). This is because data from the source domain can readily be applied to the target domain, if source and target distributions are close. The further source and target distributions are apart, the more

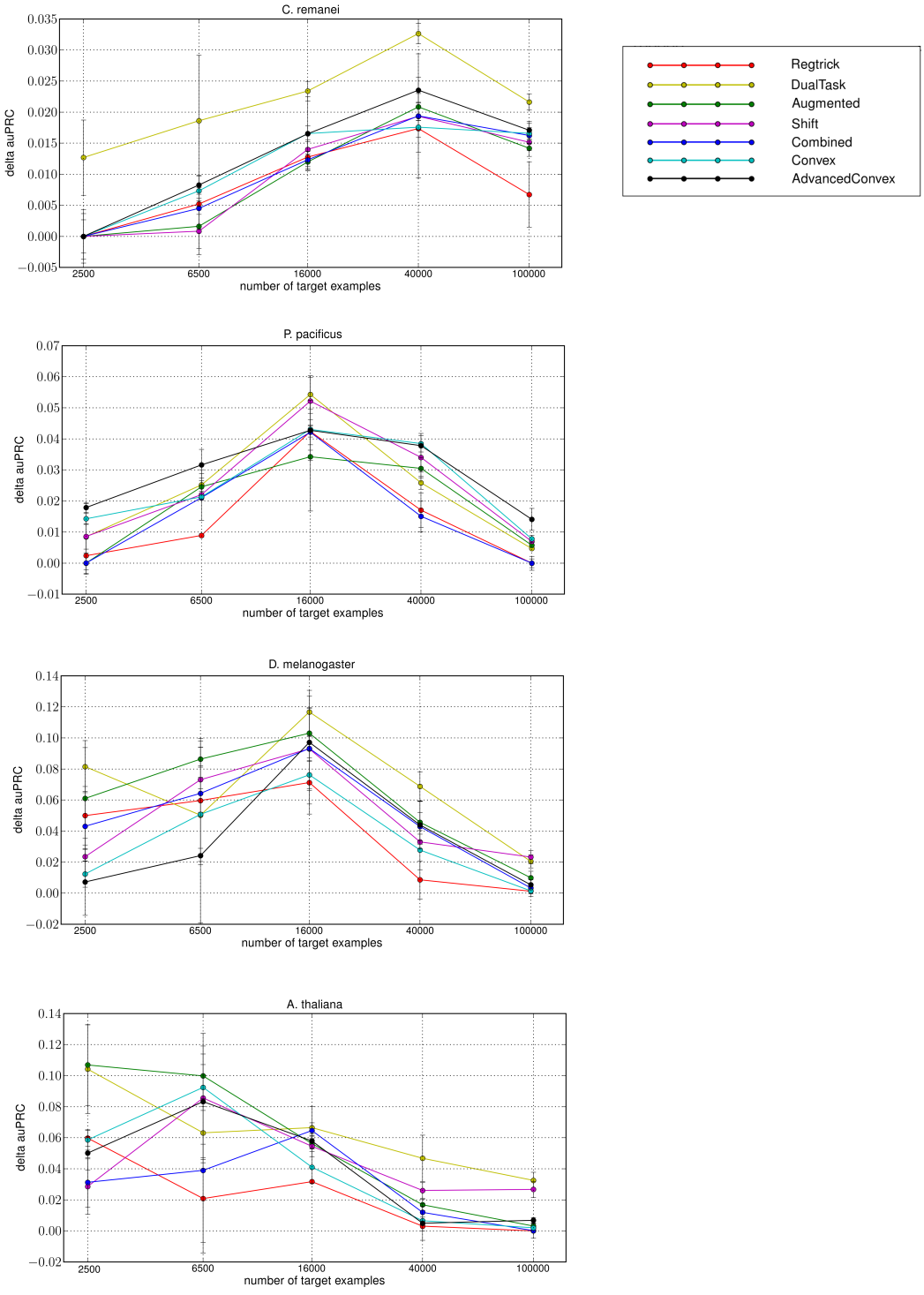


Figure 5.7: Results of single source domain experiments. The plots show the improvement over the best of *TargetOnly* and *SourceOnly* at each target size.

critical it is to perform adaptation of the data. For *C. remanei*, a maximal improvement of 3.2% can be observed, for *P. pacificus* 5.3%, for *D. melanogaster* 11.7%, and for *A. thaliana* 10.6%. Also, note that if close data is available, then it should be used.

We observe that *DualTask* outperforms the other domain adaptation methods in most cases. This is most pronounced on *C. remanei* (i.e. if domains are closest), where *DualTask* has superior performance for all target data set sizes. A maximal improvement of about 2% over the other *non*-baseline domain adaptation methods can be observed on *D. melanogaster* for 16000 and 40000 target examples. From the summary in Figure 5.6, we observe that *DualTask* performs best in 12 of 20 experiments. Moreover, *DualTask* exhibits an average rank of 2.0, which is lower than the average rank of all other baseline methods (see Table 5.3). The methods showing the least average improvement over baseline methods are *Convex*, *Combined*, and *Regtrick*. While *Regtrick* performs worse in most cases, it is able to outperform *Convex* on *D. melanogaster* for low target data set sizes.

Clearly, one-step methods (apart from *Combined*) show a larger improvement than two-step methods. This is not surprising, as they take interactions between examples from both domains into account, whereas two-step methods only "see" interactions within domains (see Figure 3.4). While two-step methods can be considered more powerful, the superior performance comes at a significantly higher cost. Considering inter-domain interactions not only leads to a higher performance but also to more expensive training in terms of time and memory.

Lastly, *AdvancedConvex* deserves some attention. While it performs mostly worse than *DualTask*, it is often second best (see Table 5.3). It combines the cheapest one-step method in terms of hyper-parameter optimization *SimpleUnion* with *TargetOnly* in a convex manner. That way it takes into account both, within-domain interaction as well as inter-domain interactions. It allows efficient trade-off parameter optimization, without having to re-train the combined classifiers. In case one faces a domain adaptation situation, where re-training is expensive, but inter-domain interactions significantly improve performance, *AdvancedConvex* constitutes a reasonable compromise.

5.5.1 Multiple Sources

In a second set of experiments we considered the three algorithms combining several sources (i.e. *MultiCombined*, *MultiConvex*, *MultiTask*). For our comparison we used 25,000 labeled examples from each of the four source organism to predict on a target organism. Thus the total number of source examples is identical to those used in single source experiments. The results are given in Figure 5.8 and a summary in Figure 5.9. We can make the following observations:

1. Relative to the corresponding single source algorithms, these algorithms perform better than the corresponding single source methods, if the organism is far away from the source used by the single source algorithm (see Table 5.4).
2. For distantly related organisms such as *D. melanogaster* and *A. thaliana*, the usage of multiple sources can lead to considerable improvements relative to the corresponding single source algorithms. This is most pronounced for small target data set sizes.

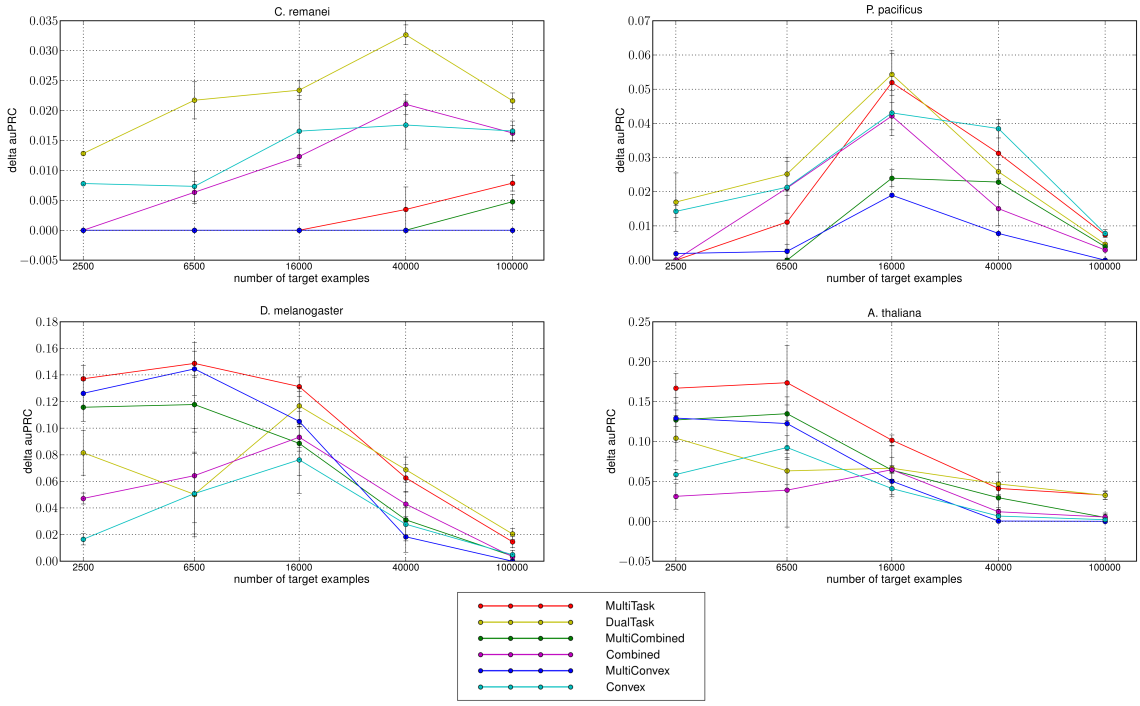


Figure 5.8: Results of multiple source domain experiments. Improvements over (single-source) baseline methods of three multi-source methods and their corresponding single-source methods are shown for each target organism. The x-axis shows the number of examples for the target domain, the y-axis shows the auPRC (area under precision recall curve) performance measure.

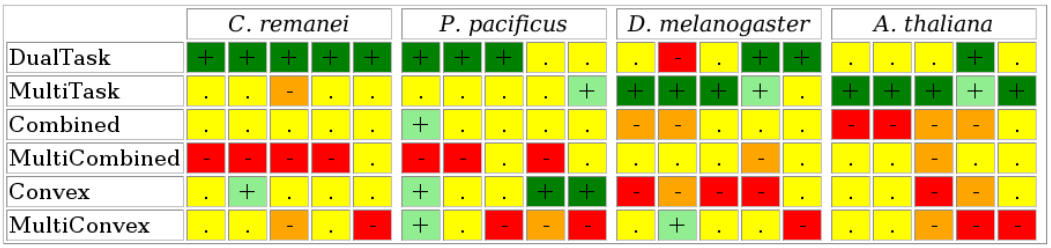


Figure 5.9: Summary of performance of multi-source domain methods (and corresponding single source methods) following the scheme defined in Figure 5.6. Briefly, ascending target data set sizes are contained in sub-columns, the best performing method is shown in green, the worst performing method is shown in red, while the remaining colors denote methods in between.

3. Among the multiple source algorithms *MultiTask* is most successful, as can be seen in Figure 5.9.

	C. remanei	*P. pacificus*	*D. melanogaster*	*A. thaliana*	Total
DualTask	1.0	1.8	2.8	2.8	2.1
MultiTask	4.4	3.0	1.4	1.2	2.5
Combined	2.8	3.8	4.0	4.4	3.75
MultiCombined	5.8	5.4	3.8	3.2	4.55
Convex	2.2	2.0	5.6	5.0	3.7
MultiConvex	4.8	5.0	3.4	4.4	4.4

Table 5.4: *Average ranks for multi-source experiments.*

From these experiments we observe that it pays off to use diverse multiple sources, in particular if there is no very close relative available. On the organisms *D. melanogaster* and *A. thaliana MultiTask* outperforms the other methods in 7 of 10 experiments.

In summary we observe that all domain adaptation algorithms lead to considerable improvements over the source and target models. For distantly related organisms learning from many different sources helps to improve the classification performance.

6 Discussion

In our comparison we determined the *DualTask* and *MultiTask* learning algorithms as the best performing methods. As pointed out earlier, it is not surprising that one-step methods, which take into account interactions between examples within domains as well as interactions between examples of different domains perform better than two-step methods which solely consider intra-domain interactions. These inter-domain interactions are important, because they allow a bidirectional exchange of information between domains. In conclusion, one-step methods are more expensive but more powerful as they take into account more information. Furthermore, a possible explanation of the superior performance of *DualTask* is the use of the logistic loss instead of the hinge loss. It is possible that the former is better suited for the data at hand. This question has to be investigated in future experiments. However, preliminary experiments suggest that similarly good performance can be achieved with the hinge loss as well.

We observe that *Combined*, whose training has the same space and time complexity and which has the same number of hyperparameters, performs worse than *DualTask* and *Augmented*. As pointed out in Section 3.4 the method *Combined* uses the loss-term of the SVM to introduce a trade-off between source and target, while *DualTask* utilizes the regularizer. Based on our results, we hypothesize that influencing the trade-off via the regularizer is better suited than via the loss term for this kind of domain adaptation problems. The exact reasons remain unclear and require a detailed analysis which is beyond the scope of this work.

It is possible to extend *DualTask* such that it allows an additional trade-off parameter via its loss term. However, a problem is that this leads to a three-dimensional search grid with respect to hyperparameter optimization. Either the search space is restricted to a very sparse grid (which might lead to worse performance) or training becomes extremely expensive. Additionally, tuning too many hyperparameters may lead to overfitting and thus to worse performance on unseen data. Yet, an analysis whether an extension of *DualTask* is beneficial could be part of future research.

Compared to the other domain adaptation methods, the performance of the method *Regtrick* is rather weak. When analyzing the dual of *Regtrick* (see Equation 3.13) one can see that it effectively reweights target examples based on their distance to the support vectors of the solution vector of *SourceOnly*. The resulting predictor f_{Reg} performs a convex combination between the solution learned on these reweighted target examples and the solution from *SourceOnly* (see Equation 3.7). Thus, it becomes clear that the reweighting step is likely to be responsible for the inferior performance compared to *Convex*. The problem could be related to the focus on support vectors in the reweighting step. Support vectors represent a rather extreme subset of the dataset (i.e. examples at class boundaries, outliers), which might lead to a problematic reweighting, in particular in extremely sparse high-dimensional

spaces. Benchmarking the *Regtrick* on other, real-valued or lower-dimensional datasets will be part of future research.

KMM was not part of the above experiments, because it did not bring significant improvements in preliminary experiments. Therefore, we considered a derivative of *KMM*, *Shift* which was even able to outperform *DualTask* in a few cases. While the idea of matching means by a translation of examples, rather than a reweighting of examples clearly has its benefits, the down-side of *Shift* is that computations have to be done explicitly in feature space and an efficient implementation is not obvious. However, *Shift* could be applicable to domain adaptation problems where *DualTask* fails because source and target distributions are too different. For the data at hand, however, we could observe only average performance for most experiments.

In large-scale machine learning, a central criterion for choosing a training algorithm is the complexity of the training procedure. Efficient hyperparameter optimization is the great benefit of convex combinations, as the combined classifiers do not have to be retrained, but only an optimal combination coefficient has to be determined. From our results we conclude that *Convex* delivers adequate performance (mostly comparable to more expensive methods) at a very low cost. The observation has not only been made in sequence analysis, but also in the field of natural language processing, where (Dau07) reported that simple methods are surprisingly hard to beat. By presenting the method *AdvancedConvex* we make the point that convex combinations can not only be used to combine classifiers that have been trained separately per domain. In fact, convex combinations of several of the methods presented above could bring improvements. In preliminary experiments, we learned that the combination of *Regtrick* and *TargetOnly* was able to considerably improve upon the performance observed for the regular *Regtrick*. However, in this work we refrained from considering all possible combinations because there are simply too many of them.

Generally, multi-task methods are of particular interest, as they can be used in more general settings than just the domain adaptation scenario. In domain adaptation, it is assumed that a large body of data is available for the source domain (or domains) while only a limited amount of data is available for the target domain. However, scenarios exist where only little data is available for *all* domains. In that case, multi-task methods are better suited because common knowledge between domains is shared, while a specific component is learned separately for each domain. That way, a predictor for each of the domains is obtained and not only for the target domain. This could be useful in many applications in computational biology.

6.1 Outlook

Application to Gene Finding From our results, we draw the following conclusions with respect to gene finding. As has been shown, simple convex combinations deliver adequate performance at a very competitive cost. In particular with respect to sequence signals for which plenty of training data is available, desirable performance levels can probably be reached by employing simple convex combinations of classifiers that were trained separately for each organism. On signals, however, where little data is available for virtually all organisms (such as transcription start sites) we could turn to *MultiTask* methods to create a joint classifier

for several organisms (see below). As pointed out before, the gene finder *mGene* decomposes the gene finding problem in a two-layered manner. The first layer deals with the recognition of sequence signals (e.g. splice sites), while the second layer integrates all of the individual sequence signals into a state model with a technique called structured output learning (THJA04). For the second layer a set of initial experiments has to be conducted in order to investigate whether the observations from the first layer carry over.

Universal splice site predictor Inspired by the good performance of the multi-task methods (using multiple sources), one could attempt to develop a method that learns to predict splice sites (or other sequence signals) for several organisms at the same time. The idea of multi-task methods is to define a trade-off between a general solution that is shared between the domains and domain-specific solutions. If we considered N domains, we could use a matrix W that encodes how important the similarity between w_i and w_j is for the optimal solution. One could take this idea further and use a phylogenetic tree to determine which domains are close and encode that information into the matrix W. That way, data from all available sources could be shared between domains with an appropriate degree of influence to obtain a predictor that works well for all organsisms. If it turned out that setting weights from phylogenetic trees does not work well, one could instead employ one of the domain distance measures presented in this work.

Efficient hyperparameter optimization Methods for efficiently determining the optimal cost parameter C (in the context of training an SVM) have for instance been proposed in (HRTZ04). The idea is to integrate the hyperparameter optimization into the regular training procedure. This idea has been extended by (WYL06) to the two dimensional case (two hyperparameters), setting cost vectors separately for each class. It would be interesting to explore, whether a similar approach can be applied to some of the presented domain adaptation algorithms in order to determine optimal trade-off parameters (e.g. the parameter B for *DualTask*) at a very low computational cost. Furthermore, (Cha07) propose SVM training procedures in the primal, including hyper-parameter optimizations. Again, this can potentially be used for efficient tuning of trade-off parameters for domain adaptation.

Kmer-reweighting The method *Kmer-reweighting* approaches the problem of domain adaptation from a different angle than most of the other methods that were considered in this work. Here, the focus is on reweighting features rather than examples. The development of the initial prototype and the evaluation on a set of experiments will be completed in the near future. Pursuing this direction of research might lead to improvements over the methods presented so far as one could potentially benefit from employing hand-tailored domain adaptation algorithms to the sequence prediction problem.

7 Conclusion

The aim of this work was to identify suitable methods to perform domain adaptation for splice site prediction. The main difficulties in splice site prediction arise from the extremely high dimensional feature space, unbalanced data sets and the fact that there is very little labeled data available for newly sequenced organisms. These challenges can be addressed by using efficient formulations of domain adaptation algorithms. The existing set of methods available for domain adaptation was complemented with extensions to some these methods and some new developments. In particular, substantial effort was spent on deriving an efficient implementation of the method *Regtrick*. Furthermore, the cost effective combination method *AdvancedConvex* was proposed.

In preparation of a thorough comparison, all methods were implemented and missing infrastructure was developed to perform large scale experiments in a cluster computing environment. To automate data storage and loading, the underlying C++ library *Shogun* was extended with serialization functionality using the *boost* library. The infrastructure that was created in the course of this project is likely to render useful in future projects.

In a set of initial experiments, we analyzed the similarity between domains. We presented and evaluated three distance measures for this. The *break even point* was identified as particularly interesting, as it gives an idea under which circumstances domain adaptation methods lead to the greatest improvement. Furthermore, POIMs constitute a powerful tool to compare sequence based problems. As we have shown, they are not only suited for visual comparisons, but can also be used to infer a numerical similarity measure.

As one of the main contributions of this thesis we conducted a large set of experiments with varying phylogenetic distances between organisms and data set sizes with the goal of assessing the performance of the domain adaptation algorithms in a number of different settings. These experiments would have taken about 1600 days of computing time on a single state-of-the-art CPU. We distinguish two classes of methods: *one-step* methods, for which a classifier is obtained by training on examples from source and target domains simultaneously, and *two-step* methods, for which a classifier is obtained by combining classifiers that were trained separately on the respective domain (e.g. a convex combination). We observed that one-step methods that take the inter- and intra-domain interactions into account mostly perform better than two-step methods which only consider intra-domain interactions. The superiority in performance comes at the cost of a considerably more expensive training procedure.

The one-step method *DualTask* has the best overall performance. It not only shows considerable improvements over baseline methods, but also outperforms the other domain adaptation methods in 12 of 20 experiments. Furthermore, the strong performance of the method *Convex* shows that simple methods are competitive. As the relatively inexpensive convex combination often performs not much worse than more expensive methods, we clearly con-

sider it to be the winner in terms of performance per cost. Moreover, we found the method *AdvancedConvex* to perform well, in particular on closely related organisms. As a compromise between one-step methods and the efficient hyperparameter optimization of convex combinations, it constitutes an excellent trade-off between performance and cost as well.

The most significant improvement over baseline classifiers could be achieved on target organisms in a rather distant relationship to *C. elegans*. Splice sites are quite well conserved between organisms, which means that the need for adaptation is not as pronounced between close organisms. For distant organisms on the other hand, intelligent adaptation can yield significant improvement over baseline methods. In a second set of experiments, we showed that it makes sense to combine data from several source organisms, if no single close relative is available. We presume that this way, we avoid an overly strong bias torwards the model from one source organisms and rather focus on the general aspects of splice sites across a range of organisms.

Based on our results, we can give the following recommendations to researchers who find themselves facing a domain adaptation scenario. As a general rule of thumb, the more distantly related the source and target domains are, the more effort has to be made to benefit from source domain information. As a first step to see how much we can gain from domain adaptation algorithms, one should compare the performance of the methods *SourceOnly* and *TargetOnly*. If they are similar (e.g. close to the break-even point), we can expect domain adaptation algorithms to work well. From our experience with splice site data, we recommend to employ the method *Convex* first and check whether the result is already good enough. Only if further improvements are desirable *and* the computational resources are available, one should turn to fully hyperparameter-optimized multi-task methods. When seeking a compromise between efficiency and performance, one could turn to *AdvancedConvex*. In particular when working with large datasets, such that hyperparameter optimization is very expensive, but at the same time one aims to improve upon the performance of *Convex*, the use of the *AdvancedConvex* can be recommended.

In summary, we conclude that domain adaptation algorithms are an effective tool to combine information from different domains in order to obtain an optimal classifier. The best suited domain adaptation method has to be chosen carefully, as it largely depends on framework requirements, such as desired accuracy or available resources and the data themselves. If a wise choice is made, however, predictive performance can be boosted considerably.

Bibliography

[BdBCP07] Shai Ben-david, John Blitzer, Koby Crammer, and O Pereira. Analysis of representations for domain adaptation. *NIPS*, pages 137–144, 2007.

[Bea96] D.M. Beazley. SWIG: an easy to use tool for integrating scripting languages with C and C++. *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996-Volume 4 table of contents*, pages 15–15, 1996.

[BHOS+08] A. Ben-Hur, C.S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support Vector Machines and Kernels for Computational Biology, 2008.

[BMP06] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2006.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[CA06] C. Chelba and A. Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. *Computer Speech & Language*, 20(4):382–399, 2006.

[CBL06] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[Cha07] O. Chapelle. Training a Support Vector Machine in the Primal. *Neural Computation*, 19(5):1155–1178, 2007.

[CL01] C.C. Chang and C.J. Lin. LIBSVM: a library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html`, 2001.

[Dau07] H. Daume. Frustratingly Easy Domain Adaptation. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 45(1):256–263, 2007.

[DG06] J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[DIM06] H. Daume III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.

[EP04] T. Evgeniou and M. Pontil. Regularized multi-task learning. *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, 2004.

[FGSS08] K. Fukumizu, A. Gretton, X. Sun, and B. Schölkopf. Kernel measures of conditional dependence. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 489–496. MIT Press, Cambridge, MA, 2008.

[GBR+07] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[GSH+08] A. Gretton, A. Smola, J. Huang, M. Schmittfull, and B. Schölkopf. Covariate shift by kernel mean matching. In J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, editors, *Dataset Shift in Machine Learning*. MIT Press, Cambridge, MA, 2008.

[HBVS04] S.B. Hedges, J.E. Blair, M.L. Venturi, and J.L. Shoe. A molecular timescale of eukaryote evolution and the rise of complex multicellular life. *BMC Evol Biol*, 4(2):1471–2148, 2004.

[HI07] C.A. Hutchison III. DNA sequencing: bench to bedside and beyond+. *Nucleic Acids Research*, 35(18):6227, 2007.

[HRTZ04] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the Support Vector Machine. *The Journal of Machine Learning Research*, 5:1391–1415, 2004.

[Hun07] J.D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[Jia08] Jing Jiang. A Literature Survey on Domain Adaptation of Statistical Classifiers. `http://sifaka.cs.uiuc.edu/jiang4/domain_adaptation/survey/`, 2008.

[Joa00] T. Joachims. SVMlight is an Implementation of Support Vector Machines (SVMs) in C. *University of Dortmund. Collaborative Research Center on Complexity Reduction in Multivariate Data (SFB475)*, 2000.

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open Source scientific tools for Python. `http://www.scipy.org`, 2001.

[JZ07] J. Jiang and C.X. Zhai. Instance Weighting for Domain Adaptation in NLP. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 264–271, 2007.

[KKSA05] T. Kato, H. Kashima, M. Sugiyama, and K. Asai. Multi-Task Learning via Conic Programming. *Advances in Neural Information Processing Systems*, 2005.

[KRM05] O.A. Kent, D.B. Ritchie, and A.M. MacMillan. Characterization of a U2AF-Independent Commitment Complex (E) in the Mammalian Spliceosome Assembly Pathway. *Molecular and Cellular Biology*, pages 233–240, 2005.

[Lew00] B. Lewin. Genes VII. *Oxford*, 2000.

[Met05] M.L. Metzker. Emerging technologies in DNA sequencing. *Genome Research*, 15(12):1767–1776, 2005.

[PDS04] A Pires-DaSilva and RJ Sommer. Conservation of the global sex determination gene tra-1 in distantly related nematodes. *Genes & Development*, 18(10):1198–1208, MAY 2004.

[PG07] F. Pérez and B.E. Granger. IPython: A System for interactive scientific computing. *Computing in Science & Engineering*, pages 21–29, 2007.

[RS04] G. Rätsch and S. Sonnenburg. Accurate splice site prediction for Caenorhabditis Elegans. *MIT Press series on Computational Molecular Biolog*, pages 277–298, 2004.

[SAB+06] E.M. Schwarz, I. Antoshechkin, C. Bastiani, T. Bieri, D. Blasiar, P. Canaran, J. Chan, N. Chen, W.J. Chen, P. Davis, et al. WormBase: better software, richer content. *Nucleic Acids Research*, 2006.

[SBB+03] LD Stein, ZR Bao, D Blasiar, T Blumenthal, MR Brent, NS Chen, A Chinwalla, L Clarke, C Clee, A Coghlan, A Coulson, P D'Eustachio, DHA Fitch, LA Fulton, RE Fulton, S Griffiths-Jones, TW Harris, LW Hillier, R Kamath, PE Kuwabara, ER Mardis, MA Marra, TL Miner, P Minx, JC Mullikin, RW Plumb, J Rogers, JE Schein, M Sohrmann, J Spieth, JE Stajich, CC Wei, D Willey, RK Wilson, R Durbin, and RH Waterston. The genome sequence of Caenorhabditis briggsae: A platform for comparative genomics. *PLoS Biology*, 1(2):166+, NOV 2003.

[SBS99] B. Schölkopf, C.J.C. Burges, and A.J. Smola. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[Shi00] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

[SRS05] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence SVM classifiers. *Proceedings of the 22nd international conference on Machine learning*, pages 848–855, 2005.

[SRSS06] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006.

[SS02] B. Schölkopf and A.J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[SSM99] B. Schölkopf, A.J. Smola, and K.R. Müller. *Kernel Principal Component Analysis*. MIT Press Cambridge, MA, USA, 1999.

[SSP+07] G. Schweikert, S. Sonnenburg, P. Philips, J. Behr, and G. Rätsch. Accurate splice site prediction using support vector machines. *BMC Bioinformatics*, 8(Suppl. 10):S7, 2007.

[STV04] B. Schölkopf, K. Tsuda, and J.P. Vert. *Kernel Methods in Computational Biology*. Bradford Books, 2004.

[SWSR08] G. Schweikert, C. Widmer, B. Schölkopf, and G Rätsch. An Empirical Analysis of Domain Adaptation Algorithms for Genomic Sequence Analysis (submitted). *NIPS*, 2008.

[SZPR08] S. Sonnenburg, A. Zien, P. Philips, and G. Rätsch. Poims: Positional oligomer importance matrices - understanding support vector machine based signal detectors. In *Bioinformatics/Proc. ISMB 2008*, volume 24, page i6. Oxford University Press, 2008.

[THJA04] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. *ACM International Conference Proceeding Series*, 2004.

[UVEB03] A Ureta-Vidal, L Ettwiller, and E Birney. Comparative genomics: Genome-wide analysis in metazoan eukaryotes. *Nature Reviews Genetics*, 4(4):251–262, APR 2003.

[Vap00] V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 2000.

[WYL06] G. Wang, D.Y. Yeung, and F.H. Lochovsky. Two-dimensional solution path for support vector regression. *Proceedings of the 23rd international conference on Machine learning*, pages 993–1000, 2006.

Appendix

	2500	6500	16000	40000	100000
AdvancedConvex	0.1923	0.2945	0.3831	0.4806	0.5397
Augmented	0.2462	0.2733	0.3819	0.4705	0.536
Combined	0.1735	0.2439	0.3809	0.4731	0.5433
Convex	0.1927	0.2452	0.3383	0.4293	0.5209
DualTask	0.2421	0.273	0.3849	0.4975	0.5654
Regtrick	0.2019	0.2255	0.3642	0.4477	0.5345
Shift	0.1708	0.2643	0.3683	0.4798	0.5596
SourceOnly	0.1422	0.1379	0.1392	0.1537	0.1422
TargetOnly	0.1023	0.1907	0.3256	0.4537	0.5328

Table 7.1: *A. thaliana*

	2500	6500	16000	40000	100000
AdvancedConvex	0.7589	0.7677	0.7734	0.7812	0.8004
Augmented	0.7537	0.761	0.7676	0.7773	0.7975
Combined	0.7567	0.7593	0.7717	0.7733	0.7995
Convex	0.7224	0.7653	0.7634	0.7741	0.789
DualTask	0.7706	0.778	0.7789	0.7902	0.8049
Regtrick	0.7594	0.7646	0.7692	0.7728	0.79
Shift	0.7578	0.7555	0.7723	0.778	0.7984
SourceOnly	0.7577	0.7594	0.7577	0.7577	0.7594
TargetOnly	0.2404	0.4645	0.6183	0.7044	0.7833

Table 7.2: *C. remanei*

	2500	6500	16000	40000	100000
AdvancedConvex	0.3333	0.3526	0.4942	0.5569	0.6183
Augmented	0.3871	0.4123	0.4931	0.562	0.622
Combined	0.3691	0.3928	0.4946	0.5598	0.6154
Convex	0.3653	0.399	0.4503	0.5202	0.5875
DualTask	0.408	0.3787	0.5233	0.5817	0.6326
Regtrick	0.3802	0.3881	0.4714	0.5465	0.6105
Shift	0.3529	0.4015	0.4899	0.546	0.6353
SourceOnly	0.326	0.3284	0.3302	0.326	0.326
TargetOnly	0.1392	0.259	0.397	0.5129	0.612

Table 7.3: *D. melanogaster*

	2500	6500	16000	40000	100000
AdvancedConvex	0.6493	0.6693	0.6747	0.7248	0.7581
Augmented	0.5767	0.6633	0.6729	0.7149	0.7499
Combined	0.6267	0.6596	0.6744	0.7034	0.7433
Convex	0.5895	0.6573	0.6503	0.7064	0.7476
DualTask	0.6472	0.6639	0.6844	0.71	0.7488
Regtrick	0.6411	0.6474	0.6679	0.703	0.7348
Shift	0.635	0.6607	0.6721	0.7146	0.7511
SourceOnly	0.6284	0.6387	0.6261	0.6387	0.6387
TargetOnly	0.1968	0.3816	0.5728	0.6844	0.7441

Table 7.4: *P. pacificus*

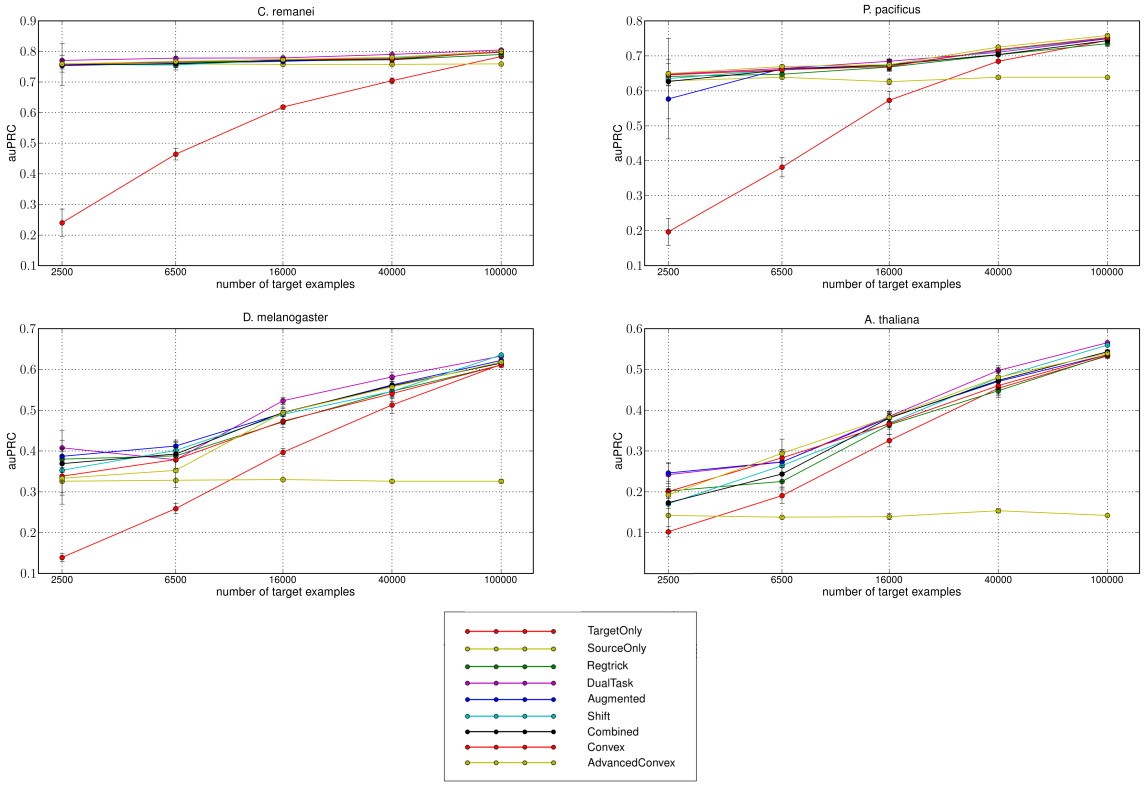


Figure 7.1: Single-source absolute results.

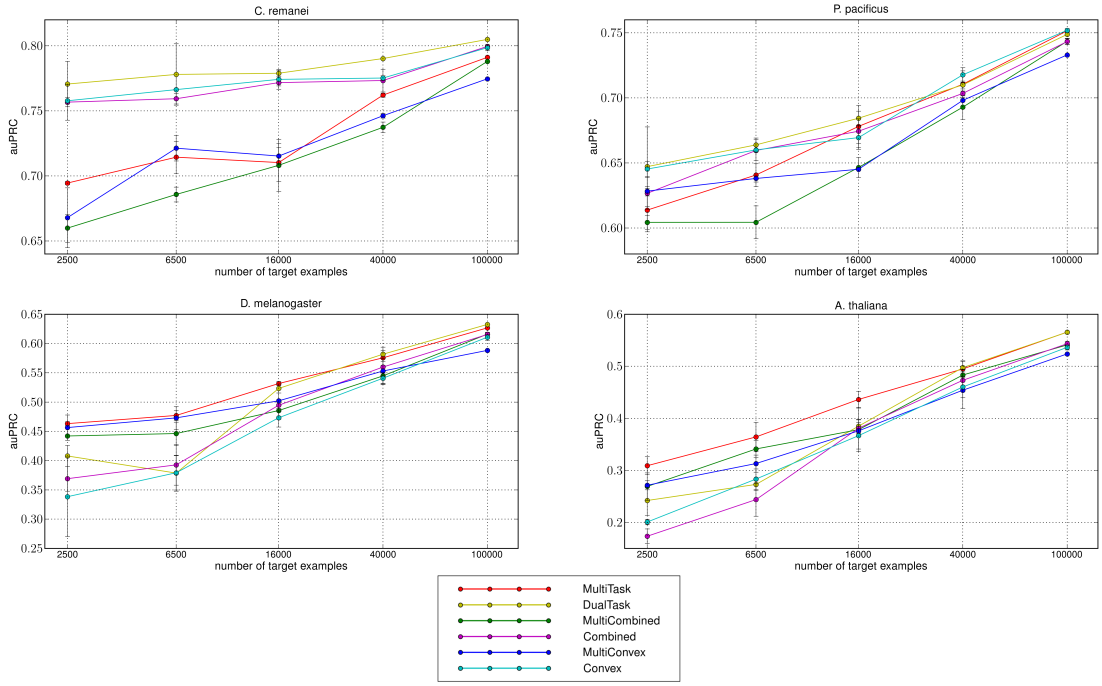


Figure 7.2: Multi-source absolute results.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Tübingen, den

Vorname Nachname des Bearbeiters