Prediction of Polycomb/Trithorax Response Elements using Support Vector Machines

Bjørn André Bredesen

Master's thesis

Department of Informatics
University of Bergen

June 3, 2013

Abstract

Polycomb/Trithorax Response Elements (PREs) are epigenetic elements that can maintain established transcriptional states over multiple cell divisions. Sequence motifs in known PREs have enabled genome-wide PRE prediction by the PREdictor and jPREdictor, using combined motif occurrences for scoring sequence windows. The EpiPredictor predicts PREs by using the method of Support Vector Machines (SVM), which enables the construction of non-linear classifiers by use of kernel functions. Aspects of using SVMs for PRE prediction can be investigated, such as setting of SVM parameters, using SVM decision values for scoring and using alternative feature sets.

The PRE prediction implementation presented in this thesis, called PRESVM, uses SVM decision values to score sequence windows. PRESVM implements the feature sets used by (j)PREdictor and EpiPredictor, as well as feature sets using relative motif occurrence distances and periodic motif occurrence. Grid search and Particle Swarm Optimization are supported for setting SVM parameters. For evaluating PRE predictions of multiple classifiers against experimental data sets, an application called PREsent has been implemented.

For a similar configuration for PRESVM and jPREdictor, PRESVM predicted a larger number of candidate PREs, which were more sensitive to but had lower Positive Predictive Values against experimental data considered than those of jPREdictor. A formal relationship was established between the PRESVM and jPREdictor decision functions for this configuration. The trade-offs make it difficult to conclude that either classifier is superior. Many configurations remain to be tested, and the results encourage further testing.

Acknowledgements

First, I would like to thank Marc Rehmsmeier (University of Bergen, Department of Informatics) for his supervision of the work with this thesis, with regular meetings with helpful and inspiring discussions. I would also like to thank Marc Rehmsmeier for proposing this project to me and thus introducing me to the exciting field of Polycomb epigenetics and the application of machine learning to genome-wide search.

I would like to thank Takaya Saito and Ksenia Lavrichenko in the Rehmsmeier group (University of Bergen, Department of Informatics) for sharing office with me and for friendly conversations.

I would also like to thank my younger brother, Marius Bredesen, with whom I have lived during my master's degree studies, and with whom I have had many inspiring conversations.

Contents

Chapter 1

Preface

The project for this thesis was proposed by Marc Rehmsmeier, my supervisor during my master's degree studies. Marc Rehmsmeier has worked with Leonie Ringrose on *in silico* prediction of instances of a type of DNA sequence element called Polycomb/Trithorax Response Elements (PREs for short). In this thesis, work continues on this task by applying a different method, the machine learning method of Support Vector Machines.

There are DNA sequences of known PREs, but what defines them is not well understood. The method Ringrose *et al.* [1] developed made use of pairing certain sequence motifs (short re-occurring substrings) in the known PREs, and they found that this better distinguished PREs from non-PREs than considering the motifs by themselves [1]. The machine learning method of Support Vector Machines enables modelling non-linear relationships, and the work by Ringrose *et al.* would suggests that non-linear relationships between motif occurrences could play a role. It is thus interesting to ask whether using Support Vector Machines might improve PRE prediction.

Soon after having started with this master's project, an article was published by Zeng *et al.* [2] in which Support Vector Machines were used for predicting PREs. However, multiple aspects of using Support Vector Machines for PRE prediction were not discussed. In this thesis, the use of Support Vector Machines for predicting PREs is explored further.

In Chapters 2-4, the background will be given. First, the biology of Polycomb/Trithorax Response Elements will be discussed. Afterwards, machine learning and Support Vector Machines will be explained. Then, the considered statistics for evaluating results will be given.

In Chapter 5, the method used in this thesis is outlined, and Chapter 6 and Chapter 7 elaborate on details of the method. Multiple parts of the method are based on the method developed by Ringrose *et al.* [1]. However, the use of Support Vector Machines has its own considerations to make, and these will be discussed. Also, Support Vector Machines are flexible, and making use of this flexibility will be explored.

It is important to evaluate how well the prediction results obtained using the method agree with experimental results. Chapter 8 outlines how such evaluations can be made. Results are presented in Chapter 9, including tests of some of the possibilities that the method of Support Vector Machines offers, as well as comparison with the method developed by Ringrose *et al.* [1].

Two main software applications have been developed during the work with this thesis, and their implementations are the subject of Chapter 10.

Finally, the thesis concludes with Chapter 11.

Chapter 2

Biological background

Inside the nucleus of eukaryotic cells, hereditary information is stored in chromosomes as sequences of *deoxyribonucleotides* (DNA). In prokaryotic cells, hereditary information is stored as DNA in the nucleoid. The DNA sequence of all chromosomes of a cell constitute its genome. Reading DNA sequences from DNA molecules is called sequencing. Today, multiple genomes have been sequenced. In this thesis, the focus is on the genome of the fruit fly, *Drosophila melanogaster*.

2.1 The Polycomb system

The *Drosophila melanogaster* genome contains over 120 million base pairs, divided on chromosomes X/Y, 2L/2R, 3L/3R and 4. The genome contains regions that encode functional products such as proteins, called genes [3], as well as regions with other functions. The FlyBase [4] *Drosophila melanogaster* annotation release 5.47 contains over fifteen thousand annotated genes. According to the central dogma of molecular biology, the genes are transcribed from DNA to messenger RNA, which in turn is translated to protein [5]. The transcription of genes to RNA is referred to as gene expression [3]. The genome is largely the same in all cells of an organism, but cells need the ability to specialize to perform particular functions in a multi-cellular organism. Thus, there is a need for regulating the expression of genes. It may also be necessary for cells to be able to remember gene expression states over cell division. The persistence of the expression states of genes over cell division, not caused by a change in the genomic sequence, nor by the event that established the expression states, is called epigenetics [6].

In *Drosophila melanogaster*, two types of genomic elements that are involved in regulating the expression of genes are called initiator elements, which are involved in establishing states of gene transcription, and maintenance elements, which maintain established gene transcription states [6].

Polycomb/Trithorax Response Elements (or PREs for short), the sequence elements that are investigated in this thesis, are maintenance elements. Polycomb group (PcG) and Trithorax group (TrxG) proteins associate with this class of *cis*-regulatory DNA elements (regulating on the same DNA molecule), and through the PREs they can maintain established transcriptional states over many cell generations, making the PREs epigenetic [6]. The roles of the PcG and TrxG proteins in PRE target gene expression state maintenance are antagonistic, where PcG proteins maintain transcriptional repression, whereas TrxG proteins maintain active transcriptional states [6].

On the PREs, PcG and TrxG proteins form complexes [7]. Currently, three PcG protein complexes have been identified in *Drosophila melanogaster*, called Polycomb Repressive Complexes 1 (PRC1) and 2 (PRC2), and Pleiohomeotic Repressive Complex (PhoRC) [7]. The PcG proteins at the core of these complexes are listed in Table 2.1. Of the PcG proteins, Pho binds to specific DNA sequences, and additional proteins that bind to PREs may be involved in the recruitment of the PcG complexes [7]. In addition to *Drosophila melanogaster*, there is also work on PcG proteins in vertebrates, where their involvement includes cancer and

maintaining stem cell and differentiated cell identity [8].

2.2 Determination of Polycomb/Trithorax Response Elements

Multiple methods have been employed in the search for Polycomb/Trithorax Response Elements. Early methods include transgene analysis and chromatin immunoprecipitation (ChIP) [1]. In recent years, multiple new methods have aided in the discovery of PREs. In 2003, Ringrose *et al.* [1] devised a computational procedure called the PREdictor, which enabled *in silico* prediction of PREs across the whole *Drosophila melanogaster* genome (genome-wide). Since then, the ChIP-chip (ChIP combined with microarray [9]) and ChIP-seq (ChIP combined with high-throughput sequencing [10]) methods have been developed and applied for discovering PREs and their associated target genes [11, 12, 13, 14, 15].

With the PREdictor, Ringrose *et al.* [1] used sequences of known 12 PREs and 16 non-PREs to predict PREs genome-wide. The 12 PRE sequences have lengths ranging from 1219bp to 5383bp, and the 16 non-PRE sequences have lengths ranging from 369bp to 7008bp. Alignment of the known PRE sequences has shown little sequence similarity, but these sequences are enriched in certain sequence motifs [1]. Sequence motifs are short, re-occurring strings of DNA. They may be degenerate, meaning that for some motif positions, multiple nucleotides might be accepted, and mismatches may also be accepted. Ringrose *et al.* [1] considered 7 sequence motifs, and these are listed in Table 2.2. Most of these correspond to PcG/TrxG protein binding sites [1].

The PREdictor scans the PRE and non-PRE sequences for occurrences of 7 sequence motifs and their reverse complements (corresponding to occurring on the opposite strand) (Figure 2.1(b)). Weights are assigned to the motifs or pairs of motifs according to how often they occur in the PRE versus non-PRE sequences, and this is used to assign scores to sequence windows. For evaluation, a 500bp sliding window is moved in 100bp increments across the PRE and non-PRE sequences (Figure 2.1(a)), the corresponding regions are scored, and the maximum window score obtained for each sequence is considered. Ringrose *et al.* [1] found that scores based on occurrence frequencies of individual motifs only weakly separate PREs from non-PREs. Ringrose *et al.* [1] then considered the frequencies with which pairs of the 7 motifs occur within 220bp from each other in PRE versus non-PRE sequences, and they found that these give better separation of PREs from non-PREs. The next step in their method establishes a score cutoff such that only one region is expected to have a higher score than the cutoff in a randomly generated sequence of the same length as and with the nucleotide distribution of the *Drosophila melanogaster* genome. The PREdictor was then applied across the *Drosophila* genome, which resulted in the prediction of 167 non-overlapping PREs. Multiple of the predicted PREs have been verified [16].

ChIP allows investigating the association between certain proteins and DNA [17]. This is done by cross-linking the proteins of interest and DNA. The sequence is then be split at random points (ideally uniformly), and the sequence fragments that are cross-linked to the protein of interest are isolated. The remaining sequence fragments are then treated with a micro-array (ChIP-chip [9]) or sequenced (ChIP-seq [10]). These methods have been applied in a number of studies, for genome-wide mapping of the binding of PcG and TrxG proteins to identify PREs and associated target genes [11, 12, 13, 14, 15].

Work on computational determination of PREs has since continued. In 2006, a new, versatile implementation of PREdictor, called the jPREdictor, was implemented in Java, containing a Graphical User Interface (GUI) and support for Position-Specific Scoring Matrix (PSSM) motifs [18]. Fiedler and Rehmsmeier [18] tested PRE prediction with the jPREdictor with the addition of a PSSM Pho motif and the DSP1 motif, which resulted in 378 PRE predictions. Additional motifs have since been discovered, and these are listed together with the DSP1 motif in Table 2.3. However, training the jPREdictor with the addition of the rest of these motifs did not result in improved PRE prediction [19]. In 2012, Zeng *et al.* devised the EpiPredictor [2], a Support Vector Machine PRE prediction method, in which the occurrence frequencies of the 7 sequence motifs considered by Ringrose *et al.* were used. In 2013, a Support Vector Machine PcG target gene

Complex	Core proteins
PRC1	
	Polyhomeotic (Ph)
	Posterior sex combs (Psc)
	Sex combs extra (Sce)
	Polycomb (Pc)
PRC2	
	Enhancer of zeste (E(z))
	Supressor of zeste 12 (Su(z)12)
	Nurf55
PhoRC	
	Pleiohomeotic (Pho)
	dSfmbt

Table 2.1: Core proteins of the PcG complexes of *Drosophila melanogaster* that have been characterized [7].

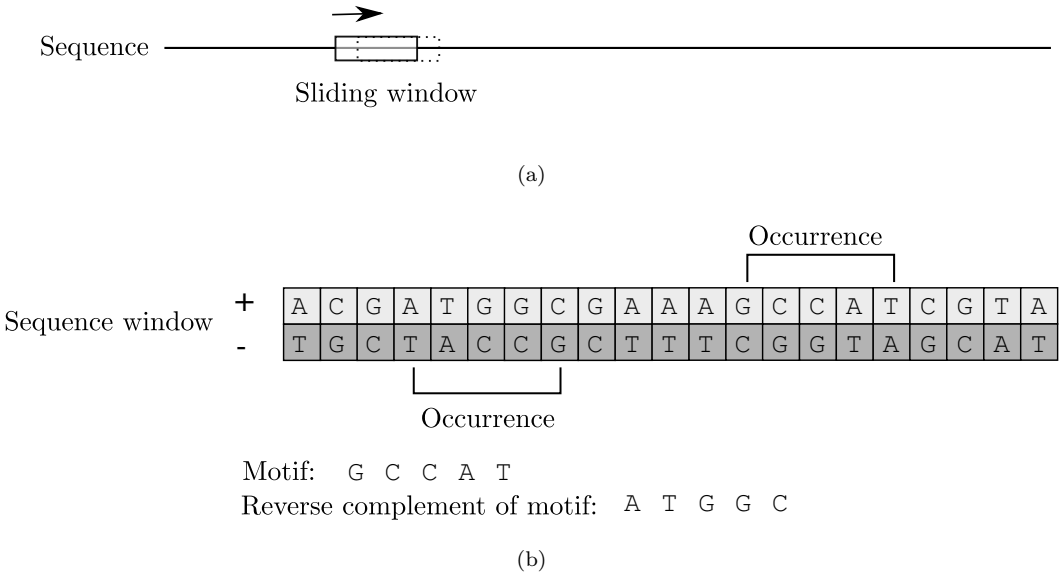


Figure 2.1: 2.1(a): A sliding window moves across the sequence in fixed increments. 2.1(b) Motif occurrences or their reverse complements are found.

Name	Function	Sequence motif	Allowed number of mismatches
G	GAGA factor (GAF) binding site	GAGAG	0
G10	Extended GAGA factor (GAF) binding site	GAGAGAGAGA	1
PS	Pleiohomeotic (Pho/Phol) core site	GCCAT	0
PM	Pleiohomeotic (Pho/Phol) consensus	CNGCCATNDNND	0
PF	Pleiohomeotic (Pho/Phol) consensus	GCCATHWY	0
EN 1	Motif important for *engrailed* PRE silencing	GSNMACGCCCC	1
Z	Zeste binding site	YGAGYG	0

Table 2.2: These sequence motifs were used by Ringrose *et al.* [1] for predicting Polycomb/Trithorax Response Elements with the PREdictor. The motif sequences are given in IUPAC nucleotide codes.

Binding protein	Sequence motif
Dsp1	GAAAA
Grainy head (Grh)	TGTTTTT
Sp1/KLF	RRGGYGY

Table 2.3: Since the original work of Ringrose *et al.* with the PREdictor, these motifs have been discovered [6]. The motif sequences are given in IUPAC nucleotide codes.

prediction method was devised that does not require a negative training set, using Mapping-Convergence [20].

In this thesis, the work continues on *in silico* PRE identification. Machine learning is central to this work, and is discussed in the next chapter.

Chapter 3

Machine learning

For many tasks, it can be difficult to formulate a particular algorithm that may perform the task well. The task might be poorly understood, and the system might need the ability to adapt to new information. One may, however, be able to formulate the problem in terms of experiences, such that if the system could learn from those experiences, the system would be able to perform the task.

Machine learning enables the construction of systems that can learn. Machine learning is applied to many different problem areas. Examples include problems of constructing systems producing advanced behaviours, such as robot control and playing games, and recognition and prediction problems. For the former type of problems, the system may learn from experimentation. For the latter, one typically selects examples for the system to learn from, and it may then be implemented as a classification problem. In this thesis, machine learning is used for classification.

3.1 Classification problems and learning

In classification problems, one has a set of objects, X, and a set of classes, $C = \{C_1, ..., C_n\}$. The task is to decide which class C_i an object $x \in X$ is an instance of. If there are two classes, this is called binary classification.

Definition 1 (Classifier) *For a set of objects X and a set of classes $C = \{C_1, ..., C_n\}$, a classifier is a function $c : X \to C$, which assigns classes C_i to objects $x \in X$. A decision function will here refer to a functional formulation of a classifier.*

Learning comes in when the correct definition of the classifier $c(x)$ is unknown, but classes are known for a subset of objects. It is then desirable to construct an approximation of $c(x)$ based on a set of objects with known classes.

Definition 2 (Function approximation) *A hat over a function name will be used to denote an approximately equal function, i.e. $\hat{c}(x) \approx c(x)$.*

Definition 3 (Cartesian product) *For two sets X and Y, $X \times Y = \{(x, y)|x \in X \wedge y \in Y\}$ denotes the Cartesian product.*

Definition 4 (Training set) *A training set is a set that will be presented to a learning machine. For using objects from a set X with known classes, it will be a set $T \subset X \times C$ of pairs of objects and known classes. The elements $(x, y) \in T$ will be referred to as training examples.*

Elements of a training set $(x, y) \in T$ may be presented to a learning machine. The learning machine constructs an approximation $\hat{c}(x) \approx c(x) = y$, which may be used to classify objects for which the correct class is not known *a priori*. This is an example of supervised machine learning, since the objects in the training set are labelled with classes (the alternative is called unsupervised learning). Supervised machine learning is often used for learning concepts, where the goal is to predict members versus non-members of a

concept. This can be implemented as a binary classification.

A machine learning method needs to store the knowledge that is to be learned. This storage will typically be in a form that makes generalizing assumptions about the relationships between the training examples that have been presented. This results in a model, and the process of generating or updating the model based on examples is called learning. The generalizing assumptions make it possible to use the resulting model for making predictions outside the training set.

Definition 5 (Model) *An approximation $\hat{c}(x) \approx c(x)$ generated by a machine learning method will be referred to as a model.*

Definition 6 (Learning) *The process of generating and updating a model based on training examples will be referred to as learning.*

The objects $x \in X$ may not be of a type readily useable by a machine learning method. It is then necessary to extract *features* from the objects that can be presented to the learning machine.

Definition 7 (Feature) *A feature $f(x)$ of an object $x \in X$ is a description of a property of x. In this thesis, such a property description will always be a real value, i.e. $f : X \to \mathbb{R}$.*

Definition 8 (Vector) *A vector $\vec{a} \in \mathbb{R}^d$ is a vector of real values $(a_1, a_2, ..., a_d)$.*

If the objects are strings of text, the features can for example be the frequencies with which particular words occur within the objects. For an object x, the word frequencies might then be collected into a vector of real values $\vec{x}$ and presented to the learning machine.

When applying machine learning to a classification problem, it is important to select a good training set and feature set. The features should be sufficiently descriptive of the objects belonging to each class, and the training set should contain objects that sufficiently represent the classes.

Ideally, the resulting model should generalize to the larger $X \times C$, *i.e.* $\hat{c}(x) \approx c(x), \forall x \in X$. If the learning machine learns the training set too well, it may be biased by the choice of training set, which can result in poor generalization. This is a common problem in machine learning and is called over-fitting [21, 22]. Multiple methods have been proposed to reduce this problem, and some of these will be described and discussed later.

There are a number of machine learning methods available, such as Artificial Neural Networks, Support Vector Machines, Bayesian methods and Decision Tree Learning. The focus in this thesis will be on Support Vector Machines (or SVM for short).

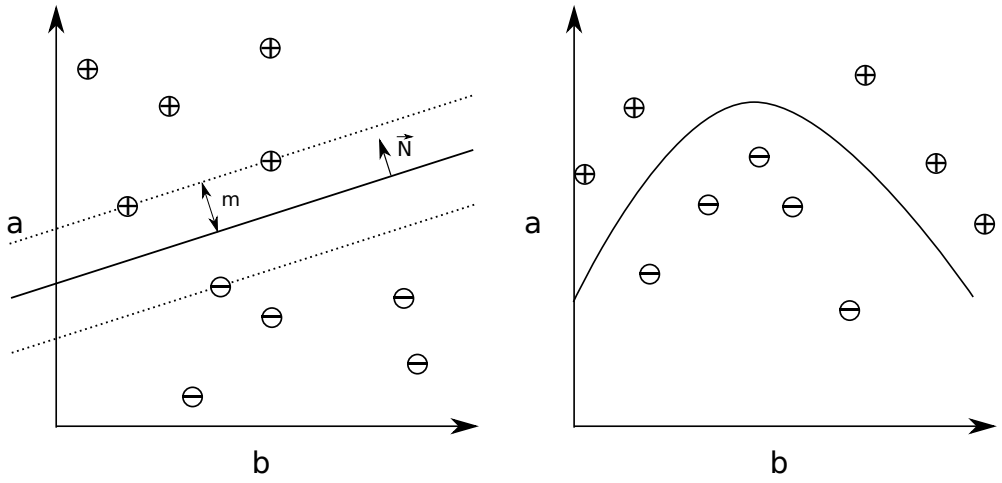
3.2 Support Vector Machines

If one can define a set of q real value features describing the objects in the domain under investigation $x \in X$, these can be collected into a vector $\vec{x}$, here called an instance vector.

Definition 9 (Instance vector) *For an object $x \in X$ and real value features $f_1(x), ..., f_q(x)$, the feature values may be collected into a vector $\vec{x} = (f_1(x), f_2(x), ..., f_q(x))$. Such a vector will be referred to as an instance vector.*

For n objects, this then gives n instance vectors in q-dimensional space. The instance vectors may additionally be associated with classes, such as whether or not the corresponding object is a member of a target concept.

The Support Vector Machine is a machine learning method that constructs a hyper-dimensional plane to separate instance vectors of different classes. This hyperplane is constructed to have maximal margin to instance vectors of each class. The vectors closest to the dividing hyperplane are called the support vectors, and



(a) (b)

Figure 3.1: 3.1(a): Example of a 2-dimensional feature space with instances drawn as $\oplus$ and $\ominus$ corresponding to positive and negative classes, respectively. The hard line is a potential dividing line, and the dotted lines denote the margins. $\vec{N}$ denotes the normal of the dividing line. 3.1(b) Example of a case where a line is unable to properly separate the training instances, but a non-linear surface can separate them.

the support vectors define the hyperplane with maximal margin. For a soft-margin classifier, some instance vectors may be ignored (treating them as noise), which may give better generalization than its alternative, called a hard-margin classifier [21]. Figure 3.1(a) illustrates this in two dimensions.

Definition 10 (Dot product) $\vec{a} \cdot \vec{b} = \sum_{i=1}^{q} a_i * b_i$ *for* $\vec{a}, \vec{b} \in \mathbb{R}^q$ *denotes the dot product of vectors* $\vec{a}$ *and* $\vec{b}$*.*

To construct a linear classifier, a hyperplane can be constructed to optimally separate the training instance vectors. Such a hyperplane can be represented by the vector equation $\vec{N} \cdot \vec{x} = b$, where $\vec{N}$ is the hyperplane normal vector, $\vec{x}$ is an arbitrary vector and b is an offset term. The signed distance from some instance vector $\vec{x}$ to this hyperplane is $\vec{N} \cdot \vec{x} - b$, where the sign may be used to assign a discrete class, *i.e.* $\hat{c}(\vec{x}) = sgn(\vec{N} \cdot \vec{x} - b)$ [21].

It may be that the instance vectors are distributed such that there is no hyperplane that separates the instance vectors. Figure 3.1(b) illustrates this issue. A non-linear classifier can be constructed by applying the kernel trick. This will be discussed in the next section.

3.3 Kernel function

The kernel trick involves using a transformation $\Phi(\vec{x}) = \vec{x^*}$ that maps vectors $\vec{x} \in \mathbb{R}^a$ to vectors in a higher-dimensional space $\vec{x^*} \in \mathbb{R}^b, b > a$. Based on this transformation, one can define the kernel function $k(\vec{x}, \vec{y}) = \Phi(\vec{x}) \cdot \Phi(\vec{y})$, which evaluates the dot product in this higher-dimensional space in terms of vectors in the original space. A linear classifier may then be constructed to separate vectors in this higher-dimensional space [21]. The decision function then may be $\hat{c}(\vec{x}) = sgn(\vec{N} \cdot \Phi(\vec{x}) - b)$.

The kernel function has a particular utility in the context of Support Vector Machines. The decision function of a Support Vector Machine makes use of the fact that the normal of the dividing hyperplane can be expressed in terms of the support vectors:

$$\vec{N} = \sum_{i=1}^{l} \alpha_i y_i \Phi(\vec{x_i}).$$

Here, l is the number of training instance vectors, $\vec{x_i}$ are training instance vectors, y_i are the associated target values (such as 1 for a positive instance and -1 for a negative instance), and α_i are Lagrangian multipliers, where for support vectors $\alpha_i > 0$ [21]. Substituting this into the decision function gives:

$$\hat{c}(\vec{x}) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i \Phi(\vec{x_i}) \cdot \Phi(\vec{x}) - b\right).$$

But this means that:

$$\hat{c}(\vec{x}) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i k(\vec{x_i}, \vec{x}) - b\right).$$

Thus, the decision function is formulated in terms of the kernel function, which can be simplified in terms of vectors in the original space. Thus, the computational complexity is not significantly increased by this mapping to higher-dimensional space.

In this thesis, the Support Vector Machine implementation in LibSVM will be used. Kernel functions provided in LibSVM include [23]:

- the linear kernel: $k(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}$;
- the polynomial kernel: $k(\vec{x}, \vec{y}) = (\gamma \vec{x} \cdot \vec{y} + c_0)^d, \gamma > 0$;
- the radial basis function kernel: $k(\vec{x}, \vec{y}) = e^{-\gamma \|\vec{x} - \vec{y}\|^2}$.

For these kernels, the parameters γ, c_0 and d are called kernel parameters. When used in this thesis, d will be locked to 2 and 3, referred to as quadratic and cubic kernels, respectively.

3.4 SVM formulations

There are multiple Support Vector Machine formulations. The formulations dictate how the Support Vector Machine is constructed from the training vectors. The formulations in LibSVM are C-Support Vector Classification, ν-Support Vector Classification, one-class SVM, ϵ-Support Vector Regression and ν-Support Vector Regression [24].

The C-Support Vector Classification requires only a parameter $C > 0$ [24], which is called the cost parameter, as an increase in C corresponds to treating errors as being more expensive when constructing the SVM [21]. Additionally, it supports the use of multiple classes. To use C-Support Vector Classification, class membership probabilities can be obtained for each class. Classification with more than two classes can be made binary by combining the probabilities into a score. In this thesis, the maximum probability for belonging to a positive class will be multiplied by one minus the maximum probability for belonging to a negative class to make a score. Additionally, the C-Support Vector Classification supports the weighting of different classes, which may be useful when when the training data is unbalanced [24].

The ν-Support Vector Classification only requires a parameter $\nu \in (0, 1]$, which controls for the fraction of training errors versus fraction of support vectors [24].

The one-class SVM formulation requires the parameter $\nu \in (0, 1]$ [24]. An interesting property of the one-class SVM is that only training instances of one class, the positives, are used during construction. By the standard implementation in LibSVM, one-class SVM does not give a continuous classification value. However, it can be tricked by switching its type to ϵ-Support Vector Regression after training, as it is handled identically in most cases after training.

Additionally, Support Vector Regression can be used to approximate a real valued function. The ϵ-Support Vector Regression has the cost parameter $C > 0$ in addition to a parameter $\epsilon > 0$ [24]. The ν-Support Vector

Regression has both the cost parameter $C > 0$ and $\nu \in (0, 1]$ [24].

The C-Support Vector Classification only requires the C parameter. The selection of parameters, discussed later, is automated and requires up to many training cycles, so a reduced number of parameters is useful. At the same time, the LibSVM implementation of the C-Support Vector Classification formulation clips probabilities as they approach 1, and this can pose issues when a larger threshold is desired (for example due to wanting very few predictions expected by chance). The ϵ-Support Vector Regression implementation on the other hand does not clip output values.

Chapter 4

Classifier validation

When one has constructed a classifier, one needs a way to measure how well it performs. This could for example be in order to compare different algorithms applied to the same kind of classification problem, or to compare multiple runs of the same algorithm but with different parameters. One may wish to determine how well an algorithm approximates the training data, and how well one might expect it to generalize to unobserved instances of the domain under investigation.

4.1 The confusion matrix and associated statistics

If the classification problem is binary, the classifier may return a score for an object $x \in X$, such that a larger value indicates larger confidence that x belongs to the class one is trying to predict members of.

Definition 11 (Binary label set) $\mathbb{B} = \{\oplus, \ominus\}$ *is the set of positive and negative classification labels, respectively.*

Definition 12 (Binary label assignment function) *For a real value $y \in \mathbb{R}$, $[y]_\pm \in \mathbb{B}$ denotes a function assigning a binary class label for y, such that $[y]_\pm = \oplus$ if $y > 0$, and $[y]_\pm = \ominus$ otherwise.*

If the classifier is of the form $\hat{c}(\vec{x}) \approx c(\vec{x}) = y$, where $\vec{x} \in \mathbb{R}^d$ is an instance vector and $y \in \mathbb{R}$ denotes a score, one obtains a binary classification for the classifier by applying a score threshold as $[\hat{c}(\vec{x}) - t]_\pm$. Given a set of pairs of instance vector and unique labels $S \subset \mathbb{R}^d \times \mathbb{B}$, the following sets can be defined:

Definition 13 (Original set classes)

$$P(S) = \{\vec{x} | (\vec{x}, y) \in S, y = \oplus\}$$

$$N(S) = \{\vec{x} | (\vec{x}, y) \in S, y = \ominus\}$$

where $P(S) \cap N(S) = \emptyset$

Definition 14 (Predicted set classes)

$$\hat{P}(S, \hat{c}, t) = \left\{\vec{x} | (\vec{x}, y) \in S, [\hat{c}(\vec{x}) - t]_\pm = \oplus\right\}$$

$$\hat{N}(S, \hat{c}, t) = \left\{\vec{x} | (\vec{x}, y) \in S, [\hat{c}(\vec{x}) - t]_\pm = \ominus\right\}$$

$\hat{P}(S, \hat{c}, t) \cap \hat{N}(S, \hat{c}, t) = \emptyset$

These constitute the sets of instance vectors that are positives, the ones that are negatives, the ones that are predicted as positives and the ones that are predicted as negatives, respectively. The condition $P(S) \cap N(S) = \hat{P}(S, \hat{c}, t) \cap \hat{N}(S, \hat{c}, t) = \emptyset$ ensures that only one label is assigned to each instance vector. Also, positives and negatives together cover all instance vectors of S, so $P(S) \cup N(S) = \hat{P}(S, \hat{c}, t) \cup \hat{N}(S, \hat{c}, t)$.

The relationship between predicted classes and actual classes may be summarized in a table called the confusion matrix. This will here be defined for the binary case.

Definition 15 (Set cardinality) *For a set S, $|S|$ refers to its cardinality, its number of elements.*

Definition 16 (Confusion matrix values) *The measures True Positives, True Negatives, False Positives and False Negatives, respectively, are defined as follows.*

$$TP(S, \hat{c}, t) = |P(S) \cap \hat{P}(S, \hat{c}, t)|$$
$$TN(S, \hat{c}, t) = |N(S) \cap \hat{N}(S, \hat{c}, t)|$$
$$FP(S, \hat{c}, t) = |N(S) \cap \hat{P}(S, \hat{c}, t)|$$
$$FN(S, \hat{c}, t) = |P(S) \cap \hat{N}(S, \hat{c}, t)|$$

For simplicity, the parameters of TP, TN, FP and FN will be dropped when they are not informative.

Definition 17 (Confusion matrix) *The confusion matrix for a binary classification problem is a 2x2 contingency table indicating the agreement and disagreement between predicted classes and actual classes [25].*

	$\hat{P}$	$\hat{N}$
P	TP	FN
N	FP	TN

A first measure that may be defined from the confusion matrix is the fraction of positives that are predicted as positives, known as the *Sensitivity*, *True Positive Rate* (*TPR*) and *Recall* [26]. In this thesis, the word *Sensitivity* will primarily be used.

Definition 18 (Sensitivity)

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

Similarly, a useful measure that can be defined from the confusion matrix is the fraction of predicted positives that are actual positives, known as the *Positive Predictive Value* (*PPV*) and *Precision* [26]. When used together with *Sensitivity*, the names *Recall* and *Precision* will be used.

Definition 19 (Positive Predictive Value)

$$PPV = Precision = \frac{TP}{TP + FP}$$

It is also useful to measure how many of the negatives are predicted as positives, known as the *False Positive Rate* (*FPR*) [26]. There is also the equivalence $FPR = 1 - Specificity$. When used together with TPR, the names *Sensitivity* and $1 - Specificity$ will primarily be used, or otherwise TPR and FPR.

Definition 20 (False Positive Rate)

$$FPR = 1 - Specificity = \frac{FP}{FP + TN}$$

One way to measure the overall classifier performance from the confusion matrix is to measure the proportion of instances that are correctly classified, which is referred to as the *Accuracy* [24].

Definition 21 (Accuracy)

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

An issue with this measure is that if the set of positives is larger than the set of negatives, then TN will have less of an impact on the *Accuracy* than TP. Similarly, if the set of negatives is larger than the set of positives, then TN will have a larger impact than TP. An alternative measure that uses all four of the confusion matrix values is *Matthews Correlation Coefficient* (*MCC*) [25].

Definition 22 (Matthews Correlation Coefficient)

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Sometimes, TN may be unknown or not well-defined. In such cases, an alternative to *Accuracy* and MCC is the F-measure, which combines *Precision* and *Recall*, and thus does not use TN [27].

Definition 23 (F-measure)

$$F_\beta = (1 + \beta^2)\frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

$\beta > 0$ is a weighting of *Precision* versus *Recall*. For $\beta > 1$, *Recall* has a larger impact on the fraction than *Precision*, and opposite for $0 < \beta < 1$.

4.2 Continuous quality measures

The confusion matrix values are discrete, in the sense that TP, TN FP and FN directly depend upon the size of the labelled set used for evaluation. Thus, the measures based on the confusion matrix values are also discrete. In certain cases, it can be useful to have continuous quality measures. For the binary case, continuous classification scores can be used. To define quality measures based on such values and allow the same values to occur multiple times, the values can be collected in vectors. One measure is the Pearson correlation coefficient [25].

Definition 24 (Pearson Correlation Coefficient) *For a vector $S = [s_1, ..., s_n] \in \mathbb{R}^n$, let $\overline{S} = \frac{1}{n}\sum_{i=1}^{n} s_i$ denote the mean vector component. Similarly, let σ_S denote the standard deviation of components of S. For vectors $A, B \in \mathbb{R}^n$, where $A = [a_1, ..., a_n]$ and $B = [b_1, ..., b_n]$, the Pearson Correlation Coefficient is given by*

$$C(A, B) = \sum_{i=1}^{n} \frac{(a_i - \overline{A})(b_i - \overline{B})}{\sigma_A \sigma_B}.$$

Definition 25 (Pearson Correlation Coefficient for classification values) *Consider a set $V = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ of n validation elements with labels $y_i \in \mathbb{B}$. Let $f(y_i) = 1$ if $y_i = \oplus$ and $f(y_i) = -1$ if $y_i = \ominus$. A vector of classification values can be denoted as $c = [\hat{c}(\vec{x_1}), ..., \hat{c}(\vec{x_n})]$. Similarly, a vector of target values can be denoted as $v = [f(y_1), ..., f(y_n)]$. The Pearson Correlation Coefficient between c and v can then be used as a quality measure.*

For a classifier $\hat{c}$, this quality measure gives the correlation coefficient between the classification values $\hat{c}(\vec{x}_i)$ and values based on correct labels for each $\vec{x}_i$ (1 for a positive label and -1 for a negative label). The respective means are subtracted from classification and target values, and the measure is scaled down by the standard deviations. Accordingly, the measure is determined by the distance of each classification values to the classification mean, with values closer to the mean having a smaller impact. When the target value and classification value are one the same side of their respective means, the measure increases, and otherwise it decreases.

The sum of squared errors has been used in gradient descent for measuring the error during training of neural networks [22]. A quality measure can be made by using one minus the sum of squared errors.

Definition 26 (1-sum of squared errors) *Consider a set $V = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ of n validation elements. Let $f(b) = 1$ if $b = \oplus$ and $f(b) = -1$ if $b = \ominus$. A vector of classification values can be denoted as $c = [\hat{c}(\vec{x_1}), ..., \hat{c}(\vec{x_n})]$. Similarly, a vector of target values can be denoted as $v = [f(y_1), ..., f(y_n)]$.*

$$1 - E^2 = 1 - \sum_{i=1}^{n}(v_i - c_i)^2$$

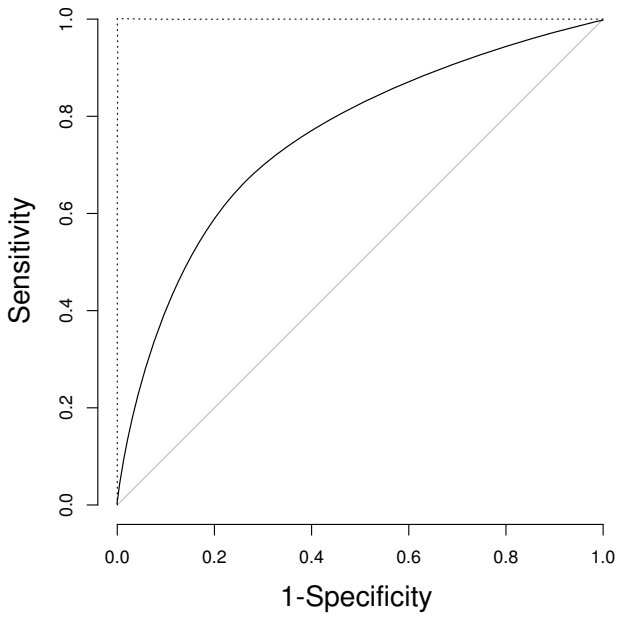


Figure 4.1: Example of a Receiver Operating Characteristic curve (solid line). The dotted line corresponds to a perfect classification. The diagonal corresponds to random classification.

4.3 Receiver Operating Characteristic curves

For binary classification as defined above, $[\hat{c}(\vec{x}) - t]_{\pm}$, the measures TP, TN, FP and FN are functions of the threshold value t. When t is maximal, all classifications will be negative. As t is decreased, more instances are classified as positive, until all instances are classified as positive. Thus, measures using confusion matrix cells may be plotted against one another by varying the threshold t.

Definition 27 (Receiver Operating Characteristic curve) *Receiver Operating Characteristic curves (ROC curves) are plots of the Sensitivity in the y-axis against* $1 - Specificity$ *in the x-axis [5].*

An example of a ROC curve is shown in Figure 4.1. Starting with no positive classifications gives a point in the bottom left of the plot, and as t is decreased and the classifier gives more positive classifications, some may be correct (contributing to the *Sensitivity*, upward) and some may be incorrect (contributing to $1 - Specificity$, rightward). Thus, a good classifier will have a ROC curve tending towards the upper left of the plot. A random classification will be around the diagonal. The construction of a Receiver Operating Characteristic curve is illustrated by Algorithm 3. This procedure could be optimized by ordering classifier score and label pairs according to scores and iteratively updating the confusion matrix values.

Since the ROC curve of a good classifier will tend towards the upper left of the plot, a quality measure has been defined to capture this tendency, called the Area Under the Curve (AUC or $AUROC$) [28]. This measure calculates the area under the ROC curve. In some cases, it may be desirable to only capture the classifier performance for lower $1 - Specificity$, and thus this measure will be defined as a function of the x axis coverage.

Definition 28 (ROC Area Under the Curve (AUC)) *Based on a ROC curve, the area under the curve in the interval* $0 \leq 1 - Specificity \leq x$ *can be calculated, and this will be referred to as* $AUC(x)$.

The regular $AUC/AUROC$ is then given by $AUC(1)$. This measure can be calculated by gradually generating points on the ROC curve and summing up areas below the curve. Algorithm 2 demonstrates the method.

Algorithm 1 ROC plot construction

Input:
$C \in \mathbb{R} \times \mathbb{B}$: A set of paired classifier scores and correct labels.
$makePoint(x, y)$: Makes a plot point with X-position x and Y-position y.
$makeLine(a, b)$: Makes a line from point a to point b.

$S \leftarrow \{v|(v, f) \in C\} \cup \{-\infty\}$ ▷ Get scores from the classifications for thresholds.
▷ For the final iteration, all should be classified as positive, thus the added $-\infty$.
$p_a \leftarrow 0$
while $S \neq \emptyset$ **do**
  $t \leftarrow \max S$ ▷ Get maximum scoring classification, to be used as a threshold.
  $S \leftarrow S/t$ ▷ Remove it from the list.
  $TP, FP, TN, FN \leftarrow 0$
  for all $(v, f) \in C$ **do** ▷ Find confusion matrix values for the cutoff.
    if $v > t$ **then**
      if $f = \oplus$ **then**
        $TP \leftarrow TP + 1$
      else
        $FP \leftarrow FP + 1$
      end if
    else
      if $f = \oplus$ **then**
        $FN \leftarrow FN + 1$
      else
        $TN \leftarrow TN + 1$
      end if
    end if
  end for
  $TPR \leftarrow \frac{TP}{TP+FN}$
  $FPR \leftarrow \frac{FP}{FP+TN}$
  $p_b \leftarrow makePoint(FPR, TPR)$ ▷ Note the point as the current line end ...
  if $p_a \neq 0$ **then**
    $makeLine(p_a, p_b)$ ▷ ... and make a line if there is a line start.
  end if
  $p_a \leftarrow p_b$ ▷ Make this line end be the start of the next line.
end while

Algorithm 2 Calculation of the $AUC(x)$ measure

Input:
$C \in \mathbb{R} \times \mathbb{B}$: A set of paired classifier scores and correct labels.

function $AUC(x)$
  $S \leftarrow \{v|(v, f) \in C\} \cup \{-\infty\}$ ▷ Get scores from the classifications for thresholds.
    ▷ For the final iteration, all should be classified as positive, thus the added $-\infty$.
  $AUC \leftarrow 0$
  $FPR_a \leftarrow 0$
  $TPR_a \leftarrow 0$
  while $S \neq \emptyset$ **do**
    $t \leftarrow \max S$ ▷ Get maximum scoring classification, to be used as a threshold.
    $S \leftarrow S/t$ ▷ Remove it from the list.
    $TP, FP, TN, FN \leftarrow 0$
    for all $(v, f) \in C$ **do** ▷ Find confusion matrix values for the cutoff.
      if $v > t$ **then**
        if $f = \oplus$ **then**
          $TP \leftarrow TP + 1$
        else
          $FP \leftarrow FP + 1$
        end if
      else
        if $f = \oplus$ **then**
          $FN \leftarrow FN + 1$
        else
          $TN \leftarrow TN + 1$
        end if
      end if
    end for
    $TPR_b \leftarrow \frac{TP}{TP+FN}$ ▷ Get new ROC point.
    $FPR_b \leftarrow \frac{FP}{FP+TN}$
    if $FPR_b > x$ **then** ▷ Restrict to $0 \leq FPR_b \leq x$.
      $FPR_b \leftarrow x$
      $TPR_b \leftarrow TPR_a + (x - FPR_a)(TPR_b - TPR_a)$
    end if
    $a \leftarrow FPR_b - FPR_a$
    $AUC \leftarrow AUC + aTPR_a + \frac{a}{2}(TPR_b - TPR_a)$
    if $FPR_b = x$ **then** ▷ End at the desired False Positive Rate.
      break
    end if
    $TPR_a \leftarrow TPR_b$
    $FPR_a \leftarrow FPR_b$
  end while
  return AUC
end function

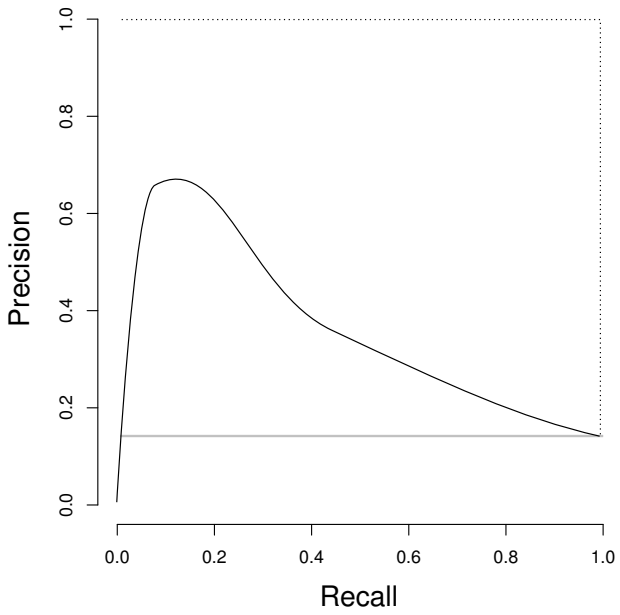


Figure 4.2: Example of a Precision/Recall curve. The dotted line marks a perfect classifier. The grey line marks random classification performance.

4.4 Precision/Recall curves

In some cases, the set of True Negatives, TN, may not be well-defined. For example, if the classification task is to mark certain phrases in a document, then TN is the set of regions marked neither in the validation set nor by the classifier, and is thus not well-defined defined due to the number of phrases in a document not being well-defined [27]. TN could also be very large, such that the curve could tend towards the upper left even if there are many false positives for each true positive. *Precision* and *Recall* do not depend on TN, and this gives an alternative plot for visualizing classifier performance.

Definition 29 (Precision/Recall curve) *In a Precision/Recall plot (PR plot), Precision is plotted on the y-axis and Recall on the x-axis [26].*

An example is shown in Figure 4.2. Similarly to the construction of ROC plots, PR plots are constructed by varying the threshold t. The construction of a Precision/Recall curve is illustrated by Algorithm 3. The approach is mostly identical to the construction of a ROC plot, except for using *Precision* and *Recall*.

For interpreting PR plots, it can be noted that going from left to right in a PR plot corresponds to increasing *Sensitivity*. The higher a point on the curve is, the higher the portion of positive classifications are correct. Thus, an ideal curve would cover the top of the plot. It has also been shown that a classifier dominates in a ROC plot if and only if it dominates in a corresponding PR plot [26]. A random classifier will have performance around a line flat over *Recall*, where $Precision = \frac{|P|}{|P|+|N|}$ [29]. However, calculating the expected random performance by this definition requires knowing N, and thus TN, and as noted these may be unknown.

4.5 Measuring generalization

A trained classifier can be applied to classify the training data to see how well it has been learned. However, it is usually more useful to measure how well one might expect the classifier to classify instances of the larger

Algorithm 3 PR plot construction

Input:
$C \in \mathbb{R} \times \mathbb{B}$: A set of paired classifier scores and correct labels.
$makePoint(x, y)$: Makes a plot point with X-position x and Y-position y.
$makeLine(a, b)$: Makes a line from point a to point b.

$S \leftarrow \{v|(v, f) \in C\} \cup \{-\infty\}$ ▷ Get scores from the classifications for thresholds.
▷ For the final iteration, all should be classified as positive, thus the added $-\infty$.
$p_a \leftarrow 0$
while $S \neq \emptyset$ **do**
  $t \leftarrow \max S$ ▷ Get maximum scoring classification, to be used as a threshold.
  $S \leftarrow S/t$ ▷ Remove it from the list.
  $TP, FP, TN, FN \leftarrow 0$
  for all $(v, f) \in C$ **do** ▷ Find confusion matrix values for the cutoff.
    if $v > t$ **then**
      if $f = \oplus$ **then**
        $TP \leftarrow TP + 1$
      else
        $FP \leftarrow FP + 1$
      end if
    else
      if $f = \oplus$ **then**
        $FN \leftarrow FN + 1$
      else
        $TN \leftarrow TN + 1$
      end if
    end if
  end for
  $Precision \leftarrow \frac{TP}{TP+FP}$
  $Recall \leftarrow \frac{TP}{TP+FN}$
  $p_b \leftarrow makePoint(Precision, Recall)$ ▷ Note the point as the current line end ...
  if $p_a \neq 0$ **then**
    $makeLine(p_a, p_b)$ ▷ ... and make a line if there is a line start.
  end if
  $p_a \leftarrow p_b$ ▷ Make this line end be the start of the next line.
end while

class under investigation. One approach to this is to apply the classifier to a set separate from the training data. This may not be practical if only a few instances of the class are known, as it may be desirable to use all of these instances for training. The technique of cross-validation provides an alternative.

The basic idea of cross-validation is to train the classifier on a subset of the training set and classify the remainder [21]. As the classified elements have not been used during training, this gives a measure of generalization. A confusion matrix can be constructed based on the resulting classifications.

Definition 30 (n-fold unbalanced cross-validation) *For a training set $T \subset X \times C$, shuffle its elements randomly and divide it into n (approximately) evenly sized subsets $T_1, ..., T_n$ (folds). For each fold $i \in [1, n]$, train the classifier on $T_1 \cup ... \cup T_{i-1} \cup T_{i+1} \cup ... \cup T_n$, and use the trained classifier to classify T_i [21].*

The unbalanced cross-validation approach derives its name from the fact that it does not consider the number of training examples for each class. If there are many more training examples of one class than of others, this could in some cases lead to folds with few or no training examples for some class or classes. An alternative is to first partition the training set according to classes, divide these partitions into class-specific folds, and join class-specific folds to form the folds used.

Definition 31 (n-fold balanced cross-validation) *For a training set $T \subset X \times C$ with m classes, divide T into subsets $T^1, ..., T^m$ according to class. For each class $j \in [1, m]$, shuffle the corresponding subset randomly, and divide it into n (approximately) evenly sized subsets $T_1^j, ..., T_n^j$ (folds). For each fold $i \in [1, n]$, make joined subsets $T_i = T_i^1 \cup ... \cup T_i^m$. For each fold $i \in [1, n]$, train the classifier on $T_1 \cup ... \cup T_{i-1} \cup T_{i+1} \cup ... \cup T_n$, and use the trained classifier to classify T_i.*

The number of folds can be selected to control for how many times the classifier is trained. Both balanced and unbalanced cross-validation use randomization for constructing the folds. Thus, there may be noise in quality measurements made on the folds. To reduce this issue, n-fold cross-validation can be repeated k times, and the quality can be measured over the full set of classifications, or it can be measured for each repeat and averaged. Alternatively, cross-validation can be done by only leaving a single training example out during training for validation, called Leave-One-Out cross-validation.

Definition 32 (Leave-One-Out cross-validation) *For a training set $T \subset X \times C$, let its elements be denoted by $t_1, ..., t_n$. For each element $t_i \in T$, train the classifier on $\{t_1, ...t_{i-1}, t_{i+1}, ..., t_n\}$ and use the trained classifier to classify t_i [21].*

Chapter 5

Prediction of Polycomb/Trithorax Response Elements with Support Vector Machines

In this thesis, Support Vector Machines are applied to the prediction of Polycomb/Trithorax Response Elements. This carries with it a series of design choices. The ideal is to develop an application that can take example sequences and additional parameters, and predict PREs in a given genome without further user intervention. The implementation of Polycomb/Trithorax Response Element prediction in this thesis is named PRESVM (read pre-S.V.M.). First, the use of Support Vector Machines for PRE prediction will be discussed, and then the design of PRESVM will be explained.

5.1 Why Support Vector Machines may improve prediction of PREs

The PREdictor scores sequence windows by summing weighted frequencies of paired motif occurrences within 220bp. Applying the PREdictor genome-wide with a score threshold calibrated for an E-value of 1, Ringrose *et al.* were able to identify 167 candidate Polycomb/Trithorax Response elements [1]. Since paired motif occurrences were found to be predictive of PREs, this may indicate non-linear relationships between motif occurrence frequencies.

Support Vector Machines have properties that could be beneficial for the prediction of Polycomb/Trithorax Response Elements. Any real valued features may be used with Support Vector Machines, so there may be feature sets other than paired motif occurrence frequencies that could help the prediction task. There might also be non-linear relationships between feature values, which an SVM might model.

In early 2012, Zeng *et al.* [2] published a Support Vector Machine implementation of PRE prediction. They found that non-linear kernels better distinguished PREs from non-PREs than a linear kernel. Thus, PRESVM is not the first Support Vector Machine implementation of PRE prediction. However, Zeng et al. [2] used single motif occurrence frequencies in windows as features for the SVM, so the question remains of whether other real valued feature sets might give improved PRE prediction. There are also additional considerations, such as selecting parameters for the Support Vector Machine and how to construct training vectors from training sequences.

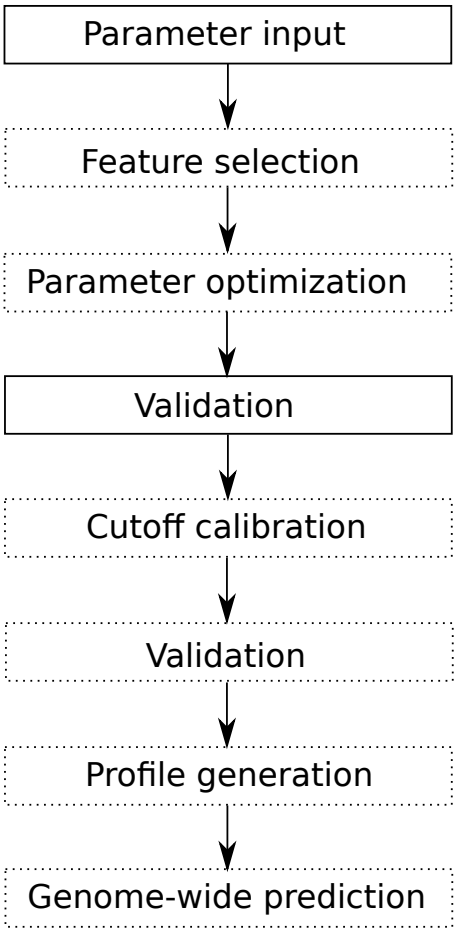


Figure 5.1: The PRESVM pipeline. A dotted box indicates that the step is optional.

5.2 PRESVM

PRESVM has been designed such that it after being supplied with parameters runs through all steps needed to make genome-wide predictions of PREs without requiring further user intervention. It is also useful to be able to evaluate different configurations without making genome-wide predictions. Thus, PRESVM consists of a pipeline with a series of optional steps, depicted in Figure 5.1.

The compulsory parameters are: 1) at least one feature set, 2) a set of motifs and 3) training sequences. Additional pipeline steps that should be executed can be specified. With threshold calibration and/or genome-wide prediction, a genome must also be specified.

In the feature selection step, a subset of the specified features are selected using mRMR feature selection [30]. In the parameter optimization step, a search is run for optimal SVM parameters. In the validation step, a classifier is constructed with the given configuration and is applied to the training sequences for validation, as well as any specified sets of validation sequences. During the cutoff calibration step, the classifier is applied to a large, randomly generated sequence, such that a cutoff is chosen for a desired E-value. After the cutoff calibration step there is a second validation step, such that the influence of the resulting cutoff on classification performance may be investigated. A profile is then generated to describe the run. Finally, the classifier is applied genome-wide for prediction. These steps are described in more detail later.

The profile generation step outputs a profile in XML format. This profile contains a detailed description of the configuration that was used during the run, as well as paths to files generated during genome-wide prediction. It also contains validation sequence scores, which can later be used for visualizing classifier performance via ROC and PR curves.

5.3 Support Vector Machines and sequences

To apply Support Vector Machines to the prediction of PREs, it must be decided how the classifier should be trained and how it should be applied for prediction. First, sequences and sequence coordinates can be defined.

Definition 33 (Sequence) *A sequence $S \in \mathbb{S}_{\mathtt{nt}}$ is a string of nucleotides (*`A`*,* `T`*,* `G` *and* `C`*).*

Definition 34 (Sequence coordinates) *For a sequence $S \in \mathbb{S}_{\mathtt{nt}}$ and $A, B \in \mathbb{N}$ where $A \leq B$, $S : A...B$ denotes the subsequence of S from nucleotide A (base 1) up to and including nucleotide B.*

Definition 35 (Sequence length) *For a sequence $S : A...B$, $S \in \mathbb{S}_{\mathtt{nt}}$, $\lambda(s) = B - A + 1$ denotes its length.*

The goal is to train a classifier and apply it across a whole genome to predict PREs. As was done with the PREdictor [1], the classifier can be applied across the genome using a sliding window.

Definition 36 (Sequence sliding window) *For a sequence $S : A...B$, $S \in \mathbb{S}_{\mathtt{nt}}$, a sliding window will refer to a function*

$$W(S, w, s, i) = S : X...Y$$

where $X = A + s(i-1)$ and $Y = \min\{X + w - 1, B\}$. $w > 0$ is the window width, $s > 0$ is the window stepping and i is the window index (base 1). When no range is specified, i goes from 1 up to including only one window where $Y = B$.

Assume there are training sequences $S_1, ...S_n$ and features $f_1(S_i), ..., f_m(S_i)$. The features can be applied to the full training sequences for constructing training vectors, or they can be applied using a sliding window over the sequences, giving a training vector for each window. Using a sliding window to make training vectors may give a very large training set, but a larger step size can be used during training to make up for this. Alternatively, one or more windows may be selected from each sequence as representative to construct training vectors.

It has been suggested that it is important that the feature values are scaled and shifted to have the same range over the training examples, and the same scaling and shifting should also be used when the trained classifier is to be applied for classification [23].

In the case of non-PRE sequences, it can be noted that every window in a non-PRE training sequence should correspond to a negative, and thus either the full sequences or sequence windows can be used for constructing training vectors.

In the case of PRE sequences, it may be that up to many of the sequence windows are not important for PRE function. The PREdictor [1] PRE training sequences have lengths in the range $1.2kb - 5.4kb$, whereas it has been suggested that PREs have lengths of only a few hundred base pairs [6]. Also, the sequences may vary in length, as with the PREdictor [1] training sequences, which has implications both for training with the full sequences and with sequence windows. If the full sequences are used for training, there are certain feature sets (discussed in Chapter 6) where sequence lengths are used for normalization, and because of this, differing sequence lengths might result in important features being normalized away. If training is done using sequence windows, there may be more windows for some sequences. For using sequence windows for training, it should be noted that a soft-margin classifier might treat irrelevant windows as noise. If irrelevant windows put constraints on the decision surface and thus distort it, then the exclusion of these windows could lead to a decision surface that better separates PREs from non-PREs. Alternatively, representative sequence windows

might be selected directly. However, for trying to select representative windows directly from the training sequences, it may be difficult to decide on a good selection criterion, and a poor selection criterion might have a negative impact on the resulting classifier.

After a classifier has been trained, it is desirable to apply it to sequences for validation, such as the training sequences. Since the classifier will be applied across the genome using a sliding window, it makes sense to also apply it using a sliding window during validation. If all validation sequence window scores are used to measure classification performance, there is the potential issue that longer sequences may have more windows, and thus have a higher influence on the classification performance measurement. To avoid this, the window scores for each validation sequence can be combined into a single sequence score. In this thesis, this will be done by letting the maximum window score for a sequence be the score of the full sequence, as was done with the PREdictor [1].

$$score(S) = \max_i \hat{c}((f_1(W(S, w, s, i)), ..., f_m(W(S, w, s, i))))$$

The classifier threshold may then be applied to obtain a binary classification for each sequence, and this enables the construction of a confusion matrix for the validation sequences.

5.4 Training data

To train a classifier to predict PREs genome-wide, it is necessary to have a set of known PRE sequences for the classifier to learn from. For Support Vector Machines, other than in the case of one-class SVM, it is also necessary to have sequences of one or more non-PRE classes to learn from. In this thesis, the training data used by Ringrose *et al.* [1] will be used. Additionally, the construction of alternative training data will be explored briefly.

After a classifier has been trained, one may attempt to apply the trained classifier to the PRE training sequences in order to identify significant sub-sequences. If the classifier can already be expected to give reasonable genome-wide predictions of PREs, it may be hoped that the classifier will assign higher scores to the more important PRE training sequence regions. One may then apply the trained classifier to the PRE training sequences and select an equal number of high-scoring windows from each training sequence as representative. These windows can then be used to train a new classifier. This is referred to as re-training in this thesis. If the selected windows are indeed important for PRE function, the removal of other sequence regions might amplify relevant features, and thus might improve prediction.

Ringrose *et al.* [1] defined PREs based on published coordinates of known PRE/TREs. For non-PREs, they used promoter regions of genes that are regulated by the Z and GAF motifs. When using a multi-class SVM, it is possible that training with additional classes may have an impact on predictions. One may for example add a class of randomly generated sequences with the same nucleotide distribution as the genome, corresponding to non-PREs.

In addition to the training sequences used by Ringrose *et al.* [1], it may be interesting to train a classifier using data from the recent genome-wide ChIP-chip/ChIP-seq studies, such as [12, 14, 15]. For positive (PRE) training examples, experimentally determined regions can be selected. If a selection of sequence motifs have already been chosen to be used for the classification task, the regions may be filtered for having occurrences of these motifs. Alternatively, motifs can be selected based on the experimentally determined regions. The selected regions can further be filtered, for example according to whether or not they are intergenic. Afterwards, a selection of the regions can be selected randomly and taken as PRE training examples.

For negative training examples, one would ideally want to select genomic regions that one can be fairly confident are not PREs. If sequence motifs are used, one possibility is to find genomic regions that are enriched in the sequence motifs and randomly select a number of regions and use them as non-PRE training examples. As the regions are selected due to being enriched in the motifs, but paired occurrences of motifs

are more predictive of PREs [1], it may be speculated that perhaps many or most of the selected regions will not be PREs. However, there is a risk of selecting PREs using this approach. With a soft-margin classifier, it is possible that false negatives will be filtered out as noise.

An alternative to selecting a set of non-PRE training examples is to use Mapping-Convergence, as was recently done for the prediction of PcG target genes in humans using Support Vector Machines [20]. Mapping-Convergence starts with a positive set and an unlabeled set, and the unlabeled set is used to iteratively construct a negative training set. This has not been implemented in PRESVM.

5.5 Genome-wide prediction

When predicting PREs in a genome, the classifier is applied across a whole genome in windows. Each window will be given a score by the classifier, and this results in a score profile for each chromosome. In PRESVM, these score profiles are exported to a Wiggle format file. This format was chosen due to its support by the Integrated Genome Browser [31], and is explained in UCSC Genome Bioinformatics FAQ for formats [32]. The Wiggle format is a human readable format, containing chromosome names, window coordinates and corresponding profile values. The Wiggle format allows defining the length and step size for windows at the start of a chromosome, such that the window scores may be given compactly, separated by line breaks. In addition to being able to visualize such score profiles using the Integrated Genome Browser, such score profiles also allow for other informative analyses, such as the correlations between different profiles.

A score threshold is applied to give binary classifications to windows. When applying a threshold to window scores, overlapping positively classified windows are merged, giving a selection of non-overlapping predicted regions in the genome. Such regions will here be referred to as *bands*.

Definition 37 (Band) *Bands will refer to positively classified regions, where overlapping regions have been merged.*

In PRESVM, predicted bands are exported to a General Feature Format (GFF) file, and as with the Wiggle format it was chosen due to its support by the Integrated Genome Browser [31], and is explained in UCSC Genome Bioinformatics FAQ for formats [32]. GFF is a human readable format. GFF files consist of lines specifying genomic regions. In addition to visualizing GFF files with the Integrated Genome Browser, these files are also useful for investigating the overlap of predictions with experimentally determined regions and what annotated genes are proximal to each predicted band. Genome-wide prediction is illustrated by Algorithm 4.

5.6 Classifier threshold calibration

To predict genomic regions as being PREs, the classifier threshold must be set to a reasonable value. It is desirable to set the threshold such that one has an idea of how many positive predictions can be expected by chance. That is, it is desirable to select the threshold for some desired E-value. In this thesis, this will be done similarly to how it was done with the PREdictor [1].

First, the distribution of nucleotides in the genome is found. Based on this distribution, a random sequence is generated. A trained classifier $\hat{c}$ is applied to this sequence in windows as it would for genome-wide prediction. If this sequence is of the same length as the genome, then the number of predictions made on this randomly generated sequence for some threshold t gives an approximate E-value. Thus, the threshold can be varied, and the approximate E-value can be noted for each threshold. This gives an approximated E-value distribution. The final threshold can then be selected based on the E-value one desires.

Algorithm 4 Genome-wide prediction

Input:
$Chromosomes$: Set of chromosome sequences.
$\hat{c} : \mathbb{R}^n \to \mathbb{R}$: Classifier.
$f_1, ..., f_n : \mathbb{S}_{\mathtt{nt}} \to \mathbb{R}$: Sequence features.
$w_{size} \in \mathbb{N}$: Window width ($w \geq 1$).
$w_{step} \in \mathbb{N}$: Window step size ($s \leq w$).

$P \leftarrow \emptyset$ ▷ Set of genome-wide predictions.
for all $C : A...B \in Chromosomes$ **do**
  $i \leftarrow A$
  $p_a, p_b \leftarrow -1$ ▷ This will note the start/end of a prediction when not -1.
  loop
    $j \leftarrow \min\{i + w_{size}, B\}$
    $W \leftarrow C : i...j$ ▷ Get the window, cropped by the sequence length.
    $v \leftarrow \hat{c}((f_1(W), ..., f_n(W)))$ ▷ Score the window.
    if $v > t$ **then** ▷ Score above threshold so update prediction.
      if $p_a = -1$ **then**
        $p_a \leftarrow i$ ▷ Prediction not started, so note start.
      end if
      $p_b \leftarrow j$
    else if $p_a \neq -1 \wedge i > p_b$ **then** ▷ Score below cutoff, and passed prediction end, so finish it.
      $P \leftarrow P \cup \{C : p_a...p_b\}$ ▷ Register prediction.
      $p_a, p_b \leftarrow -1$
    end if
    $i \leftarrow i + w_{step}$
    if $j = B$ **then** ▷ Last window in the sequence, so break out of the reading loop.
      break
    end if
  end loop
  if $p_a \neq -1$ **then** ▷ Finish any open prediction.
    $P \leftarrow P \cup \{C : p_a...p_b\}$
  end if
end for
return P

If the desired E-value is low, such as 1, the threshold found can be expected to vary significantly from run to run when the randomly generated sequence is of the same length as the genome. The threshold can be made more stable by scaling up the size of the randomly generated sequence, and scaling down the approximated E-values. Let $b(\hat{c}, t)$ denote the number of predictions made by a classifier $\hat{c}$ on the randomly generated sequence when using threshold t. Let l denote the size of the genome, let $s \geq 1$ be a scaling factor, and let the length of the randomly generated sequence be $l * s$. The approximated E-value is then given by:

$$\hat{E}_{bands}(\hat{c}, t) = \frac{b(\hat{c}, t)}{s},$$

The scaling factor s can be used to make the thresholds for E-values more stable. Given a constant probability for each window in the randomly generated sequence to be predicted, increasing the length of the sequence increases the expected number of predicted windows, and thus the number of predicted bands is divided by the scaling factor. This in turn reduces the influence of individual predictions on the approximated E-value.

It then needs to be decided how the threshold should be varied. One could find all window classification values over the full randomly generated sequence, and then iteratively use each as a threshold and find the number of predictions (bands) in the random sequence. This can be computationally very expensive if the scaling is large. Typically, a low E-value, such as $\hat{E}_{bands}(\hat{c}, t) = 1$ is desired. To make the threshold fairly stable a large scaling factor such as $s = 100$ should be used. Thus, it is desirable to have a more efficient approach for varying the threshold.

If the E-value of interest is low, there should be a limited amount of predictions in the randomly generated sequence. Each of these predictions may be based on up to multiple overlapping window classifications, but the number of overlapping positively predicted windows should then typically be limited. A possibility then is to find n top-scoring windows in the randomly generated sequence. The threshold can then be varied over the scores of these windows. As these windows are top-scoring, only these windows would be classified as positive in the randomly generated sequence when using their scores as threshold values, and one can therefore avoid classifying all windows from the randomly generated sequence multiple times. For each threshold, taking the overlaps between positively predicted top-scoring windows into account gives the predicted regions, and this can be used to get a corresponding approximated E-value.

This approach, called the highscore based E-value threshold calibration method, is implemented in PRESVM. An issue with this approach is that the number n of top-scoring windows to find in the randomly generated sequence must be decided. If n is too small, the threshold for the desired E-value may not be found. One possibility is to select n based on an expectation of how many windows will overlap in a prediction. If one expects i overlapping windows for a prediction, one possibility is to use $n = i * s * E$, where E is the desired E-value and s is the scaling factor for the randomly generated sequence. In PRESVM, n is chosen by the following equation

$$n = 2s * E * \frac{w_{size}}{w_{step}}$$

where w_{size} is the window size used and w_{step} is the sliding window step size. Thus, if the step size is decreased, n will increase. This makes sense, since if the step size is reduced, one would expect more overlapping windows to be predicted as positive. The chosen n may be larger than necessary, but this should reduce the probability of n being too small. The approach is illustrated in Algorithm 5.

Another way to vary the threshold is to find the range of window classification values over the genome, and to divide this range into m evenly spaced thresholds. The classifier is then applied to the randomly generated sequence, and the number of predictions for each threshold is noted. The approximated E-value for each threshold is then calculated based on the corresponding number of predictions.

As shown for genome-wide prediction, predictions can be made as the classifier is applied to the sequence. Thus, this approach can be implemented by keeping the necessary prediction state variables for each of the

Algorithm 5 Highscore based E-value threshold calibration

Input:
$\hat{c} : \mathbb{R}^n \to \mathbb{R}$: Classifier.
$f_1, ..., f_n : \mathbb{S}_{\mathtt{nt}} \to \mathbb{R}$: Sequence features.
$w_{size} \in \mathbb{N}$: Window width ($w \geq 1$).
$w_{step} \in \mathbb{N}$: Window step size ($s \leq w$).
$s \in \mathbb{N}$: Random sequence scaling factor.
$\lambda(G)$: Size of the genome.
$R : A...s\lambda(G)$: Randomly generated sequence.
$UpdateHighscore(v, i, j)$: Updates the highscore with score v for the window spanning from i to j. If the highscore is not full then an entry for the window spanning from i to j with score v is added to the highscore. Otherwise, if v is greater than the lowest window score in the highscore, the lowest scoring highscore entry is replaced with an entry for the window spanning from i to j with score v.
$GetHighscores()$: Gets the set of top window scores found.
$GetPredictions(x)$: Gets the number of predicted bands for treshold x based on the high-score windows.

$i \leftarrow A$
loop
  $j \leftarrow i + w_{size}$
  if $j \geq \lambda(R)$ **then** ▷ Last window in the sequence, so break out of the reading loop.
    break
  end if
  $W \leftarrow R : i...j$ ▷ Get the window.
  $v \leftarrow \hat{c}((f_1(W), ..., f_n(W)))$ ▷ Score the window.
  $UpdateHighscore(v, i, j)$
  $i \leftarrow i + w_{step}$
end loop
for $x \in GetHighscores()$ **do**
  $\hat{E}(x) \leftarrow \frac{1}{s} GetPredictions(x)$ ▷ Calculate the approximated E-value for each threshold.
end for

m thresholds considered, and to update them for each threshold as the classifier is being applied to the randomly generated sequence. Note, however, that only some prediction state variables will change for some classification value changes. If the classification value increases, this may correspond to the beginning of one or more predicted regions. If the classification value decreases, it may similarly result in some predicted regions having ended.

The interval based E-value calibration method is implemented in PRESVM. A reasonable value for m, the number of thresholds, should be chosen. In PRESVM, m is chosen using the following equation:

$$m = 50 * s$$

where s is the scaling factor for the randomly generated sequence. It makes sense to scale m with s, as an increase in s should make finer threshold E-value approximations more stable. The approach is illustrated in Algorithm 6.

5.7 Comparison of PRESVM and other PRE prediction methods

To predict PREs, the PREdictor assigns scores to sequence windows. These scores are weighted sums of paired motif occurrence frequencies. Using PRESVM with a similar feature set and a linear kernel allows for a comparison of PRESVM and the PREdictor.

In LibSVM, the ϵ-Support Vector Regression function approximation when using a linear kernel is given by

$$\begin{aligned}\hat{c}(\vec{x}) &= \sum_{i=1}^{l} (c_i k(\vec{y}_i, s(\vec{x}))) - \rho \\ &= \sum_{i=1}^{l} (c_i \vec{y}_i \cdot s(\vec{x})) - \rho.\end{aligned}$$

where the $\vec{y}_i$ are support vectors, and the c_i are coefficients. It is important to scale feature values to have the same range over the training data [23], and thus $s([x_1, ..., x_l]) = [(x_1 + q_1)r_1, ..., (x_l + q_l)r_l]$ denotes the feature vector scaling transformation, where q_i is the shifting of feature i and r_i is the scaling factor.

The dot product in the decision function can be reformulated as a vector component product sum, and rearranged as follows:

$$\begin{aligned}\hat{c}(\vec{x}) &= \sum_{i=1}^{l} (c_i \vec{y}_i \cdot s(\vec{x})) - \rho \\ &= \sum_{i=1}^{l} \left(c_i \sum_j (y_{i,j}(x_j + q_j)r_j) \right) - \rho \\ &= \sum_j \left(\sum_{i=1}^{l} (r_j c_i y_{i,j}) (x_j + q_j) \right) - \rho \\ &= \sum_j \left(\sum_{i=1}^{l} (r_j c_i y_{i,j}) x_j \right) + \sum_j \left(\sum_{i=1}^{l} (r_j c_i y_{i,j}) q_j \right) - \rho\end{aligned}$$

where x_j denotes the j'th component of $\vec{x}$, and $y_{i,j}$ denotes the j'th component of $\vec{y}_i$.

This decision function can be expressed as a weighted sum of feature values as follows:

$$\hat{c}(\vec{x}) = \sum_j w_j x_j + d$$

Algorithm 6 Interval based E-value threshold calibration

Input:
$\hat{c} : \mathbb{R}^n \to \mathbb{R}$: Classifier.
$f_1, ..., f_n : \mathbb{S}_{\mathtt{nt}} \to \mathbb{R}$: Sequence features.
$w_{size} \in \mathbb{N}$: Window width ($w \geq 1$).
$w_{step} \in \mathbb{N}$: Window step size ($s \leq w$).
$s \in \mathbb{N}$: Random sequence scaling factor.
$\lambda(G)$: Size of the genome.
$R : A...s\lambda(G) + w_{step}$: Randomly generated sequence.
m: Number of prediction states to keep track of.
$S_1, ..., S_m$: A set of prediction states.
$ClosestPredictionState(v)$: Gets the prediction state S_i with threshold closest to v.
$UpdatePredictionState(S_x, i, v)$: Updates prediction state S_x for window start coordinate i and classification value v.
$Predictions(S_x)$: Gets the number of predictions for prediction state S_x.
$Threshold(S_x)$: Gets the threshold for prediction state S_x.

$t_{last} \leftarrow 1$
$i \leftarrow A$
loop
  $j \leftarrow i + w_{size}$
  $W \leftarrow R : i...j$ ▷ Get the window.
  $v \leftarrow \hat{c}((f_1(W), ..., f_n(W)))$ ▷ Score the window.
  $t \leftarrow ClosestPredictionState(v)$
  if $t > t_{last} \vee i = A$ **then** ▷ Make necessary prediction state updates.
    for $x \in [t_{last}, t]$ **do**
      $UpdatePredictionState(S_x, i, v)$
    end for
  else if $t < t_{last}$ **then**
    for $x \in [t, t_{last}]$ **do**
      $UpdatePredictionState(S_x, i, v)$
    end for
  end if
  $t_{last} \leftarrow t$
  $i \leftarrow i + w_{step}$
  if $j \geq s\lambda(G)$ **then** ▷ Last window in the sequence, so break out of the reading loop.
    break
  end if
end loop
for $x \in [1, m]$ **do**
  $\hat{E}(Threshold(S_x)) \leftarrow \frac{1}{s} Predictions(S_x)$ ▷ Calculate the approximated E-value for each threshold.
end for

where

$$w_j = \sum_{i=1}^{l} r_j c_i y_{i,j}$$

and

$$d = \sum_j \left(\sum_{i=1}^{l} (r_j c_i y_{i,j})\, q_j \right) - \rho.$$

Note that the vector components x_j correspond to feature values, so assuming the feature sets are similar, weights w_j for PRESVM may be compared with corresponding weights assigned by the PREdictor. Thus, this gives a relationship between the PREdictor and PRESVM (with such a configuration) in terms of the decision functions.

The PREdictor calculates the feature weights by the following equation [1]

$$w_j = \ln \frac{f(p_j|\text{PRE})}{f(p_j|\text{non-PRE})} = \ln f(p_j|\text{PRE}) - \ln f(p_j|\text{non-PRE})$$

where $f(p_j|\text{PRE})$ is the number of occurrences of some motif pair p_j in a PRE sequence per kilobase, and similarly for $f(p|\text{non-PRE})$ in non-PRE sequences. For a Support Vector Machine, the actual placement of the decision surface, and thus the weights, will depend on the Support Vector Machine parameters, as well as how the training data is sampled for constructing the training vectors. This makes it difficult to compare the decision surface construction in a general manner. It may however be noted that the PREdictor could be seen as constructing one vector for each class based on finding mean vectors for each class and applying the logarithm. Outliers could thus have an impact on the PREdictor, where the SVM should be more robust. Also, it is possible that constructing the decision surface with maximal margin (as an SVM would) may improve separation of the training data and/or generalization compared to taking the logarithm of class means (as the PREdictor would).

As was noted in the first section of this chapter, PRESVM is not the first Support Vector Machine implementation of Polycomb/Trithorax Response Element prediction. Zeng *et al.* [2] used motif occurrence frequencies combined with a Support Vector Machine to predict PREs genome-wide. However, there are some important differences between their implementation, EpiPredictor, and the implementation in this thesis.

Firstly, the EpiPredictor uses the SVM for binary classification to filter out sequence windows, and for windows that are not classified as negative it assigns scores by using the total number of motif occurrences in each window [2]. A trained SVM can assign a continuous classification value to a presented instance vector, and PRESVM makes use of this to score each sequence window. It is possible that the classification value from a trained SVM may give a better model of whether or not a region corresponds to a PRE than the overall number of motif occurrences.

Secondly, although Zeng *et al.* [2] discuss the use of motif occurrence frequencies, and although this was combined with different kernels, there are other potential feature sets to consider. Multiple feature sets have been implemented in PRESVM, and combining these with a Support Vector Machine gives classifiers with different performances (see Chapter 9).

Also, there are feature sets implemented in PRESVM that make use of motif occurrence pair distances. The (j)PREdictor incorporates distances in terms of motif occurrence pairing distance cutoffs. However, additional ways of using distances are implemented in PRESVM.

Chapter 6

Sequence features

Feature sets in PRESVM are primarily based on occurrences of sequence motifs. A number of feature sets can be defined based on motifs. Feature sets for motifs are divided into motif occurrence frequency feature sets, motif occurrence distance feature sets and periodic motif occurrence frequency feature sets. Some additional feature sets will be discussed briefly.

6.1 Motif occurrence frequency features

Multiple sequence motifs have been defined for PREs with a biological justification. An alternative is to consider a set of k-mers, the set of all words of k nucleotides. When a set of motifs, M, has been defined, one natural way to define a set of features is by the number of times the motifs occur within a sequence. To formalize this, some measures need to be defined. For a sequence s, it is useful to define its set of motif occurrences and what motif each occurrence corresponds to (here referred to as the motif occurrence type).

Definition 38 (Motif occurrences) *For a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, the set $O(s)$ denotes the set of motif occurrences in s.*

Definition 39 (Motif occurrence type) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$ and a motif occurrence $o \in O(s)$, $\tau(o) \in M$ denotes the type of the motif occurrence.*

Definition 40 (Type-specific motif occurrences) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$ and motif $m \in M$, the set $O_m(s) = \{o | o \in O(s) \wedge \tau(o) = m\}$ denotes the set of motif occurrences in s of type m.*

Based on this, the following feature set can be defined.

Definition 41 (The *nOcc* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$ and motif $m \in M$, the occurrence frequency of m in s is defined as*

$$nOcc(s, m) = \frac{|O_m(s)|}{\lambda(s)}$$

This is the number of occurrences of motifs $m \in M$ in sequence s, divided by the sequence length $\lambda(s)$. This feature set has been used by Zeng *et al.* for predicting PREs with a Support Vector Machine [2]. The feature set is illustrated by Figure 6.1. Although this feature set does not directly take paired motif occurrence into account, Zeng *et al.* suggested that combining it with a Support Vector Machine with a non-linear kernel may model motif occurrence combinations [2].

Features that directly consider combined motif occurrences will be considered next. For this, the positions of and the distances between motif occurrences need to be defined.

Definition 42 (Absolute value) *For a real value $v \in \mathbb{R}$, $|v| \geq 0$ denotes its absolute value. Although the same notation is used for set cardinality, the meaning will be clear from the context.*

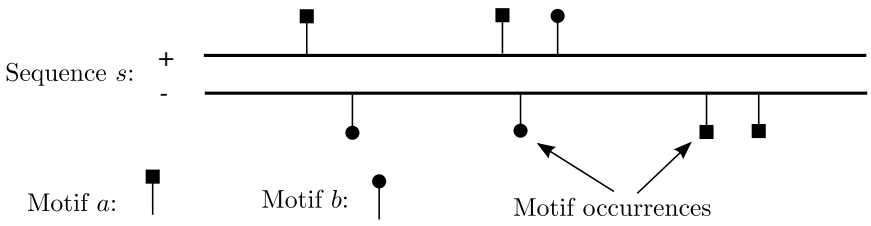


Figure 6.1: Illustration of the *nOcc* feature set. Motif occurrences are counted, and the counts are normalized by sequence length.

Definition 43 (Motif occurrence start position) *For a motif occurrence $o \in O(s)$, $\pi_\alpha(o)$ denotes the index of the first nucleotide of the motif occurrence o.*

Definition 44 (Motif occurrence end position) *For a motif occurrence $o \in O(s)$, $\pi_\beta(o)$ denotes the index of the nucleotide after the last of the motif occurrence o.*

Definition 45 (Motif occurrence center position) *For a motif occurrence $o \in O(s)$, $\pi_\gamma(o) = \frac{\pi_\alpha(o)+\pi_\beta(o)}{2}$ denotes the position of the center nucleotide of the motif occurrence o. $\pi_\gamma(o) \in \mathbb{R}$.*

Definition 46 (Motif occurrence gap distance) *For motif occurrences $o_1, o_2 \in O(s)$,*

$$\delta_\alpha(o_1, o_2) = \max\{\pi_\alpha(o_2) - \pi_\beta(o_1), \pi_\alpha(o_1) - \pi_\beta(o_2), 0\}$$

denotes the distance between motif occurrences o_1 and o_2, not counting nucleotides covered by occurrences o_1 and o_2.

Based on these measures, a measure of motif occurrence pairing can be defined, with a distance cutoff c (i.e. the pairs have this distance or less if they affect the measure).

Definition 47 (The *nPair* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, motifs $m_1, m_2 \in M$ and distance cutoff c, a pairing set can be defined as*

$$P(s, m_1, m_2, c) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\alpha(o_1, o_2) \leq c\}.$$

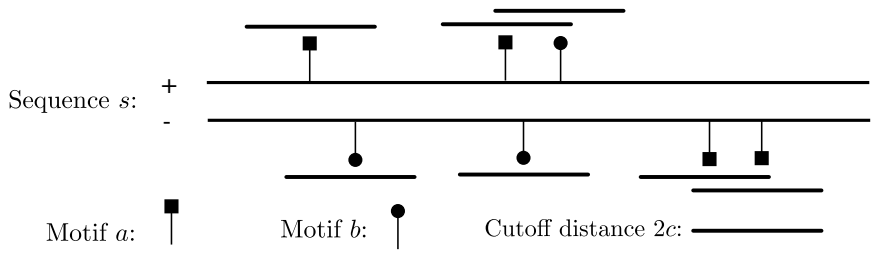
Based on this pairing set, the distance-constrained, paired motif occurrence frequency is defined as

$$nPair(s, m_1, m_2, c) = \frac{|P(s, m_1, m_2, c)|}{\lambda(s)}.$$

This feature set is similar to the feature set used by Ringrose *et al.* [1] in the PREdictor to predict PREs. It is worth noting that the $nPair(s, m_1, m_2, c)$ feature set is reflexive, in the sense that $nPair(s, m_1, m_2, c) = nPair(s, m_2, m_1, c)$. This is clear since switching m_1 and m_2 only corresponds to a reordering of terms in the pairing set, where order will not influence its cardinality. The reflexivity is expected, since Ringrose *et al.* [1] reported having 28 features for 7 motifs. This feature set is illustrated in Figure 6.2.

In addition to directly considering motif occurrence pairing, the distance cutoff c in the *nPair* feature set provides an additional, useful property. When training on full PRE sequences using the *nOcc* feature set, motif occurrence frequencies will be found over the whole sequences, and thus any interactions between the frequencies that a non-linear kernel might model would also be over the full sequences. If such interactions are local, the distance cutoff c restricts the considered combinations.

The distance measure $\delta_\alpha(o)$ is primarily used for comparison with the PREdictor. For other motif occurrence feature sets, the distance between the motif occurrence centers is used.



$$nPair(s,a,a,c) = \frac{1}{\lambda(s)}, nPair(s,b,b,c) = \frac{1}{\lambda(s)}, nPair(s,a,b,c) = \frac{3}{\lambda(s)}$$

$$nOccPair(s,a,a,c) = \frac{2}{\lambda(s)}, nOccPair(s,b,b,c) = \frac{2}{\lambda(s)}, nOccPair(s,a,b,c) = \frac{2}{\lambda(s)},$$

$$nOccPair(s,b,a,c) = \frac{3}{\lambda(s)}$$

Figure 6.2: Illustration of the *nPair* and *nOccPair* feature sets. The cutoff distance is marked as $2c$ in the figure due to being two-sided. + and - denote the strands, and pairing is independent of whether or not the occurrences are on the same or on opposite strands. For the *nPair* feature set, motif occurrence pairs within the cutoff distances are counted, and the counts are normalized by sequence length. For the *nOccPair* feature set, occurrences of the primary motif are counted if at least one occurrence of the secondary motif is within the cutoff distance, and the counts are normalized by sequence length.

Definition 48 (Motif occurrence center distance) *For motif occurrences $o_1, o_2 \in O(s)$,*

$$\delta_\gamma(o_1, o_2) = |\pi_\gamma(o_1) - \pi_\gamma(o_2)|$$

denotes the distance between the centers of motif occurrences o_1 and o_2.

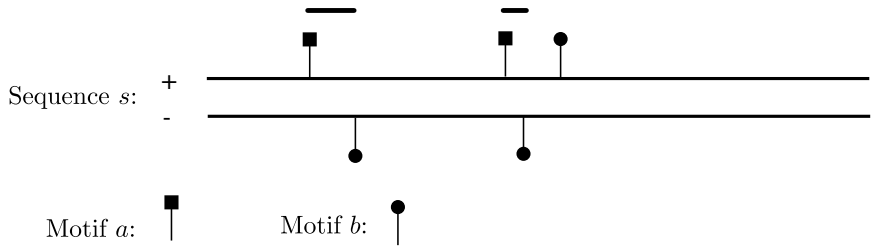
An alternative, similar feature set for motif occurrence pairing, which is not reflexive, is one that considers the frequency with which a particular motif occurs within a set distance of any occurrence of a secondary motif.

Definition 49 (The *nOccPair* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, motifs $m_1, m_2 \in M$ and distance cutoff c, the nOccPair feature is defined as*

$$nOccPair(s, m_1, m_2, c) = \frac{|\{(o_1, m_2) | o_1 \in O_{m_1}(s) \wedge \exists o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c\}|}{\lambda(s)}$$

For this feature set, a motif occurrence of the first type is paired with the secondary motif, rather than with each occurrence of the secondary motif, influencing pair uniqueness. To demonstrate its non-reflexiveness, imagine there are five occurrences of m_a and one of m_b, all within the cutoff distance c from each other. For m_a as parameter m_1 and m_b as m_2, the set will contain five elements, as all five occurrences of m_a are within the cutoff distance from the occurrence of m_b. However, if m_b is taken to be parameter m_1 and m_a as m_2, the set will only contain one element, as only the one occurrence of m_b is counted. This is illustrated in Figure 6.2.

A potentially useful property of the *nOccPair* feature set is that it in addition to pairing motif occurrences also should capture the contribution of each motif. A drawback is that there will be more features, and this may in turn increase the computational cost. The larger number of features may also require a larger training set.



$$MDP(s,a,b) = \frac{10+5}{2} = 7.5$$

Figure 6.3: Illustration of the MDP feature set. The bold lines are distances to the closest motif occurrence of b from each occurrence of a. Assuming that the left-most line is of length 10 and the right-most is of length 5 gives the above feature value.

6.2 Motif occurrence distance features

A further possibility is that there is a trend towards particular distances between motif occurrences. This might vary for different motif combinations. The first such feature that is considered in this thesis makes use of proximal motif occurrence distances. The mean distance from occurrences of motif m_1 to the closest occurrences of m_2 is called Mean Distance Proximal (MDP) in PRESVM.

Definition 50 (Set element exclusion) *The set element exclusion operation is in this thesis defined as $X/e = \{x|x \in X \wedge x \neq e\}$. Thus, it excludes the element e if it is a member of X, or otherwise gives X.*

Definition 51 (The MDP feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, a window size w and motifs $m_1, m_2 \in M$, the MDP feature is defined as*

$$dp(s,m_1,m_2,w) = \left\{(d,o_1)|o_1 \in O_{m_1}(s) \wedge d = \min_{o_2 \in O_{m_2}(s)/o_1} \delta_\gamma(o_1,o_2) \wedge d < w\right\}$$

$$MDP(s,m_1,m_2,w) = \frac{1}{|dp(s,m_1,m_2,w)|} \sum_{(d,o_1)\in dp(s,m_1,m_2,w)} d.$$

If $dp(s,m_1,m_2,w) = \emptyset$, then to avoid MDP becoming undefined, $MDP(s,m_1,m_2,w) = w$.

The condition in the definition to avoid MDP becoming undefined uses the window size. The window size was chosen assuming that there are motif occurrences outside the window. The window size is also used as a distance cutoff, such that only local relative distances influence the features. The feature set is illustrated in Figure 6.3.

If this feature set is combined with a linear kernel, it can be expected that most feature values will be too common to be predictive. If it is combined with a non-linear kernel, feature values may be modelled more specifically. However, this feature set does not include motif occurrence frequencies, and it may be necessary to combine it with a motif occurrence frequency feature set to get satisfactory classification performance.

A similar feature set can be defined, where the mean distance from occurrences of motif m_1 to the closest occurrences of any motif is used. This is called Mean Distance Proximal Any type (MDPA) in PRESVM.

Definition 52 (The $MDPA$ feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, a window size w and motif $m \in M$, the $MDPA$ feature is defined as*

$$dpa(s,m,w) = \left\{(d,o_1)|o_1 \in O_m(s) \wedge d = \min_{o_2 \in O(s)/o_1} \delta_\gamma(o_1,o_2) \wedge d < w\right\}$$

$$MDPA(s,m,w) = \frac{1}{|dpa(s,m,w)|} \sum_{(d,o_1)\in dpa(s,m,w)} d.$$

If $dpa(s,m,w) = \emptyset$, then to avoid $MDPA$ becoming undefined, $MDPA(s,m,w) = w$.

The reason why this feature set could be useful is that it might be that occurrences of any type need to have a given distance from the considered motif in an actual PRE. The smaller number of features may further reduce the risk of fitting unimportant variations in the training data.

Two additional feature sets can be constructed by changing the min in the formulae for MDP and $MDPA$ to max, giving the Mean Distance Distal (MDD) and Mean Distance Distal Any type (MDDA) feature sets.

Definition 53 (The MDD feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, a window size w and motifs $m_1, m_2 \in M$, the MDD feature is defined as*

$$dd(s,m_1,m_2,w) = \left\{(d,o_1)|o_1 \in O_{m_1}(s) \wedge d = \max_{o_2 \in O_{m_2}(s)/o_1} \delta_\gamma(o_1,o_2) \wedge d < w\right\}$$

$$MDD(s,m_1,m_2,w) = \frac{1}{|dd(s,m_1,m_2,w)|} \sum_{(d,o_1)\in dd(s,m_1,m_2,w)} d.$$

If $dd(s,m_1,m_2,w) = \emptyset$, then to avoid MDD becoming undefined, $MDD(s,m_1,m_2,w) = w$.

Definition 54 (The $MDDA$ feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, a window size w and motif $m \in M$, the $MDDA$ feature is defined as*

$$dda(s,m,w) = \left\{(d,o_1)|o_1 \in O_m(s) \wedge d = \max_{o_2 \in O(s)/o_1} \delta_\gamma(o_1,o_2) \wedge d < w\right\}$$

$$MDDA(s,m,w) = \frac{1}{|dda(s,m,w)|} \sum_{(d,o_1)\in dda(s,m,w)} d.$$

If $dda(s,m,w) = \emptyset$, then to avoid $MDDA$ becoming undefined, $MDDA(s,m,w) = w$.

The mean distance from occurrences of one motif to occurrences of a second motif is called Mean Distance Mean (MDM) in PRESVM.

Definition 55 (The MDM feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, a window size w and motifs $m_1, m_2 \in M$, the MDM feature is defined as*

$$dm(s,m_1,m_2,w) = \{(d,o_1,o_2)|o_1 \in O_{m_1}(s) \wedge d = \delta_\gamma(o_1,o_2) \wedge d < w\}$$

$$MDM(s,m_1,m_2,w) = \frac{1}{|dm(s,m_1,m_2,w)|} \sum_{(d,o_1,o_2)\in dm(s,m_1,m_2,w)} d.$$

If $dm(s,m_1,m_2,w) = \emptyset$, then to avoid MDM becoming undefined, $MDM(s,m_1,m_2,w) = w$.

A potential issue with these feature sets is that they consider mean distances, but they do not take the variations in the distances into account. Thus, the mean might be the same, but the distancing different. One might attempt to augment these features with features giving the corresponding variances or standard deviations. This has been considered, but has not been explored in this thesis.

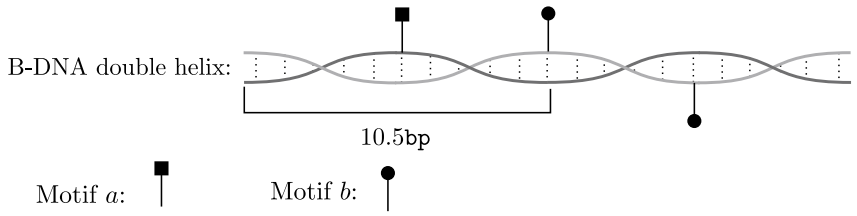


Figure 6.4: Illustration of periodic motif occurrences. The left-most occurrences of a and b would have centers almost in the same direction relative to the double helix axis. The two occurrences of b, however, face almost opposite directions.

6.3 Periodic motif occurrence frequency features

Considering DNA as a double-helix, for B-DNA, there are approximately 10.5bp per helix turn [3]. Thus, shifting a motif occurrence by 5.25bp in either direction would make it face in the opposite direction relative to the double helix axis. This is illustrated in Figure 6.4.

There are multiple ways to design feature sets to take this into account. One simple feature set uses paired motif occurrences, where the pairs are weighted by a cosine curve. The curve is scaled to be non-negative, so as to count pairs on the peaks. This is called the PEriodic DIstance (PEDI) feature set.

Definition 56 (The *PEDI* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, motifs $m_1, m_2 \in M$ and periodicity ω, the $PEDI$ feature is defined as*

$$PEDI(s, m_1, m_2, \omega, c) = \frac{1}{2\lambda(s)} \sum_{\{o_1,o_2\} \in p(s,m_1,m_2,c)} \left(\cos\left(\frac{\delta_\gamma(o_1, o_2)}{\omega} 2\pi \right) + 1 \right),$$

where

$$p(s, m_1, m_2, c) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c\}.$$

For an expanded version of this feature set, pairs are weighted with a periodicity of 10.5bp (for B-DNA), where if the occurrences are on opposite strands, the phase is shifted by 5.25bp. This is called the nPairDH (nPair, Double Helix) feature set.

Definition 57 (The *nPairDH* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, and motifs $m_1, m_2 \in M$, the $nPairDH$ feature is defined as*

$$nPairDH(s, m_1, m_2, c) = \frac{1}{2\lambda(s)} \sum_{\{o_1,o_2\} \in p(s,m_1,m_2,c)} \left(\cos\left(\frac{\delta_\gamma(o_1, o_2) + 5.25 s(o_1, o_2)}{10.5} 2\pi \right) + 1 \right),$$

where

$$p(s, m_1, m_2, c) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c\},$$

and $s(o_1, o_2) = 0$ if o_1 and o_2 are on the strand, and $s(o_1, o_2) = 1$ otherwise.

If the distances between the motif occurrence centers might indicate whether or not factors bound to the motifs might be able to interact, then the $nPairDH$ feature set might model this.

The differences between the $PEDI$ and $nPairDH$ feature sets are illustrated by Figure 6.5. A potential issue with these feature sets is that the distances between motif occurrence centers might not give the correct curve phases. When considering the double helix, motif occurrence pairs can be considered to give a relative

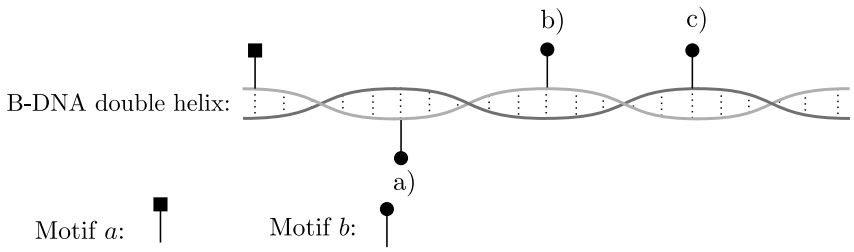


Figure 6.5: Illustration of the differences between the $PEDI$ and $nPairDH$ feature sets. Assume that all occurrences of b are within the cutoff distance from the occurrence of a. For a), both $PEDI(s, a, b, 10.5, c)$ and $nPairDH(s, a, b, c)$ will weight pairing the occurrence of a with it around zero. For b), both will weight pairing with it around one. For c), $nPairDH(s, a, b, c)$ will weight the pairing around one, whereas $PEDI(s, a, b, 10.5, c)$ will weight pairing with it around zero.

direction/phase. It may be that some relative phases for pairs are more predictive of PREs than others. Without prior knowledge of what such optimal phases may be, it may instead be attempted to learn phases from the training data.

One approach is to create a feature set where there are multiple curves with shifted phases for each motif pair. For such a pairing, the motif occurrence pair distance should be signed. To make the feature value the same when applied to the reverse complement sequence, the phase is reversed when the motif occurrence of the first type is on the minus strand. For occurrence pairs on opposite strands, one could try shifting the phases half way. However, to avoid making the assumption that pairing would be the same on the same strand as on opposite strands, the curves are instead duplicated. Thus, for each motif pair, there will be $2 * r$ features, where r is the number of curves, here called the resolution.

Definition 58 (The *nPairCosI* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, motifs $m_1, m_2 \in M$, and periodicity ω, the nPairCosI feature is defined as*

$$nPairCosI(s, m_1, m_2, i, r, q, \omega, c) = \frac{1}{2\lambda(s)} \sum_{\{o_1,o_2\} \in p(s,m_1,m_2,c,q)} PairCosI(o_1, o_2, i, r, \omega),$$

where

$$PairCosI(o_1, o_2, i, r, \omega) = \cos\left(2\pi\left(\frac{\sigma(o_1)(\pi_\gamma(o_2) - \pi_\gamma(o_1))}{\omega} + \frac{i}{r}\right)\right) + 1,$$

$\sigma(o_1) = 1$ *if o_1 is on the plus strand, or* $\sigma(o_1) = -1$ *if it is on the minus strand, and*

$$p(s, m_1, m_2, c, 1) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c \wedge \sigma(o_1) = \sigma(o_2)\},$$

$$p(s, m_1, m_2, c, 2) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c \wedge \sigma(o_1) \neq \sigma(o_2)\}.$$

For this feature set, q specifies strandedness and i specifies the curve index. This feature set can give many features per motif pair, so it can be more computationally expensive than for example the $nPairDH$ feature set, and it can require a larger training set. However, assuming that primarily certain phases are informative, it can be combined with feature selection.

An alternative approach is to think of the $nPair$ feature set as summing up 1's for pairing motif occurrences. For a particular motif occurrence pair, the relative direction around the double helix based on the distance measure can be represented in two dimensions as a vector, as with the arms of a clock, by having the x-axis correspond to the cosine of the direction and the y-axis correspond to the sine of the direction. Such vectors can be summed up. To get the same feature value on the reverse complement of a sequence, the direction should be reversed if the occurrence of the first motif type considered is on the minus strand.

For occurrences on opposite strands, one could shift the phase by half the periodicity. However, as with $nPairCosI$, this is avoided here, by summing direction vectors separately for occurrences on the same strand and on opposite strands. Thus, this feature set has 4 features per motif pair.

Definition 59 (The *nPair2D* feature set) *For a set of motifs M, a sequence $s \in \mathbb{S}_{\mathtt{nt}}$, and motifs $m_1, m_2 \in M$, where $m_1 \neq m_2$, the $nPair2D$ feature is defined as*

$$nPair2D_x(s, m_1, m_2, q, c) = \frac{1}{\lambda(s)} \sum_{\{o_1,o_2\} \in p(s,m_1,m_2,c,q)} \cos\left(\frac{2\pi\sigma(o_1)(\pi_\gamma(o_2) - \pi_\gamma(o_1))}{10.5}\right),$$

$$nPair2D_y(s, m_1, m_2, q, c) = \frac{1}{\lambda(s)} \sum_{\{o_1,o_2\} \in p(s,m_1,m_2,c,q)} \sin\left(\frac{2\pi\sigma(o_1)(\pi_\gamma(o_2) - \pi_\gamma(o_1))}{10.5}\right),$$

where $\sigma(o_1) = 1$ if o_1 is on the plus strand, or $\sigma(o_1) = -1$ if it is on the minus strand, and

$$p(s, m_1, m_2, c, 1) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c \wedge \sigma(o_1) = \sigma(o_2)\},$$

$$p(s, m_1, m_2, c, 2) = \{\{o_1, o_2\} | o_1 \in O_{m_1}(s) \wedge o_2 \in O_{m_2}(s) \wedge o_1 \neq o_2 \wedge \delta_\gamma(o_1, o_2) \leq c \wedge \sigma(o_1) \neq \sigma(o_2)\}.$$

For the case where $m_1 = m_2$, the same feature may be used but with $2\pi\sigma(o_1)(\pi_\gamma(o_2) - \pi_\gamma(o_1))$ replaced with $2\pi|\pi_\gamma(o_2) - \pi_\gamma(o_1)|$.

The distinction between the cases where $m_1 = m_2$ and $m_1 \neq m_2$ is due to one occurrence not being a clear reference for the phase when the types are the same. It can be noted that for motifs m_a and m_b and a same/opposite strand $nPair2D$ feature, if all pairs of occurrences of motifs have the same pairing phase, then the length of the resulting vector will be the corresponding motif pair occurrence frequency. Also, motif occurrences with very variable phases could cancel each other out.

6.4 Other motif occurrence features

One interesting potential feature set would be one in which actual motif occurrence positions are used in such a way that the number of features is minimized, but the positional distribution of motifs in PREs could be modelled, and the integration of this information would be left up to the training of the Support Vector Machine. One potential way of doing this might be to use a sliding window when generating training vectors, to use one side of the window as a positional reference, and to find the mean distance of occurrences of each motif from this reference position. Additionally, the deviation from the mean position could be used to model the spreading of motif occurrences.

A problem comes if one wishes for this to generalize to the reverse complement of the sequence. For the reverse complement, the other side of the window will correspond to the positional reference point. This could be solved by also training on the reverse complement of the sequence. Alternatively, one might try to establish some landmark by which the strandedness could be checked. Unfortunately, it is not clear what a good landmark would be, and training on twice the amount of windows would computationally be considerably more expensive, as well as potentially increase the amount of noise. This has been considered, but not implemented.

Another potential feature set could be constructed by training a separate classifier on motif pairs themselves, including features such as distances, and sum the output values to generate final features. This could give an indication of how typical each individual pair is of a PRE versus non-PRE. This has been considered, but not been implemented.

6.5 Other sequence features

The PRE prediction method developed by Zeng *et al.* [2], the EpiPredictor, makes use of the GC content in addition to sequence motifs. The GC content is the ratio of nucleotides in the sequence that are `G` or `C`.

$$GC(s) = \frac{1}{\lambda(s)} \sum_{i=1}^{\lambda(s)} GC_{\mathtt{nt}}(s_i)$$

where $GC_{\mathtt{nt}}(s_i)$ for nucleotide s_i is 1 if s_i is `G` or `C`. The EpiPredictor uses GC content for filtering predictions. It applies the rule that PREs should have a GC content of at least 44% to be predicted. In PRESVM, GC content is implemented as a feature.

Chapter 7

Configuration optimization

A particular machine learning method may have parameters that need to be set by the user. There may also be multiple potential feature sets, where not all features would be relevant. Depending on how many parameters there are and whether they are continuous or discrete, the search space may be too large for it to be practical to search manually for optimal parameter values. This chapter explores automated approaches to searching for optimal parameter values by optimizing a classifier quality measure. The chapter concludes with a brief discussion of feature selection.

7.1 Configuration vectors

If the parameters are real values, a vector space can be defined, here referred to as the configuration space, where each vector corresponds to a configuration, and each vector component corresponds to a parameter. Finding an optimal configuration then corresponds to searching in this space using some measure of quality as a guidance.

In the case of Support Vector Machines and LibSVM, the number of parameters to set depends on the kernel and SVM formulation used. If, for example, the ϵ-SVR formulation is used with a cubic kernel (polynomial with parameter $d = 3$), one needs to set C, γ, c_0 and ϵ, all of which are continuous. However, these parameters have different ranges. Also, it may vary what sequence of values would be worth testing. In the case of C and γ, it has been suggested that parameter values may be tested growing exponentially [23]. For the rest of the parameters, linearly growing parameter values have been considered in PRESVM.

Configuration vectors can be transformed to vectors in another space, here referred to as a unit space, where values can be tested in linear increments. Any additional range restrictions can be applied when transforming vectors back from the unit space to the configuration space. It is also useful to have the transformations between the configuration space and unit space scaled, such that values in the unit space can be tested with the same stepping in all dimensions.

7.2 Grid search

A hyper-dimensional grid can be defined in the unit space. To do this, a set of evenly spaced values $P_i \subset \mathbb{R}$ is chosen for each parameter i. Vectors for configurations are then made by combining these values into vectors. These configuration vectors constitute a set $P = P_1 \times ... \times P_n$. If each such configuration is compared, this forms a grid search.

One way to implement this search is to nest loops, with one loop for each parameter. In each iteration of the innermost loop, one tests how well the classifier performs with the configuration. The best configuration

observed so far is kept track of, and the final best observed configuration gives the found optimum. Another way of implementing this search is to keep one count c_i for each parameter i. The counter c_1 is incremented in each iteration, and when it exceeds its limit, it is reset and c_2 is incremented. If c_2 exceeds its limit, it is reset and c_3 is incremented, etc. The search then ends when the final parameter exceeds its limit. This formulation of the search generalizes to a variable number of parameters, and is illustrated in Algorithm 7.

Algorithm 7 Grid search, generalized

Input:
$Q(\vec{x})$: Gives the quality of configuration vector $\vec{x}$.
n: Number of configuration parameters.
$\vec{c}(i_1, ..., i_n)$: Gives a configuration vector for indices $i_1, ..., i_n$.
$m_1, ..., m_n$: The maximum indices for each parameter.

```
i_1, ..., i_n ← 0                                  ▷ Initialize the indices.
c⃗_best ← 0                                         ▷ Best observed configuration.
q_best ← −∞                                        ▷ Quality of best observed configuration.
loop
    c⃗_c ← c⃗(i_1, ..., i_n)                         ▷ Get configuration ...
    q_c ← Q(c⃗_c)                                   ▷ ... and quality of it.
    if q_c > q_best then                           ▷ If best observed so far, note it.
        q_best ← q_c
        c⃗_best ← c⃗_c
    end if
    for x ∈ [1, n] do                              ▷ Update indices.
        i_x ← i_x + 1
        if i_x ≤ m_x then                          ▷ If it did not exceed the index maximum ...
            break                                  ▷ ... then stop incrementing.
        else
            if x ≠ n then                          ▷ Reached the maximum, but this was not the last index ...
                i_x ← 0                            ▷ ... so reset it and continue incrementing.
            else                                   ▷ Reached the maximum of the last index ...
                return c⃗_best                      ▷ ... so return the best configuration that was found.
            end if
        end if
    end for
end loop
```

A tool for grid search written in Python is included in LibSVM version 3.11, and grid search has been recommended by the LibSVM authors [23]. An internal implementation of grid search has been made for PRESVM.

A drawback of the grid search approach to selecting parameters is that as the number of parameters increase, it can become computationally very expensive. For example, if there are four parameters and one wants to check 100 values for each parameter, one needs to check $100^4 = 100,000,000$ configurations. The LibSVM authors, however, suggest that one starts with a coarse grid, and then iteratively zoom in on the grid around the best region found [23]. This has not been implemented in PRESVM.

7.3 Approximated gradient search

For a particular configuration, $\vec{p} = (p_1, ..., p_n)$, small changes in each parameter may affect classifier performance. Because of this, a greedy search can be performed by treating a configuration as a moving point,

measuring the classifier performance for small variations of the parameters, and moving towards more optimal configurations. Random parameter values can be selected for the starting configuration. To perform this search, an extra axis can be added for classification error, where the configuration is attracted to points of lower error by moving along the slope of the error surface (corresponding to gravity along the error axis). This leads to gradient descent.

If one has a function $E(\vec{p})$ that gives the error when using configuration $\vec{p}$, the error gradient is defined as [22]

$$\nabla E(\vec{p}) = \left(\frac{\partial E}{\partial p_1}, ..., \frac{\partial E}{\partial p_n} \right).$$

Gradient descent can then be defined by the rule [22]

$$\vec{p}_{i+1} = \vec{p}_i - \eta \nabla E(\vec{p}_i),$$

where $\vec{p}_i$ is the current configuration and η is a learning rate.

The formulations of Support Vector Machines are complex, and it is easier to try to estimate the gradient than it is to derive it mathematically. The partial derivative takes the derivative with respect to the specified variable and treats the other variables as constants. The partial derivative thus gives a tangent line for one parameter. The tangent of a continuous function $f(x)$ can be formally defined as [33]

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

Instead of using the actual derivative, the limit, an approximation may be attempted by choosing a small value for Δx. Doing this for every parameter p_i gives an approximation of the gradient:

$$\widehat{\nabla E}(\vec{p}) = \left(\frac{E(\vec{p} + \vec{v}_1 \Delta p) - E(\vec{p})}{\Delta p}, ..., \frac{E(\vec{p} + \vec{v}_n \Delta p) - E(\vec{p})}{\Delta p} \right),$$

where $\vec{v}_1 = (1, 0, ..., 0, 0)$ and $\vec{v}_n = (0, 0, ..., 0, 1)$.

This is implemented in PRESVM, with the change that the gradient is approximated based on both positive and negative parameter value changes. The reason for doing this is the assumption that a small step in the positive direction may not have as big of an impact as a small step in the negative direction, or vice versa.

$$\widehat{\nabla E}(\vec{p}) = \left(\frac{E(\vec{p} + \vec{v}_1 \Delta p) - E(\vec{p} - \vec{v}_1 \Delta p)}{2\Delta p}, ..., \frac{E(\vec{p} + \vec{v}_n \Delta p) - E(\vec{p} - \vec{v}_n \Delta p)}{2\Delta p} \right),$$

Also, in PRESVM a quality measure is used instead of an error function. Thus, the update rule becomes gradient ascent, defined as

$$\vec{p}_{i+1} = \vec{p}_i + \eta \widehat{\nabla Q}(\vec{p}_i),$$

for quality measure Q. Additionally, given the potential for very steep approximated gradients, PRESVM incorporates a terminal velocity to reduce speeds along the quality gradient. This is illustrated in Algorithm 8.

There are multiple drawbacks to this approach. First, the approach to estimating the gradient uses a specified Δp, and the approach is not guaranteed to give the correct tangents based on Δp. Second, a quality measure used should be differentiable. Measures such as *Accuracy* and $AUC(x)$ can still be used if Δp is large enough to significantly affect scores for incorrectly classified instances, but otherwise it is possible that these measures will not be affected by the parameter changes. Third, the search is greedy, and the randomly selected starting point may largely influence how good the optimum found is, depending on whether or not there are local optima. Fourth, it may require many training cycles due to the approach to estimation. However, if the used performance measure is continuous, small changes in parameters have a positive effect on classification performance and the starting point is in proximity of the global optimum, this procedure enables fine tuning of these parameters.

Algorithm 8 Approximated gradient search

Input:
$Q(\vec{x})$: Gives the quality of configuration vector $\vec{x}$.
n: Number of configuration parameters.
$\vec{r}()$: Gives a random vector of dimensionality n.
Δp: A small increment to use for tangent approximation.
$\Delta(i)$: Gives a vector in n dimensions with component i equal to Δp, and all others as zero.
l: Number of iterations to run.
η: Learning rate.
$v_{terminal}$: Terminal velocity for gradient approximation.

$\vec{c} \leftarrow \vec{r}()$
$\widehat{\nabla Q} = (\widehat{Q}_1, ..., \widehat{Q}_n)$ ▷ Gradient approximation
$\vec{c}_{best} \leftarrow 0$ ▷ Best observed configuration.
$q_{best} \leftarrow -\infty$ ▷ Quality of best observed configuration.
for $i \in [1, l]$ **do**
  for $x \in [1, n]$ **do** ▷ Find gradient approximation components
    $\widehat{\nabla Q}_x \leftarrow \eta \frac{Q(\vec{c}+\Delta(x)) - Q(\vec{c}-\Delta(x))}{2\Delta p}$
  end for
  if $|\widehat{\nabla Q}| > v_{terminal}$ **then** ▷ Restrict velocity.
    $\widehat{\nabla Q} \leftarrow \widehat{\nabla Q} * \frac{v_{terminal}}{|\widehat{\nabla Q}|}$
  end if
  $\vec{c} \leftarrow \vec{c} + \widehat{\nabla Q}$
  $q \leftarrow Q(\vec{c})$
  if $q > q_{best}$ **then**
    $q_{best} \leftarrow q$
    $\vec{c}_{best} \leftarrow \vec{c}$
  end if
end for
return $\vec{c}_{best}$

7.4 Particle Swarm Optimization

Another way to optimize parameters is to treat multiple parameters as moving vectors. The technique of Particle Swarm Optimization (PSO) [34] uses a set of m points ("particles") $\vec{p}_1, ..., \vec{p}_m$ starting at random positions, each with an initially random velocity $\Delta\vec{p}_i$ and best observed position $\vec{P}_i$ (i.e. the observed position of the particle $\vec{p}_i$ where the measured quality was largest). The overall best observed position $\vec{P}_\forall$ is also kept track of.

In each iteration, the velocity of each vector is updated according to the rule [34]

$$\Delta p_{id}^j = \Delta p_{id}^{j-1} + c_1 r_1 \left(P_i^{j-1} - p_{id}^{j-1} \right) + c_2 r_2 \left(P_{i\forall}^{j-1} - p_{id}^{j-1} \right),$$

where i denotes the particle index, d denotes the vector component and j denotes the iteration. c_1 determines the attraction towards an earlier observed optimum by the particle, called the cognition learning rate [34]. c_2 determines the attraction towards the globally observed optimum, called the social learning rate [34]. r_1 and r_2 are randomly generated, real values from 0 to 1, and these values are generated anew in each iteration and for each component.

The vector update rule is given by [34]

$$\vec{p}_i^{\,j+1} = \vec{p}_i^{\,j} + \Delta\vec{p}_i$$

where i denotes the particle index and j denotes the iteration.

This method has been reported to give favourable classification accuracy [34] and has been implemented in PRESVM (see Algorithm 9). The randomization can be expected to lead to variable results, and it may not be obvious what parameter values of the method will lead to the best results. However, given randomized initial particle positions and velocities, it is possible that the particles will spread across the search space and test a large range of parameter values. Also, only one quality measurement is needed for each particle in each iteration.

7.5 Feature selection

There may be many potential features to use, and it may not be obvious which features will be relevant for a given prediction task. Using many potentially redundant features may be problematic, increasing the computational cost without a return in classification accuracy. There is also a possibility that using redundant features adversely affects the quality of the resulting classifier. Feature selection uses a base set of features, and tries to determine a subset of relevant features for the prediction task.

As with searching for parameters, features may be selected by various methods. One approach is to consider a boolean vector, where each component corresponds to whether or not a feature is used. A greedy search might iteratively try to add one feature at a time, choosing the best combination found. This can be time consuming, and it could reach a locally optimal solution. Also, if this is based on measuring classification quality, SVM parameters will need to be selected prior to feature selection or in each iteration.

Alternatively, feature selection can be incorporated into the search for optimal SVM parameters. Feature selection combined with SVM parameter search has been attempted for Particle Swarm Optimization, where favourable results were reported [34]. This has not been implemented in PRESVM.

Alternatively, only the feature values may be considered when choosing a feature set. Minimum Redundancy, Maximum Relevance (mRMR) feature selection uses information theoretical properties of the feature values to select an optimal feature set of a requested size. This approach to feature selection has been combined with Support Vector Machines and bioinformatics, albeit in a different field, but with favourable results

Algorithm 9 Particle Swarm Optimization

Input:
d: Dimensionality of the configuration space.
n: Number of iterations the Particle Swarm Optimization should run.
m: The number of particles.
$Q(\vec{c})$: Gives the quality of configuration $\vec{c}$.
$\vec{r}()$: Gives a random vector of dimensionality d with value ranges $[0, 1]$. Each call to $\vec{r}()$ gives a new random vector.
$\vec{a} * \vec{b}$: Makes a vector with components being products of the components of $\vec{a}$ with those of $\vec{b}$.
c_1: Cognition learning rate.
c_2: Social learning rate.

$\vec{p}_1, ..., \vec{p}_m$ ▷ Particle positions
$\Delta\vec{p}_1, ..., \Delta\vec{p}_m$ ▷ Particle velocities
$\vec{P}_1, ..., \vec{P}_m$ ▷ Best observed positions for the particles
$Q_1, ..., Q_m$ ▷ Quality of the best observed particle positions
$\vec{P}_\forall$ ▷ Best observed position overall
$Q_\forall = -\infty$ ▷ Quality of best observed position overall
for all $j \in [1, m]$ **do** ▷ Initialize particles
  $\vec{p}_j \leftarrow (\vec{r}() - \vec{r}()) * 8$
  $\Delta\vec{p}_j \leftarrow \vec{r}() - \vec{r}()$
  $\vec{P}_j \leftarrow \vec{p}_j$
  $Q_j \leftarrow Q(\vec{P}_j)$
  if $Q_j > Q_\forall$ **then**
    $Q_\forall \leftarrow Q_j$
    $\vec{P}_\forall \leftarrow \vec{P}_j$
  end if
end for
for all $i \in [1, n]$ **do** ▷ Run through cycles
  for all $j \in [1, m]$ **do** ▷ Update particles
    $\Delta\vec{p}_j \leftarrow \Delta\vec{p}_j + c_1\vec{r}() * (\vec{P}_j - \vec{p}_j) + c_2\vec{r}() * (\vec{P}_\forall - \vec{p}_j)$ ▷ Update velocity
    $\vec{p}_j \leftarrow \vec{p}_j + \Delta\vec{p}_j$ ▷ Move the particle
    $q \leftarrow Q(\vec{p}_j)$ ▷ Measure quality
    if $q > Q_j$ **then** ▷ Update best observed position for the particle
      $Q_j \leftarrow q$
      $\vec{P}_j \leftarrow \vec{p}_j$
      if $q > Q_\forall$ **then** ▷ Update best overall observed position
        $Q_\forall \leftarrow q$
        $\vec{P}_\forall \leftarrow \vec{P}_j$
      end if
    end if
  end for
end for
return $\vec{P}_\forall$

[30]. mRMR feature selection is supported in PRESVM.

Chapter 8

Validation of Polycomb/Trithorax Response Element predictions

After having predicted PREs genome-wide, it is desirable to get an idea of how good the predictions are. To this end, a dedicated software application has been implemented, called PREsent. This will be discussed in this chapter.

After threshold calibration, PRESVM outputs a table of an approximated score E-value distribution. During genome-wide prediction, PRESVM outputs a score profile and predicted PRE coordinates for each chromosome. An approximated E-value distribution, score profiles and predicted PRE coordinates can also be output by jPREdictor, enabling comparisons. For investigating the quality of predictions, it makes sense to compare them with experimental data from genome-wide studies. Experimental data investigated in the work with this thesis takes three forms: 1) Experimentally determined PREs, 2) PcG target genes and 3) score profiles and PcG/TrxG protein and histone methylation enrichment profiles.

8.1 Validating predicted PRE regions

For comparing predicted PREs with experimentally determined regions, their overlaps can be considered. If lengths of experimentally determined regions and predictions vary, considering the size in base pairs of each overlap could bias measurements, and so it seems reasonable to only consider if there is any or no overlap between each pair of regions. Confusion matrix values can then be defined in terms of the overlaps.

Definition 60 (Genomic region) *For a genome sequence G, a chromosome will be taken to be a sequence $C \in \mathbb{S}_{\mathtt{nt}}$, such that $C : A...B$ refers to a region in C from A up to and including B.*

Definition 61 (Region overlap function) *For regions $r_1 = C_1 : A_1...B_1$ and $r_2 = C_2 : A_2...B_2$, $o(r_1, r_2)$ denotes the overlap function. It is equal to 1 when r_1 and r_2 overlap, or 0 otherwise.*

$$o(r_1, r_2) = \begin{cases} 0 & if C_1 \neq C_2 \vee B_1 < A_2 \vee B_2 < A_1 \\ 1 & otherwise \end{cases}$$

Consider two sets of regions, R_1 and R_2. *Sensitivity* is defined as TP/P, where P is the total number of positives and TP the portion of these that are predicted as positives. Taking R_1 as a set of predictions and R_2 as a set of positives, it makes sense to take $P = |R_2|$ as the number of positives, and TP to be the number of regions in R_2 that overlap with one or more regions in R_1 (the maximum *Sensitivity* should be 1).

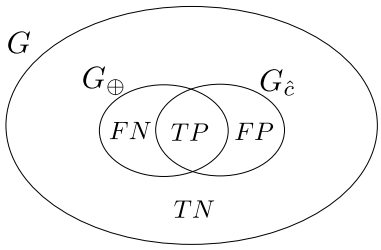


Figure 8.1: Illustration of the gene set overlaps for constructing a confusion matrix. G is the set of all annotated genes, $G_\oplus$ is the set of validation genes and $G_{\hat{c}}$ is the set of predicted target genes. TP, TN, FP and FN are confusion matrix values.

Definition 62 (Region Set Sensitivity) *For sets $R_1 = \{C_i : A_i...B_i\}$ and $R_2 = \{C_j : A_j...B_j\}$ of regions, the sensitivity of R_1 to R_2 is defined as*

$$Sensitivity(R_1, R_2) = \frac{|\{r_2 | r_1 \in R_1 \wedge r_2 \in R_2 \wedge o(r_1, r_2) = 1\}|}{|R_2|}.$$

Similarly, $PPV = TP/(TP+FP) = TP/P'$, where P' is the number of predictions. For a set of predicted regions R_1, and a set of positive regions R_2, it thus makes sense to let $P' = |R_1|$ and to let TP be the number of regions in R_1 that overlap with regions in R_2 (because the maximum PPV should be 1).

Definition 63 (Region Set Positive Predictive Value) *For sets $R_1 = \{C_i : A_i...B_i\}$ and $R_2 = \{C_j : A_j...B_j\}$ of regions, the Positive Predictive Value of R_1 to R_2 is defined as*

$$PPV(R_1, R_2) = \frac{|\{r_1 | r_1 \in R_1 \wedge r_2 \in R_2 \wedge o(r_1, r_2) = 1\}|}{|R_1|}$$

The number of regions from one set that overlap with regions in another set is used to define TP, and the roles of the sets are opposite for Region Set Sensitivity and Region Set Positive Predictive Value, leading to two definitions of TP due to the potential of multiple overlaps with single regions. Having defined *Sensitivity* and PPV for sets of regions, which use TP, FP and FN (albeit here with two definitions of TP), what is missing to form a confusion matrix is TN. Although TN could be defined as all possible regions in the genome that neither overlap with predictions nor with positives, different potential lengths of regions would likely make this number difficult to calculate and too large to be informative.

However, performance measurements may still be made without defining TN. Given that thresholds of considered classifiers have been calibrated for the same E-value, the relative differences in *Sensitivity* and PPV should still be informative. Given that *Sensitivity* and PPV have been defined, the F_1 measure can be calculated and *Precision/Recall* curves can be constructed.

8.2 Predicting and validating PcG target genes

For the experimental data sets considered in this thesis, PcG target genes have been given [12, 14, 15]. If all annotated genes are taken to be in a set G, the genes for a given validation set can be collected in a subset $G_\oplus \subset G$. If, furthermore, PcG target genes are predicted from predicted PREs and collected in a set $G_{\hat{c}} \subset G$, the sets can be used to construct a confusion matrix: $TP = |G_\oplus \cap G_{\hat{c}}|$, $FP = |(G - G_\oplus) \cap G_{\hat{c}}|$, $FN = |G_\oplus \cap (G - G_{\hat{c}})|$ and $TN = |(G - G_\oplus) \cap (G - G_{\hat{c}})|$. This is illustrated in Figure 8.1.

It is not obvious how PcG target genes should be predicted based on predicted PREs. There are, however, approaches that may be attempted. One way to predict target genes is to predict the closest annotated gene to each predicted PRE. This approach has been used by Ringrose *et al.*, although they note that they cannot

be sure that the predicted target genes are regulated by the corresponding predicted PREs [1].

An alternative approach is to predict target genes in both directions from a predicted PRE. This approach has been used by Zeng *et al.* [2], with the additional condition that the second closest gene should be within $4kb$ from the predicted PRE to be predicted as a target gene. In this thesis, the latter approach will be used for predicting target genes.

8.3 PREsent

For comparing predictions and experimental data, a software application called PREsent has been implemented. The goal when designing PREsent was for it to be able to compare multiple classifier runs with multiple experimental data sets, and to output informative statistics. The output should also be in a format that is easy to use. PREsent takes the following arguments: 1) classifier run XML-profiles (such as output by PRESVM), 2) genomic validation regions in GFF format (such as experimentally determined PREs), lists of validation genes and Wiggle format profiles.

The tasks of PREsent can be summarized as follows:

- Load classifier profiles and associated files (predicted regions, score profiles and score E-value table).
- Output information about the configuration of each classifier run compared.
- Output classifier validation statistics and curves based on the profiles.
- Compare classifier score profiles and any validation profiles.
- Compare predicted regions with validation regions
  - at the threshold used,
  - at thresholds varied over the genome-wide score range, and
  - at thresholds corresponding to different E-values.
- Predict target genes from predicted regions and compare these with validation genes
  - at the threshold used,
  - at thresholds varied over the genome-wide score range, and
  - at thresholds corresponding to different E-values.
- Compile report.

8.4 Genome-wide validation plots

For visualizing the performance of multiple classifiers, validation plots can be useful. For a score profile S, a set of validation regions V and a threshold t, regions can be predicted based on the threshold, and the $Sensitivity$, PPV and F_1-measure can be calculated. One way to visualize prediction results then is to construct a histogram for each measure, lining up classifiers for different validation data sets. This is useful, as having relative differences in the measures visualized with a common scale makes it easier to see the overall outcome. Venn-diagrams can also be constructed, which can be helpful as they will show both the relative set sizes and the size of the overlap.

If t is varied over the score range in S, a $Precision/Recall$ curve can be constructed. Doing this for multiple classifier score profiles against the same set of validation regions, the performance of multiple classifiers can be compared. Alternatively, the threshold t can be varied over the scores in the E-value table for the

classifier, and approximated E-values can thus be plotted against the *Sensitivity* and PPV.

The same types of plots can be made for target gene predictions. However, as TN can be defined for genes, ROC curves can also be constructed. For the ROC and PR plots, it makes sense to find the predicted genomic regions for different thresholds, and predict target genes for each of these sets of regions, such that complete ROC and PR plots can be constructed.

Additionally, it can be interesting to investigate how the scores of one classifier relate to the scores of another classifier, or to ChIP enrichment profiles. For a pair of profiles a and b, one way to do this is to get the ranges over a and b, and to divide each range into buckets. This, in turn, enables the construction of a grid, where for each window in a and an overlapping window in b, a count is added to the grid point corresponding to their profile value combination. The final grid is normalized to have some desired range and is visualized by drawing blocks with colour intensities indicating values. In case there are extreme profile values, it can be useful to first transform the profile values.

8.5 Comparing against multiple data sets

There are multiple types of validation data that are considered in this thesis, and multiple data sets. If classifiers $\hat{c}_1, ..., \hat{c}_n$ are compared, and a classifier $\hat{c}_i$ clearly dominates in all cases, then it makes sense to assume that $\hat{c}_i$ is superior. Typically, however, results are less clear cut. Some classifiers may perform better on some data sets and worse on others, making the interpretation of results more difficult.

To get an overall *Sensitivity* against multiple validation data sets, it can make sense to consider the consensus between the considered data sets. If there is a set of elements that all data sets agree on, these elements may correspond to strong examples, and it would thus be desirable to predict these. If the consensus of all the considered data sets is empty, another possibility is to find the consensus between pairs of data sets and take the union of these. A set of paired consensus will also consist of elements agreed upon by more than one data set, and these elements could also represent strong examples.

For an overall PPV against multiple validation data sets, it can be taken over the union of the considered data sets. The PPV indicates how many of the positive predictions correspond to validation data elements, and if more of the positive predictions agree with at least some validation data set, that may indicate better classification performance.

It may be that these overall *Sensitivity* and PPV values already are informative enough. Otherwise, to get an overall score for a classifier, one possibility is to calculate an F-score based on these overall sensitivities and Positive Predictive Values.

There is a potential issue with interpreting predictions that do not overlap with any regions in the experimental data sets. Experimental data from multiple genome-wide studies is used for evaluation in this thesis. It is possible that these studies have covered a large portion of the actual Polycomb/Trithorax Response Elements in the *Drosophila melanogaster* genome. However, there may also be many PREs that have not been covered by these studies. It seems safest to consider how well a classifier predicts experimentally determined PREs, and to take this as an indication of how good any novel predictions may be. Ideally, novel predictions of the best classifier found should be tested experimentally at a later point in time.

Chapter 9

Results

There are many possible configurations for PRESVM. Focus was on a subset of configurations that seemed reasonable. In this chapter, the results obtained will be presented and discussed.

9.1 Tests of PRESVM configurations

There are many potential configurations that could be investigated, resulting from selecting different SVM formulations, kernels, feature sets, parameter optimization methods and quality measurements used during parameter optimization. Some of these also have different parameters that could be varied. There are also different sets of motifs that could be used, and also potential different training sets. Due to the number of possibilities, it makes sense to find a reasonable base configuration to start from, and based on this investigate variations.

The training data used by Ringrose *et al.* [1] is based on known PREs, and it makes sense to start the investigation using this training set. It also makes sense to start the investigation using their set of motifs. The Particle Swarm Optimization method can investigate many different configurations as the particles spread over the search space. For the SVM formulation, the ϵ-Support Vector Regression formulation does not restrict the range of output values, and only requires two parameters in addition to the kernel parameters. Feature values should be scaled to be in the same range.

An issue is measuring the quality for parameter optimization. The training set is small, consisting of 12 PRE sequences and 16 non-PRE sequences, so there is a risk of over-fitting. Balanced cross-validation can be used when measuring quality during optimization to reduce this risk. For the quality measure, the Area Under the Curve would be affected by the small size of the training set, and smaller changes in quality may probably not be captured. Thus, it may be better to use a continuous measure, so instead the Pearson Correlation Coefficient was used. The chosen base configuration is summarized in Table 9.1.

This base configuration was tested both with the *nOcc* feature set and *nPair* feature set. For sampling of the training data to construct training vectors, three methods were attempted: 1) using the full sequences, normalizing frequencies according to sequence length, 2) sampling windows from the training sequences with a step size of 250`bp` (so as to keep the number of training vectors manageable) and 3) sampling windows from the negative training sequences with a step size of 250`bp`, and sampling the windows with the highest number of motif occurrences from the positive training sequences. Note that 3) is similar to the approach used by Zeng *et al.* [2], but here it was tested also with the *nPair* feature set.

For deciding the outcome of these tests, a separate quality measurement needs to be made. The training data is fairly small, so instead of cross-validating the whole procedure to measure the overall quality of each run, a separate validation set was used. For the positive validation set, 60 intergenic regions overlapping with windows with at least 10 occurrences of the motifs considered were randomly selected from the Enderle *et al.* [15] data. For the negative validation set, 60 sequences of length $10kb$ were randomly generated based on the

Random number generator seed:	123456.
Training set:	The same as used by Ringrose *et al.* [1] with the PREdictor.
Motif set:	The same as used by Ringrose *et al.* [1] with the PREdictor.
Window size:	500bp.
Window step size:	10bp.
Feature value scaling:	To the interval $[-1, 1]$.
SVM formulation:	ϵ-Support Vector Regression.
Parameter optimization method:	Particle Swarm Optimization, with $c_1 = c_2 = 0.5$, 100 iterations and using 7 particles.
Parameter optimization quality measure:	Pearson Correlation Coefficient based on 5-fold balanced cross-validation with two repeats.

Table 9.1: Base configuration used when testing different kernels.

nucleotide distribution of the *Drosophila melanogaster* genome. This validation set should give an indication of how well the classifier distinguishes the positive regions from noise. It is also useful to consider how well the classifier has learned to generalize to the training data, so the Area Under the Curve over a final training data cross-validation was also considered.

Sampling mode	Kernel	CVB5x10 $AUC(1.0)$	V_R $AUC(1.0)$	C
Full	Linear	<u>0.866818</u>	<u>0.629722</u>	745.271
	Quadratic	0.842045	0.584444	0.742821
	Cubic	0.835341	0.428889	0.387655
	RBF	0.844261	0.449028	1.41842
Windows	Linear	0.795341	0.417917	1.44569
	Quadratic	0.757528	0.0666667	0.866076
	Cubic	<u>0.876847</u>	<u>0.540556</u>	6.76265
	RBF	0.861989	0.361944	2.72602
Max. motifs/Windows	Linear	0.882812	<u>0.570417</u>	448.023
	Quadratic	0.912727	0.487917	0.552703
	Cubic	<u>0.926534</u>	0.54375	0.471798
	RBF	0.902443	0.486806	1.55038

Table 9.2: Tests of the *nOcc* feature set with different kernels and different approaches to sampling training sequence regions to construct training vectors. *Full* means the full training sequences were used to construct the training vectors, with normalization by sequence length. *Windows* means that windows were sampled from the training sequences with a step size of 250bp. *Max. motifs/Windows* means that for the positive training sequences the windows with the maximum number of motif occurrences were used, and for the negative training sequences windows were sampled with a step size of 250bp. *CVB5x10* $AUC(1.0)$ is the Area Under the Curve on the training data based on 5-fold cross-validation of training, with 10 repeats. V_R $AUC(1.0)$ is the Area Under the Curve against the validation data. The maximum value for each quality measure for each sampling mode is underlined. The cost parameter C is also shown.

The results of using the *nOcc* feature set are summarized in Table 9.2. The Area Under the Curve values for the validation set are generally low. It may be that the chosen positive examples are hard to learn to distinguish from noise when training on the used training sequences. The cost parameter C is largest when using linear kernels in two sampling modes, which may be due to difficulties with fitting a linear decision surface. It seems possible that a good parameter combination is not found by Particle Swarm Optimization and cross-validated correlation, given the modest size of the training set. When considering the measurements for moderate values of C, the combination of a cubic kernel and training with the positive sequence windows

with the highest number of motif occurrences gives the best Area Under the Curve values.

Sampling mode	Kernel	CVB5x10 $AUC(1.0)$	V_R $AUC(1.0)$	C
Full	Linear	0.935	0.590278	0.0348511
	Quadratic	0.939034	0.569444	4.43325
	Cubic	0.941705	0.573611	1.3227
	RBF	0.943295	0.571111	17.9308
Windows	Linear	0.914545	0.589167	2.30119
	Quadratic	0.878182	0.465278	2.02631
	Cubic	0.895568	0.474722	5.57897
	RBF	0.876591	0.478056	69.9747
Max. motifs/Windows	Linear	0.891477	0.600556	1.00347
	Quadratic	0.93608	0.481111	0.00190558
	Cubic	0.929943	0.486111	1.60879
	RBF	0.938239	0.4875	1.80039

Table 9.3: Tests of the *nPair* feature set with different kernels and different approaches to sampling training sequence regions to construct training vectors. The cutoff distance was set to 220`bp`. The meaning of the names are the same as in Table 9.2.

The results of using the *nPair* feature set are summarized in Table 9.3. This feature set gives better separation of the training data, but validation Area Under the Curve values are still low. However, the C values are moderate. The use of a linear kernel gives the highest Area Under the Curve values for the validation set. This may indicate that the use of non-linear kernels with the *nPair* feature set will not improve classification when training with the training data used, or that optimal SVM parameter values are not found for the non-linear kernels.

Considering that the number of features in the *nPair* feature set with 7 motifs is already equal to the number of training sequences (28), and since it seems that using a linear kernel with the *nPair* feature set gives the best separation of the validation set, it does not seem reasonable to extend the number of features with this training set.

Assuming that the original training data is too small to both search for optimal parameter values and to generalize, it is interesting to try using a new, larger training set. For the positive training examples, 60 new intergenic, motif-enriched regions were randomly selected from the Enderle *et al.* [15] data (not overlapping with the ones selected for validation). For the negative training examples, 60 motif-enriched, intergenic regions were randomly selected. Also, ideally, the classifier should learn to separate the original training data without it being used for training, so the Area Under the Curve against the original training data was also considered. Results of tests training with this data are summarized in Tables 9.4 and 9.5. Here, other feature sets were also tested.

Interestingly, when using this larger training set and the *nOcc* or *nPair* feature sets, the non-linear kernels give better separation than the linear kernels both of the validation sequences and of the original training data. The *nPair* feature set separates the larger training data more poorly than the *nOcc* feature set does, even when the *nOcc* feature set has a linear kernel. However, the *nPair* feature set separates the validation set and original training data better. This may indicate that non-linear kernels improve the generalization to unobserved instances for both of these feature sets when the training set is sufficiently large.

9.2 Genome-wide prediction with PRESVM

The aim of the PREdictor and of PRESVM is to predict PREs genome-wide. Focus was on training with the original PREdictor training data and on the *nOcc* and *nPair* feature sets.

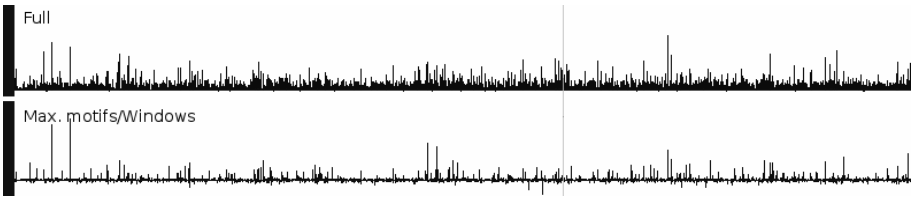


Figure 9.1: Snapshot from the Integrated Genome Browser [31] of the score profile curves of the two *nPair* runs on chromosome 3R. *Full* means that the classifier was trained using the full training sequences with normalization by length. *Max. motifs/Windows* means that for the positive training sequences the windows with the largest number of motif occurrences was used, and for the negative training sequences windows were samples with a step size of 250bp. There seem to be cleaner peaks for the *Max. motifs/Windows* run.

For the *nOcc* feature set, PRESVM was trained with a cubic kernel, using the positive training sequence windows with the highest number of motif occurrences for constructing the positive training vectors, and windows sampled with a step size of 250bp from the negative training sequences (i.e. the second to last configuration in Table 9.2).

For the *nPair* feature set, two configurations were tested. For both configurations, a linear kernel was used. For the first configuration, the whole training sequences were used with length normalization. For the second configuration, the positive training sequence windows with the highest number of motif occurrences were used, and negative training sequence windows were sampled with a step size of 250bp.

For all three configurations, the classifier threshold was calibrated for an E-value of 1, based on a randomly generated sequence 100 times the size of the *Drosophila melanogaster* genome, based on its nucleotide distribution. The results are summarized in Table 9.6. There is a large difference in the number of predictions made using each configuration. In the study by Ringrose *et al.* [1], the PREdictor predicted 167 candidate PREs genome-wide. For the comparison here, jPREdictor was trained again, using a step size of 10bp, which resulted in the prediction of 179 candidate PREs. As the use of a linear kernel and the *nPair* feature set should be similar to the PREdictor, the larger number of predictions compared to the PREdictor is likely due to the way the decision surface of the Support Vector Machine was constructed.

With the *nPair* feature set, when sampling the positive training sequence windows with the highest number of motif occurrences and negative training sequence windows with a step size of 250bp, PRESVM makes a much larger number of predictions than when training using the full sequences with length normalization. Three potential explanations for this are: 1) important features of some training sequences are normalized away when training using the full sequences and length normalization, due to different sequence lengths; 2) as multiple windows are sampled from the negative training sequences in the last approach listed, the classifier becomes more selective, which impacts predictions made in the randomly generated sequence during threshold calibration; 3) parameter optimization is noisy, and the difference in the number of predictions is caused by this.

The score curves for the *nPair* runs are shown in Figure 9.1 for chromosome 3R, visualized with the Integrated Genome Browser [31]. The peaks seem to be cleaner when having trained with the positive training sequence windows with the maximum number of motif occurrences and negative training sequence windows with a step size of 250bp. Thus, it appears that the classifier trained with this configuration is more specific. This also agrees with the higher V_R $AUC(1.0)$ for this run, as seen in Table 9.6.

The PRESVM predictions made in the *Max. motifs/Windows nPair* run were compared with the predictions made by the jPREdictor, as well as experimental data. The experimentally determined sets of regions considered are listed in Table 9.7. As is seen in Figure 9.2(a), the regions predicted by PRESVM overlap with slightly fewer than half of the regions predicted by jPREdictor. Figures 9.2(c) and 9.2(d) show that

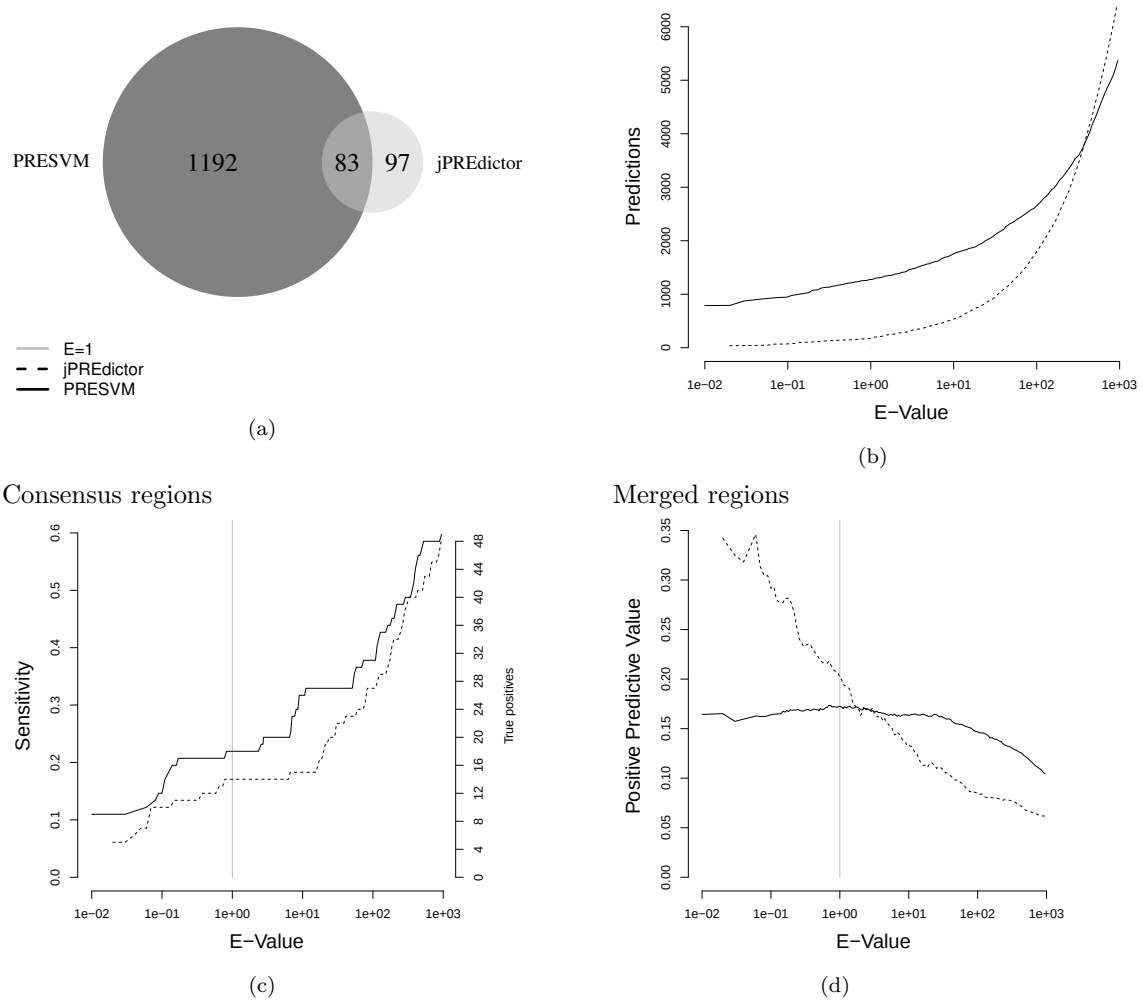


Figure 9.2: 9.2(a): Venn diagram illustrating the number of predictions of and overlap between predictions made by jPREdictor and PRESVM at an E-value of 1. 9.2(b): Numbers of predictions versus E-values. In each plot, the solid line corresponds to PRESVM and the dotted line corresponds to jPREdictor. 9.2(c): Sensitivities against the consensus regions of the considered data sets for different E-values. 9.2(d): Positive Predictive Values against the merged regions of the considered data sets for different E-values.

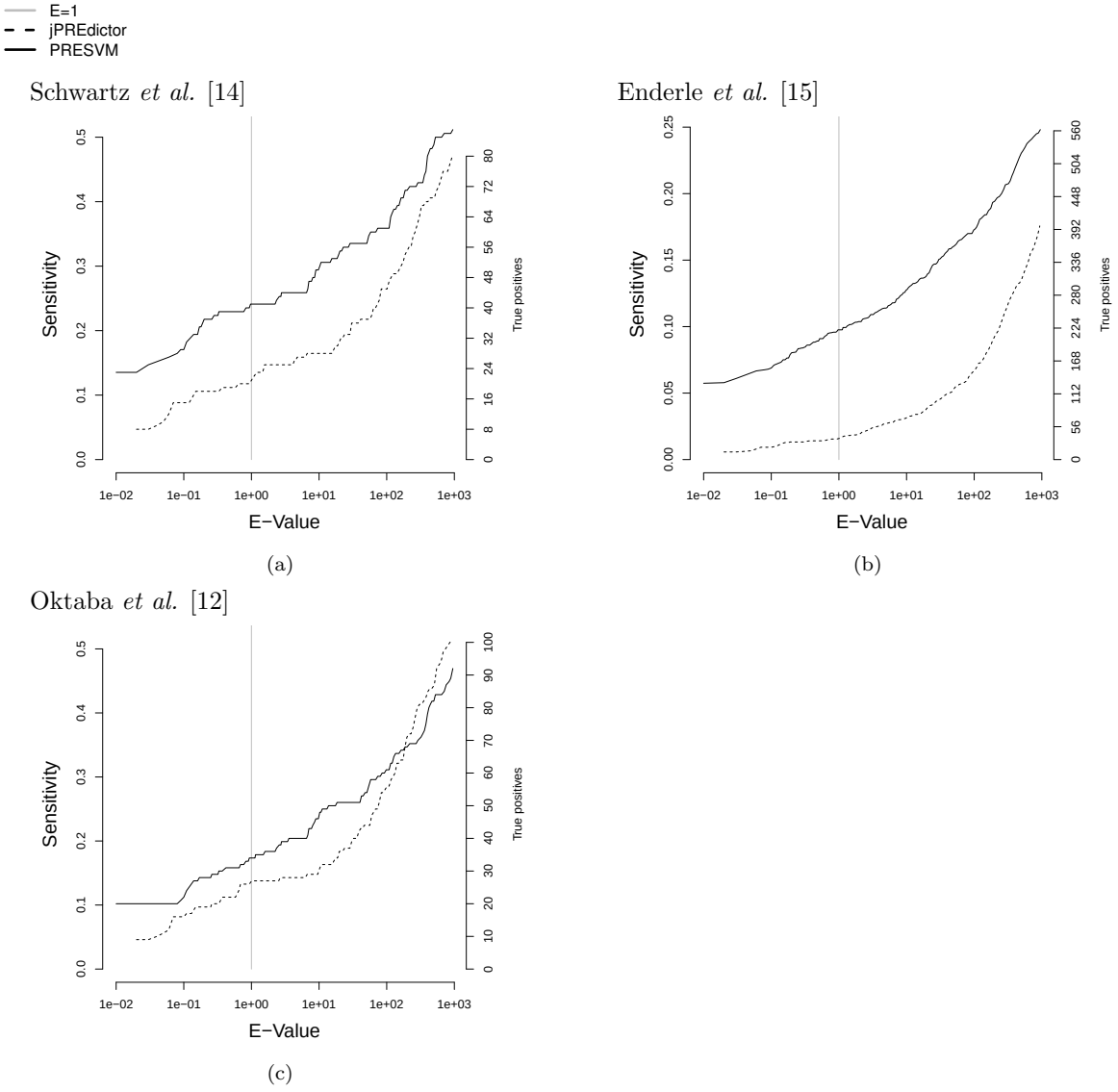
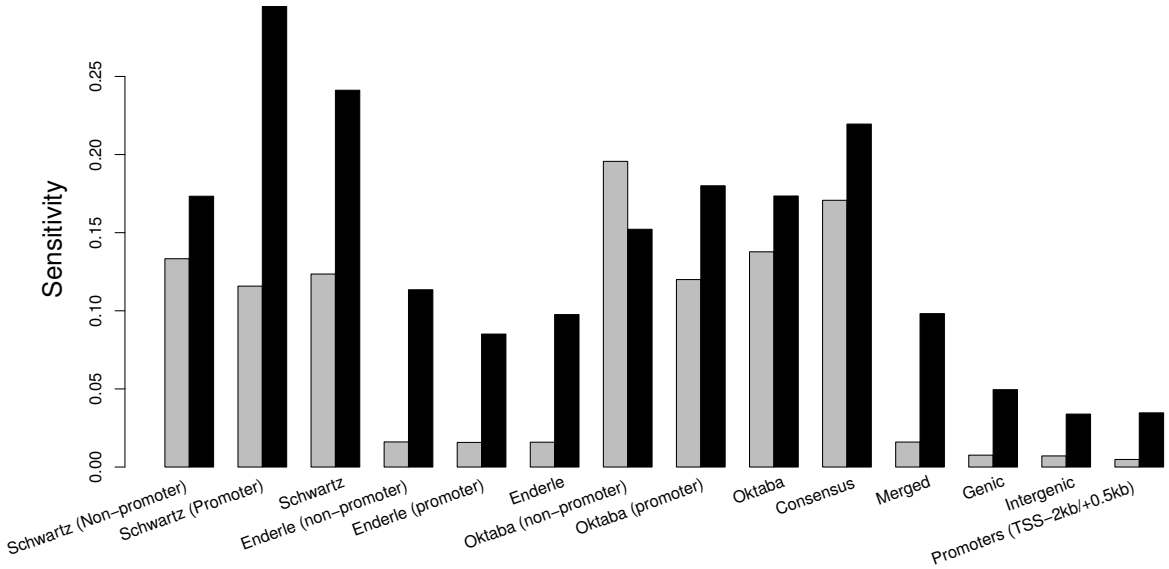
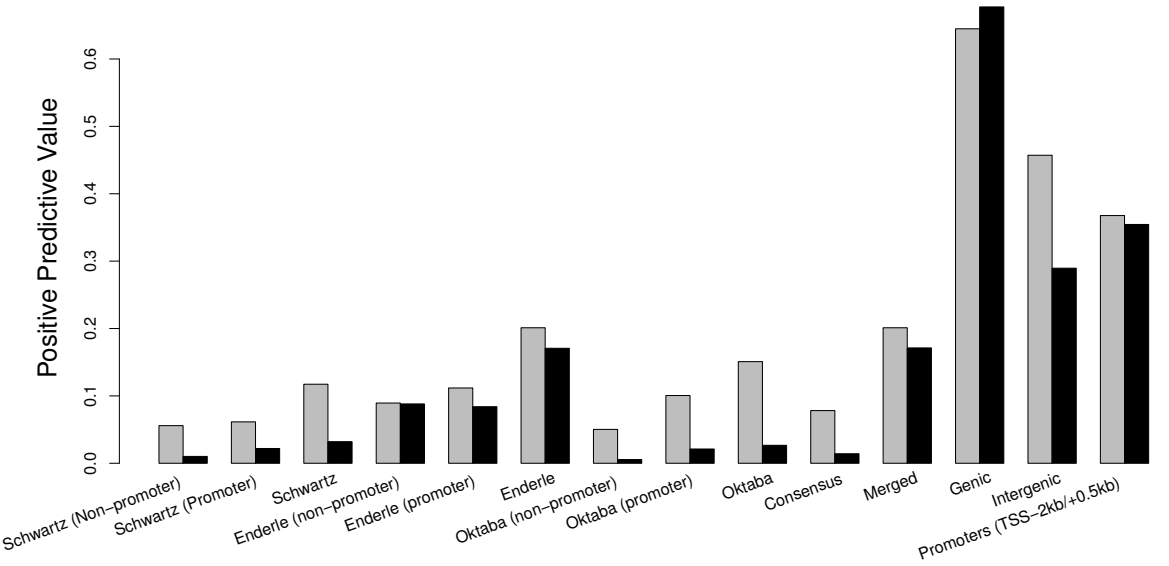


Figure 9.3: Sensitivities against the considered sets of experimentally determined regions. In each plot, the solid line corresponds to PRESVM and the dotted line corresponds to jPREdictor. 9.3(a): Schwartz *et al.* [14] regions. 9.3(b): Enderle *et al.* [15] regions. 9.3(c): Oktaba *et al.* [12] regions.



(a)



(b)

Figure 9.4: Histograms of Sensitivities and Positive Predictive Values for regions predicted by PRESVM (black bars) and jPREdictor (grey bars) at an E-value of 1. The considered experimentally determined regions are separated by whether or not they overlap with promoter regions, denoted *Promoter* and *Non-promoter* respectively. Promoter regions were defined as the Transcription Start Site (TSS) of annotated genes -2kb/+0.5kb.

PRESVM predicts only few additional regions that overlap with the consensus regions, and the Positive Predictive Values for the merged regions are similar, but slightly lower for PRESVM, indicating that the tendency to predict regions overlapping with the sets of experimentally determined regions considered is similar. Figure 9.3 shows the Sensitivities of PRESVM and jPREdictor against the considered sets of experimentally determined regions. The Sensitivities and Positive Predictive Values for an E-value of 1 are summarized in Figure 9.4.

Target genes may be predicted from predicted regions. The results of predicting target genes from predicted regions were investigated. An overview of the results is given in Figure 9.5. PRESVM is more sensitive to the gene intersection, but has a lower PPV than jPREdictor against the union of target genes.

Combined F_1-scores were calculated for PRESVM and jPREdictor. These are based on the sensitivities against consensus regions and PPV against merged regions for bands, and sensitivities against the intersection and PPV against the union for genes. The combined F_1-scores are listed in Table 9.9.

9.3 Comparison with the jPREdictor

The PRESVM configuration for which genome-wide prediction was investigated used a linear kernel and the *nPair* feature set. Thus, the trained classifier may be compared with jPREdictor in terms of the support vectors.

To find weights for motif pairs based on a PRESVM classifier trained with the *nPair* feature set and a linear kernel, the reformulation of the decision function into a weighted sum described earlier can be used.

$$\begin{aligned}\hat{c}(\vec{x}) &= \sum_{i=1}^{l} \left(c_i k(\vec{y}_i, s(\vec{x}))\right) - \rho \\ &= \sum_{j} w_j x_j + d\end{aligned}$$

where

$$w_j = \sum_{i=1}^{l} r_j c_i y_{i,j}$$

and

$$d = \sum_{j} \left(\sum_{i=1}^{l} \left(r_j c_i y_{i,j}\right) q_j \right) - \rho.$$

where the $\vec{y}_i$ are support vectors, and the c_i are coefficients. x_j denotes the j'th component of $\vec{x}$, and $y_{i,j}$ denotes the j'th component of $\vec{y}_i$. q_i is the shifting of feature i and r_i is the scaling factor.

The weights for the PRESVM classifier used for genome-wide prediction were calculated. These weights, as well as the weights assigned by jPREdictor to motif pairs are listed in Table 9.10. Whereas jPREdictor assigns negative weights to three motif pairs, PRESVM assigns negative weights to 11 pairs. Two of the pairs are assigned the weight 0 by PRESVM.

9.4 Interpretations of results

When using the original PREdictor [1] training data, the *nOcc* feature set seems to benefit from a non-linear kernel, but not the *nPair* feature set, which might be due to the modest size of the training set. When using a larger training set, where the positive training examples are based on ChIP data, both of these feature sets seem to benefit from non-linear kernels, as summarized in Table 9.4.

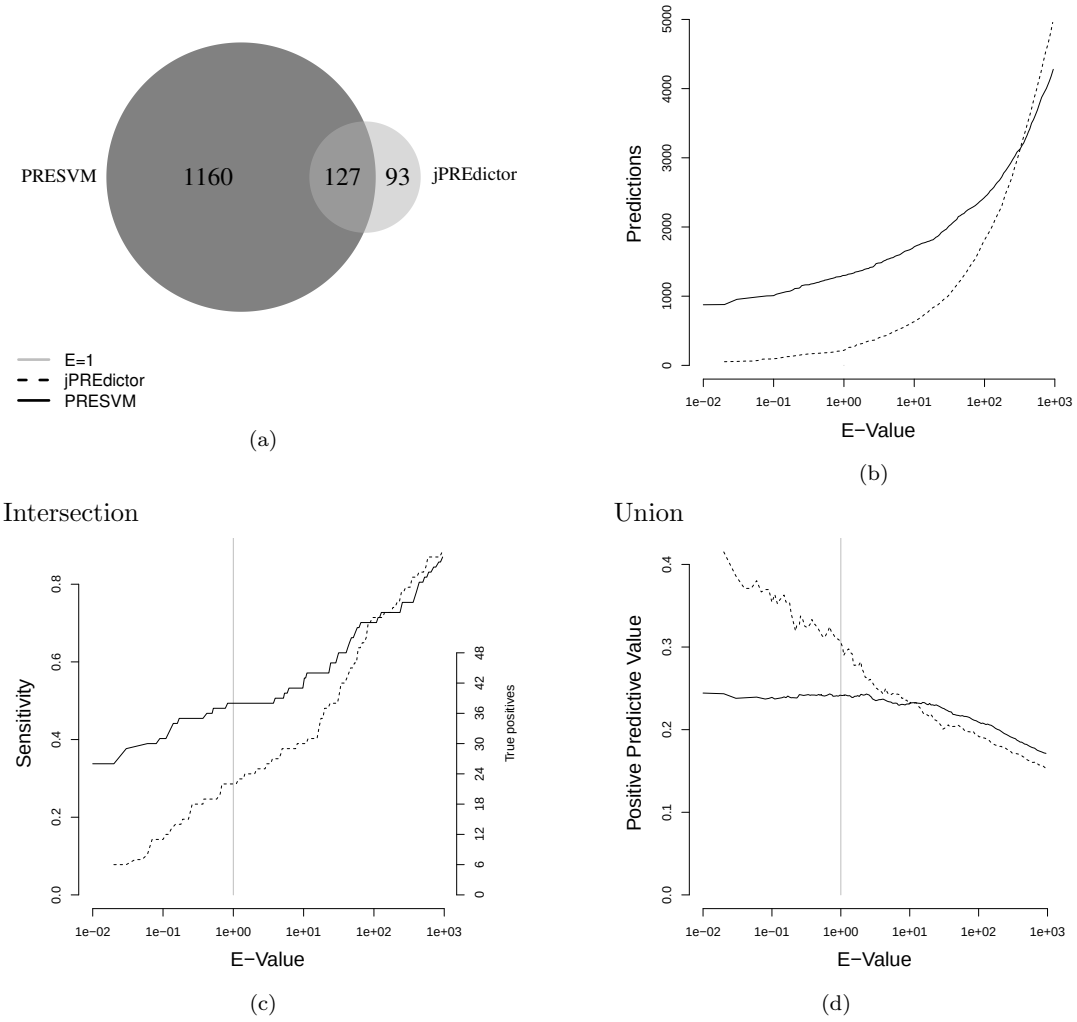


Figure 9.5: 9.5(a): Venn diagram illustrating the number of target gene predictions based on regions predicted by PRESVM and jPREdictor at an E-value of 1 and their overlap. 9.5(b): Numbers of predicted target genes versus E-values. The solid line corresponds to PRESVM and the dotted line corresponds to jPREdictor. 9.5(c): Sensitivities against the intersection of the considered target gene sets for different E-values. 9.5(d): Positive Predictive Values against the union of the considered target gene sets for different E-values.

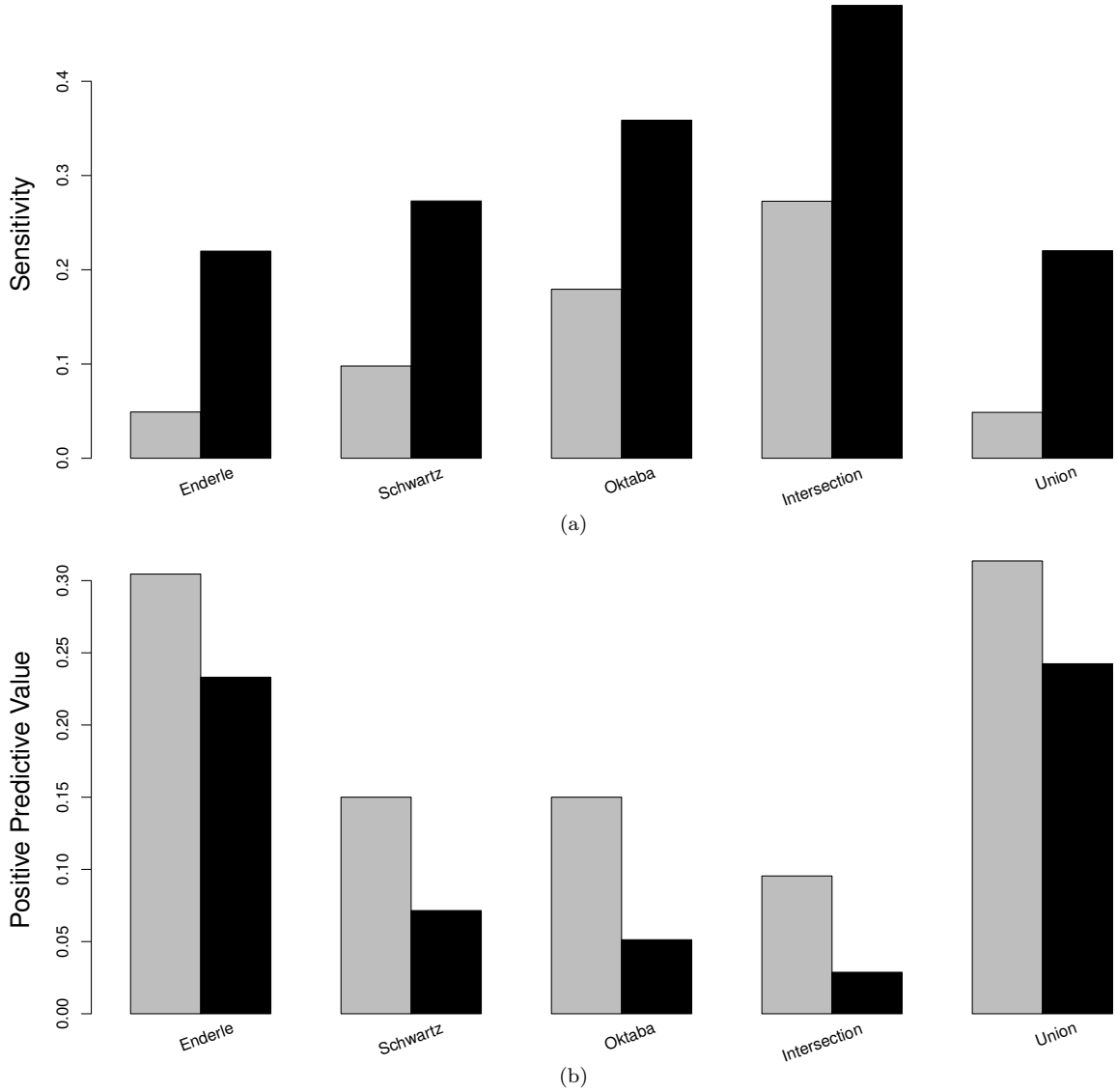


Figure 9.6: Histograms of Sensitivities and Positive Predictive Values for target genes predicted based on PRESVM (black bars) and jPREdictor (grey bars) predicted regions at an E-value of 1.

As for the other feature sets that have been considered in this thesis, considering the results summarized in Table 9.4, some feature sets show promising results. For example $nPair + MDP$ and $nOccPair$ give good Area Under the Curve values for some kernels. For the periodic motif occurrence frequency feature sets, results are not as impressive. This could be due to periodic occurrence of these motifs not being predictive of PREs, that the considered phases are not optimal, or that perhaps other frequencies would have been more predictive. As focus was kept on the original PREdictor [1] training set, the new feature sets were not considered for genome-wide runs.

Regarding the genome-wide PRESVM run that was investigated, when training PRESVM with the original PREdictor training data [1], a linear kernel, sampling the positive training sequence windows with the highest numbers of motif occurrences, and negative training sequence windows with a step size of 250`bp`, also calibrating the threshold for an E-value of 1, 1275 regions were predicted as candidate PREs. This is $\frac{1275}{179} \approx 7.2$ times as many predictions as were made by jPREdictor for a similar configuration. However, the sensitivities against the validation region consensus and Positive Predictive Values against the merged validation regions were similar. The sensitivities against the validation region sets were generally higher for PRESVM. This could indicate that the way the SVM assigns weights to the features, when sampling the training data in this way, enables increased sensitivities overall, while not making it much less specific. It should be noted that the largest increase in sensitivity for PRESVM was seen on the Enderle *et al.* [15] regions, and this set is much larger than the other considered data sets. Also, the combined F_1 scores (see Table 9.9) are sufficiently similar that it is difficult to conclude that one classifier is superior.

Feature set	Kernel	CVB5x10 $AUC(1.0)$	V_R $AUC(1.0)$	V_O $AUC(1.0)$	C
$nOcc$	Linear	0.699383	0.424583	0.666667	2.19978
	Quadratic	0.701031	0.505417	0.677083	3.21686
	Cubic	0.700833	0.479583	0.6875	13.7148
	RBF	0.701911	0.492083	0.666667	0.300478
$nPair$	Linear	0.656664	0.659722	0.713542	0.0133011
	Quadratic	0.653158	0.672917	0.822917	4.53643
	Cubic	0.598004	0.394028	0.697917	1.56859
	RBF	0.658914	0.595139	0.776042	0.264751
$PEDI$	Linear	0.666081	0.624167	0.677083	0.010201
	Quadratic	0.655475	0.644167	0.666667	40.5446
	Cubic	0.664117	0.528889	0.708333	0.409522
	RBF	0.67605	0.597222	0.708333	0.203865
$nPairDH$	Linear	0.655236	0.728611	0.677083	1e-05
	Quadratic	0.655694	0.586667	0.742188	0.145617
	Cubic	0.666103	0.576111	0.723958	0.000388748
	RBF	0.652753	0.581667	0.752604	0.249097
$nOcc + MDP$	Linear	0.717572	0.466667	0.723958	0.0274017
	Quadratic	0.724003	0.561667	0.765625	6.06882
	Cubic	0.720178	0.475556	0.739583	1.93316
	RBF	0.729153	0.551667	0.729167	0.0520131
$nPair + MDP$	Linear	0.705578	0.669167	0.78125	0.00605001
	Quadratic	0.697669	0.690556	0.796875	1.25972
	Cubic	0.695842	0.704722	0.802083	2.34015
	RBF	0.711331	0.525278	0.75	0.123289
$nOccPair$	Linear	0.675633	0.748333	0.8125	0.0293423
	Quadratic	0.687433	0.654722	0.65625	2.21382
	Cubic	0.675711	0.745556	0.8125	5.7479
	RBF	0.692388	0.186528	0.570312	8.59597
$nPairCosi$ (mRMR 30)	Linear	0.568675	0.684167	0.799479	2.83436
	Quadratic	0.622078	0.703611	0.609375	56.8753
	Cubic	0.622017	0.703889	0.609375	7.70055
	RBF	0.675086	0.613889	0.643229	0.254287
$nPair2D$	Linear	0.521264	0.315833	0.869792	4.87805
	Quadratic	0.581803	0.290278	0.84375	0.316633
	Cubic	0.575117	0.409444	0.78125	2.34948
	RBF	0.582686	0.287778	0.770833	1.94428

Table 9.4: Test results of using different feature sets and kernels while training on ChIP-based positive regions and general motif-enriched negative regions. Most values mean the same as in Table 9.2. V_O $AUC(1.0)$ is the Area Under the Curve for the original PREdictor [1] training data. In all cases, training was done using the full sequences with length normalization.

Feature set	Kernel	CVB5x10 $AUC(1.0)$	V_R $AUC(1.0)$	V_O $AUC(1.0)$	C
$nOcc$	Linear	0.604608	0.691944	0.75	0.507584
	Quadratic	0.468426	0.145972	0.934896	262.477
	Cubic	0.617692	0.704722	0.84375	169.529
	RBF	0.578867	0.188611	0.731771	3.5352
$nPair$	Linear	0.585178	0.695694	0.776042	0.142415
	Quadratic	0.614217	0.220556	0.760417	0.0304963
	Cubic	0.555761	0.3875	0.78125	1.43644
	RBF	0.587881	0.408056	0.802083	0.846935
$PEDI$	Linear	0.554475	0.622222	0.791667	0.580142
	Quadratic	0.618903	0.665833	0.890625	0.0191771
	Cubic	0.582611	0.764167	0.822917	3.10135
	RBF	0.577194	0.769444	0.807292	0.453361
$nPairDH$	Linear	0.574644	0.484722	0.807292	1.86601
	Quadratic	0.587886	0.528333	0.8125	1.45742
	Cubic	0.573561	0.401389	0.817708	0.437373
	RBF	0.580014	0.463056	0.885417	0.040138
$nOcc + MDP$	Linear	0.625939	0.370278	0.739583	0.151875
	Quadratic	0.610614	0.373889	0.776042	2.0951
	Cubic	0.691667	0.462778	0.942708	2.62215
	RBF	0.621406	0.255972	0.817708	0.296532
$nPair + MDP$	Linear	0.625928	0.309444	0.802083	0.639738
	Quadratic	0.61545	0.289444	0.880208	0.796035
	Cubic	0.695517	0.482361	0.901042	0.129159
	RBF	0.634572	0.221806	0.760417	1.72311
$nOccPair$	Linear	0.652858	0.567778	0.869792	2.67249
	Quadratic	0.656928	0.584722	0.901042	0.459459
	Cubic	0.648886	0.556667	0.880208	48.7238
	RBF	0.655014	0.5875	0.854167	4.14149
$nPairCosi$ (mRMR 30)	Linear	0.563488	0.495	0.927083	3.27594
	Quadratic	0.393911	0.649167	0.880208	1.00529
	Cubic	0.586544	0.534722	0.921875	1.06533
	RBF	0.61786	0.504167	0.947917	1.18963
$nPair2D$	Linear	0.471525	0.506667	0.755208	0.191621
	Quadratic	0.508333	0.439167	0.65625	1.52428
	Cubic	0.548936	0.445278	0.828125	0.0584171
	RBF	0.522369	0.424444	0.645833	0.280761

Table 9.5: Test results of using different feature sets and kernels while training on ChIP-based positive regions and general motif-enriched negative regions. Most values mean the same as in Table 9.2. V_O $AUC(1.0)$ is the Area Under the Curve for the original PREdictor [1] training data. In all cases, training was done using the the positive training sequence windows with the largest number of motif occurrences and windows from the negative training sequences sampled with a step size of 250bp.

Feature set	Sampling mode	Kernel	CVB5x10 $AUC(1.0)$	V_R $AUC(1.0)$	Train TP	Predictions
nOcc	Max. motifs/Windows	Cubic	0.921932	0.54375	1	209
nPair	Full	Linear	0.939602	0.590278	2	394
nPair	Max. motifs/Windows	Linear	0.898466	0.600556	3	1275
jPREdictor						179

Table 9.6: Overview of results of training PRESVM on the original PREdictor [1] training data, calibrating the classifier threshold for an E-value of 1 and applying each classifier genome-wide for prediction. CVB5x2 $AUC(1.0)$ and V_R have the same meanings as in Table 9.2. Train TP is the number of positive training sequences that are classified as positive after calibrating the threshold for an E-value of 1. Predictions is the number of predictions made genome-wide. The number of predictions made by jPREdictor when trained using a step size of 10`bp` is also shown.

Authors	Method	Size
Enderle *et al.* [15]	ChIP-Seq	2265
Schwartz *et al.* [14]	ChIP-chip	170
Oktaba *et al.* [12]	ChIP-chip	196
Merged		2251
Consensus		82

Table 9.7: Sets of experimentally determined regions considered. For the merged set, overlapping regions in the different sets are merged into larger regions. For the consensus set, the merged regions that overlap with all sets of regions are used.

Authors	Method	Size
Enderle *et al.* [15]	ChIP-Seq	1365
Schwartz *et al.* [14]	ChIP-chip	337
Oktaba *et al.* [12]	ChIP-chip	184
Union		1417
Intersection		77

Table 9.8: Sets of experimentally determined PcG target genes considered.

Bands:

Classifier	Consensus sensitivity	Merged PPV	Combined F_1 score
PRESVM	21.951220 %	17.118513 %	0.192359771
jPREdictor	17.073171 %	20.111732 %	0.184683036

Genes:

Classifier	Intersection sensitivity	Union PPV	Combined F_1 score
PRESVM	48.051948 %	24.242424 %	0.32226456
jPREdictor	27.272727 %	31.363636 %	0.291754754

Table 9.9: Combined F_1-scores for PRESVM and the jPREdictor.

Motif a	Motif b	jPREdictor weight	PRESVM weight
En	En	0.4844	0
G10	En	1.1902	-27.2969443963334
G10	G10	-1.9585	15.2765051128981
GAF	En	1.5476	38.2290657971049
GAF	G10	-0.8176	-0.0451424507851569
GAF	GAF	-0.5668	-1.62808256575056
PF	En	2.0314	-21.1215020716622
PF	G10	2.9524	28.1916576961376
PF	GAF	3.0782	5.29855040007117
PF	PF	2.7144	-1.37346152969305e-13
PM	En	1.0182	0
PM	G10	3.0405	16.2795843997754
PM	GAF	1.6248	0.609360273423749
PM	PF	3.3341	13.564644415107
PM	PM	1.5849	-4.05432631911395e-13
PS	En	2.1917	-39.9567205108171
PS	G10	0.9423	-15.9005217074748
PS	GAF	0.7052	3.26551486991582
PS	PF	2.0339	3.11064525888481
PS	PM	3.0454	6.10448495909537
PS	PS	1.8341	-6.13350429382364
Z	En	2.2569	20.178348888145
Z	G10	1.1588	-1.48157035777308
Z	GAF	1.5001	2.29584692845266
Z	PF	0.9795	0.859655642652733
Z	PM	2.6204	2.56286529994112
Z	PS	1.3609	7.95097369527364
Z	Z	1.1226	-17.5524791615663

Table 9.10: Motif pair weights of jPREdictor compared with those of PRESVM.

Chapter 10

Implementation

The implementations of PRESVM and PREsent are described in this chapter. This includes a series of design choices and algorithms.

10.1 PRESVM implementation

PRESVM has been implemented in C++. For the Support Vector Machine implementation, LibSVM [24] is used. For the specification of sequence motifs and the genome, XML is used. For XML parsing, RapidXml [35] is used. Feature selection via mRMR has been added by calling on the statistics application R [36], for which mRMR feature selection is provided with the mRMRe package [37].

As there are many possible configurations for PRESVM, it has been desirable to run many tests. The implemented methods for searching for Support Vector Machine parameters can require many training cycles. Also, during threshold calibration, the trained classifier is applied to a large, randomly generated sequence (typically 100 times the size of the genome). As both PRESVM and LibSVM have been implemented in C++, PRESVM is statically linked with LibSVM, giving PRESVM efficient access to the Support Vector Machine implementation. In addition, motifs are central for the PRESVM feature sets, and efficient parsing and handling of motifs has therefore been desirable. Sequence parsing will be discussed in the following sections.

10.2 Sequence reading

Reading sequences in PRESVM has been implemented by defining a general base class with an operation to read a certain number of nucleotides. The implementations of this base class are referred to as sequence stream classes. This base class is implemented in multiple variants: reading from a buffer, generating a random sequence for a given nucleotide distribution, streaming from a raw sequence file and streaming from a FASTA file.

This base sequence reading class set is augmented by a FASTA batch reader, which generates a sequence stream for each batch sequence. Also, since mainly sequence windows are used, a class has been implemented that takes a sequence stream and reads windows, taking care of any needed buffering needed and keeping track of coordinates, as well as edge effects.

For a genome annotation, an XML file is used, in which paths to chromosome sequence files are given. Loading of a genome annotation is implemented as its own class. For chromosome sequence files, both raw and FASTA format files are supported. This class contains a method by which a chromosome sequence stream can be requested, as well as general information about the genome, such as the nucleotide distribution and the length in base pairs. In addition to providing chromosome sequence streams, this class also includes

loading of annotated genes. For annotated genes, annotations were downloaded from FlyBase [4] in FASTA format, and the sequence headers were used to extract information, such as gene coordinates, strand and names (including synonyms).

10.3 Motif occurrence parsing

A simple method for parsing motif occurrences is as follows: 1) Step through nucleotides of an input sequence. 2) From the current nucleotide, try to match all motifs used. 3) Note any matches found. This method has been implemented in PRESVM under the name "naïve parsing". It is illustrated in Algorithm 10. Although this method for parsing motif occurrences works, it is inefficient.

Algorithm 10 Naïve motif occurrence parsing

Input:
$S : A...B \in \mathbb{S}_{\mathtt{nt}}$: Input sequence.
$Motifs \subset \mathbb{S}_{\mathtt{nt}}$: Set of motifs.
$Match(S_1, S_2)$: Compares $S_1, S_2 \in \mathbb{S}_{\mathtt{nt}}$. Returns the number of differing nucleotides.
$ReverseComplement(m)$: Gives the reverse complement of $m \in Motifs$.
$AllowedMismatches(m)$: Returns the number of allowed mismatches for motif $m \in Motifs$.

$i \leftarrow A$
while $i \leq B$ **do**
  for all $m : m_a...m_b \in Motifs$ **do**
    if $Match(S : i...i + \lambda(m), m) \leq AllowedMismatches(m)$ **then**
      Register occurrence of m on the plus strand, starting at nucleotide i.
    else if $Match(S : i...i + \lambda(m), ReverseComplement(m)) \leq AllowedMismatches(m)$ **then**
      Register occurrence of m on the minus strand, starting at nucleotide i.
    end if
  end for
  $i \leftarrow i + 1$
end while

To get a more efficient motif occurrence parser, it can instead be considered that the parsed motif occurrences will cover up to multiple nucleotides, and thus every nucleotide of the input sequence corresponds to some state of parsing up to multiple motif occurrences. This can be noted more abstractly as a set of motif occurrence parsing states. Starting with a start state of parsing no motif occurrences, reading a particular nucleotide may initiate the parsing of one or more motif occurrences. This gives a transition from the start state for the considered nucleotide. This can be done for `A`, `C`, `G` and `T`, marking the transitions with the corresponding nucleotides, which covers the possibilities for the first read nucleotide. This process can be continued for the resulting states, also ensuring that equal states are re-used, and continuing until no new states are constructed. This results in a graph called a Finite State Machine (FSM), which covers the complete parsing of the considered motifs. The construction of the FSM is illustrated in Algorithm 11.

When parsing a sequence using a Finite State Machine, one starts from the starting state, and for each nucleotide read one moves along the corresponding transition in the graph. For some states, the ends of parsing some motif occurrences are reached, and for these states the corresponding motif occurrences are registered. This is illustrated in Algorithm 12.

For the resulting Finite State Machine graph, the states of parsing up to multiple motif occurrences are implicit to each node. Thus, for each read nucleotide, one only needs to transition to the appropriate node, and potentially register some motif occurrences.

Algorithm 11 Motif occurrence parsing Finite State Machine construction

Input:
$StartParseState()$: Gives the state of parsing no motif occurrences.
$Extend(s, n)$: Creates a parsing state based on state s extended by nucleotide n.
$Transition(s_1, s_2, n)$: Creates a transition from s_1 to s_2 for nucleotide n.

(Parsing states contain nucleotide indices of motifs being parsed, the strands they are being parsed on as well as registered mismatches so far)

$S \leftarrow \{StartParseState()\}$ ▷ Set of states.
$T \leftarrow \emptyset$ ▷ Set of transitions.
$Q \leftarrow S$ ▷ Queue for states to extend.
while $Q \neq \emptyset$ **do**
  $s_b \in Q$
  $Q \leftarrow Q / s_b$
  for all $n \in \{\mathtt{A}, \mathtt{T}, \mathtt{G}, \mathtt{C}\}$ **do**
    $s_n \leftarrow Extend(s_b, n)$
    if $s_n \notin S$ **then** ▷ Add to queue if it is a new state.
      $Q \leftarrow Q \cup \{s_n\}$
      $S \leftarrow S \cup \{s_n\}$ ▷ Also add to the states.
    end if
    $T \leftarrow T \cup \{Transition(s_b, s_n, n)\}$ ▷ Add a transition (unless already present).
  end for
end while

Algorithm 12 Motif occurrence parsing with a Finite State Machine

Input:
$S : A...B \in \mathbb{S}_{\mathtt{nt}}$: An input sequence.
$StartParseState()$: Gives the state of parsing no motif occurrences.
$Transition(s, n)$: Gives the state for transitioning from state s for nucleotide n.
$MotifOccurrences(s)$: Gives the set of motif occurrences that have finished parsing in state s.

$i \leftarrow A$
$s \leftarrow StartParseState()$
while $i \leq B$ **do**
  $s \leftarrow Transition(s, S : i...i)$
  for all $o \in MotifOccurrences(s)$ **do**
    Register occurrence o (with strand and starting position based on state).
  end for
  $i \leftarrow i + 1$
end while

10.4 Motif occurrence handling

Motif occurrences are added and removed frequently when moving across sequences in windows. Thus it is desirable to have an efficient data structure for managing motif occurrences. This data structure should also give efficient access to occurrences of a particular motif, as such access will help for feature sets making use of occurrences of particular motifs.

The data structure implemented in PRESVM keeps track of motif occurrences in a table. This table is expanded when needed, by doubling its size. Additionally, the following lists of motif occurrences are kept:

1) A doubly linked list of all motif occurrences,
2) doubly linked lists of occurrences of particular motifs, and
3) a singly linked list of free motif occurrences.

These lists are implemented using indices for the motif occurrence table.

As the motif occurrence table is expanded, unused motif occurrences are added to 3). As new motif occurrences are registered, free occurrences are removed from 3), updated with information about the particular occurrences and added to lists 1) and 2). As motif occurrences are removed, memory is not freed, but instead the occurrences are removed from 1) and 2) and added to 3).

There are multiple arguments for why this data structure was chosen for handling motif occurrences:

1) As linked lists are used (with links relating the motif occurrences themselves, rather than by an auxiliary data structure), motif occurrences can be removed in linear time, and added in close to linear time (except for when the table needs to be expanded). This is useful, as motif occurrences will be registered and removed often.
2) Using memory from a single table to keep track of motif occurrences should keep them close in memory, potentially improving memory access times.
3) Since motif occurrences index the motif occurrence table for their lists (instead of keeping pointers), the need for updating pointers upon reallocation of the table is avoided.
4) Using a list of free motif occurrences, instead of freeing memory, reduces the need for reallocation.

This data structure is illustrated in Figure 10.1. In addition to the points mentioned, the container data structure also contains a $Flush()$ method, which removes all motif occurrences (moving them to the free list). This is useful for when moving from one sequence to another.

10.5 PREsent implementation

PREsent has been implemented in C++. RapidXml [35] is used for parsing XML files. PREsent calls LaTeX to generate reports in PDF format. Plots are generated by calling the statistics software R [36]. R in turn outputs the plots as PDF files, which are incorporated into the reports. For generating Venn diagrams, the VennDiagram package for R is used [38].

As R is used for constructing many different plots, a class was implemented for convenient calling of R. It has been constructed such that a script may be constructed by using the `<<` operator upon an instance, and after the script has been constructed, R can be called by calling a class method. Handling the interaction with R in this way reduces the amount of code for the construction of each R script. Using the `<<` operator also keeps syntax in the style of the C++ standard library.

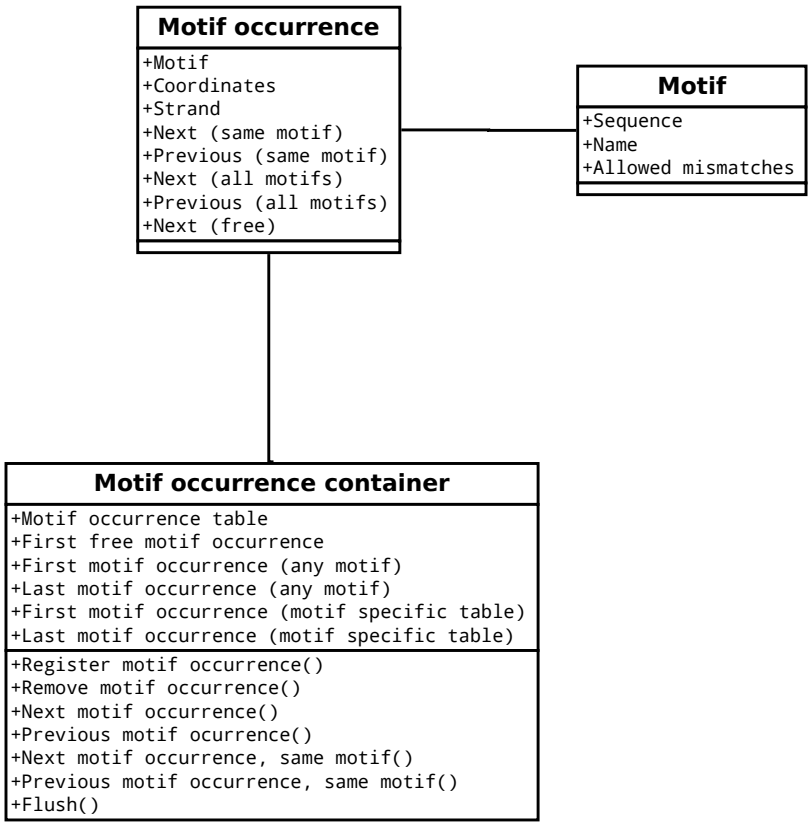


Figure 10.1: Overview of the motif occurrence container data structure.

In PREsent, parsing of GFF files, gene list files, *E*-value table files and Wiggle files has been implemented. For loading genome annotations, the code from the PRESVM implementation is used.

For handling genomic regions (such as from GFF files), a data structure is used to represent each region, a class is used for each chromosome to contain genomic regions, and a class contains the chromosome region containers. This will be referred to as a band database. The implemented operations for band databases include:

- find genes from bands;
- find the intersection between two band databases;
- get confusion matrix values based on overlaps of one band database with a secondary band database;
- separate regions according to whether they overlap with a secondary band database or not;
- load regions (from a GFF file, output by jPREdictor, from a tabulated list or from a list of the format `C:A...B`, where `C` is the chromosome, `A` is the start and `B` is the end of the region).

Gene lists are handled by referring to genes in the annotation data structure implemented for PRESVM, where one class handles genes for each chromosome and a class handles chromosome gene containers. This is referred to as a gene database. Implemented operations include:

- get the top N scored genes;
- find the intersection of two gene databases;
- find the union of two gene databases;
- get confusion matrix values based on overlap with a secondary gene database;
- load a gene list.

Wiggle files are handled by a class in which profile data is kept for each chromosome, and a class to contain chromosome profiles. The following operations are supported:

- get the highest profile value for each band in a band database;
- get confusion matrix values against a band database based on applying a threshold to the score profile and finding bands;
- get number of bands resulting from applying a threshold to the profile;
- load profile from a Wiggle file;
- get a band database based on applying a threshold to the profile.

E-value tables are handled by one data structure containing each threshold and corresponding *E*-value, and a class for handling the *E*-values. The supported operations are:

- load an *E*-value table;
- get threshold corresponding to the closest *E*-value in the table to a specified *E*-value.

There is a dedicated class for XML profiles. During creation, any GFF files, Wiggle files and *E*-value table specified in the profile are loaded. Output of the configuration specified in the profile, as well as any specified validation information, is output in TEX format by class methods.

10.6 Other implementations

As the class for handling genomic regions grew in functionality due to requirements in PREsent, this functionality has been made usable by itself through a separate implemented software application, called auxBandTool. The features of this tool are:

- loading of coordinate lists (`C:A...B`), tabulated lists and General Feature Format files;
- output of coordinate lists, tabulated lists, General Feature Format and FASTA format (with regions sequences taken from a specified genome);
- separation of regions in one set based on overlap with regions in a separate set;
- random selection of a specified number of regions;
- finding gene promoter regions based on a genome annotation and specified relative coordinates to each gene Transcription Start Site (TSS);
- merging of regions in specified sets of regions.

This software has been used for treating experimental data, such as separating genic and intergenic regions, as well as for constructing ChIP-based training data.

jPREdictor outputs genome-wide scores in its own format. Thus, a tool was also written for converting genome-wide jPREdictor scores to the Wiggle format.

Chapter 11

Discussion

In this chapter, the thesis ends with a discussion of what has been learned from this work, and a discussion of some possibilities that are left for future work, as there was not time to investigate them.

11.1 Conclusion

A Support Vector Machine can learn to classify sequences based on a number of potential feature sets. One such feature set, the frequencies with which motifs occur, has previously been used for genome-wide prediction of PREs by use of Support Vector Machines [2]. The use of Support Vector Machines also includes considerations such as selecting the Support Vector Machine formulation and parameters, and deciding on how to construct vectors from the training sequences.

In this thesis, additional feature sets based on motif occurrences have been explored. These features make use of relative distances of motifs as well as periodic occurrence of the motifs. As these feature sets were typically larger than the number of training sequences in the original PREdictor [1] training set, and focus was mainly on this training set, they were not considered for genome-wide prediction. Tests of genome-wide prediction using these feature sets and larger training sets will have to be run in the future to determine their effectiveness. The selection of Support Vector Machine parameters has also been considered. The method of using Particle Swarm Optimization for searching for optimal SVM parameters, as was proposed by Lin *et al.* [34], can explore many configurations with a manageable number of training cycles.

The results show that Support Vector Machines can be used with a linear kernel and the *nPair* feature set to predict a larger number of PREs genome-wide than the jPREdictor does with a similar configuration. The decision function of the Support Vector Machine in this case can be reformulated into a sum of weighted feature values, which is similar to that of the PREdictor. Thus, the difference in the number of predictions should be due to a difference in learning from the training sequences when using Support Vector Machines. The results also indicate that the way training vectors are constructed from training sequences can impact generalization and the number of genome-wide predictions made for an E-value of 1.

For moderate values of the SVM cost parameter C and training with the training sequences used by Ringrose *et al.* [1], use of the *nOcc* feature set showed improved generalization to a separate test set when using non-linear kernels. For using the *nPair* feature set, generalization to a separate test set was better with a linear kernel. Noting that the *nPair* feature set with 7 motifs gives 28 features, which is equal to the number of training sequences, this may make sense. The training set may be too small to fully make use of non-linear kernels in this case.

PRESVM has also demonstrated use of the continuous classification value produced by a trained Support Vector Machine, with the addition of a threshold calibrated for a desired E-value, for the genome-wide prediction of PREs. This separates PRESVM from the method developed by Zeng *et al.* [2], the EpiPredictor,

where regions are scored by the number of motif occurrences.

11.2 Future work

During the work of this thesis, multiple potential feature sets have been considered, and a selection of these have been implemented and tested. These feature sets may be interesting to test for genome-wide prediction. There are also feature sets that have been considered but that have not been implemented. This has partly been due to lacking a sufficiently large, carefully constructed training set. The MDP (Mean Distance Proximal) and similar feature sets could be combined with corresponding standard deviation feature sets. Motif occurrence clusters could be considered by making features based on mean positions and standard deviations of positions, and training both on sequence windows and their reverse complements. It could also be interesting to train a secondary SVM to try to distinguish motif occurrence pairs from PRE training sequences from those in non-PRE training sequences, where a feature set may be generated by summing output values for each motif pair and used with a primary SVM.

There is also potential for tuning feature parameters. For periodic feature sets, combinations of different phase shifts and periodicities could be investigated. The work by Lin *et al.* [34] showed that features can be selected at the same time as the Support Vector Machine parameters when using Particle Swarm Optimization. However, parameters of individual features, such as distance cutoffs and the frequencies and phases of periodic feature sets could also be tuned using Particle Swarm Optimization. Even high-level parameters, such as the kernels and the approach to sampling training sequence vectors could potentially be combined with Particle Swarm Optimization.

Some aspects of training could be investigated by use of simulation, such as the impact of varying training sequence lengths. PRESVM could be trained using a larger set of motifs. The use of k-mers could be interesting for larger training data. Motifs defined as Position-Specific Scoring Matrices, as is supported by the jPREdictor [18], could be interesting to test with Support Vector Machines. It may be interesting to apply PRESVM to the prediction of PREs in other species. It may also be interesting to apply PRESVM to other genome-wide prediction tasks in which sequence motifs may be used. Finally, there is also room for further refinement of the software.

List of Figures

List of Tables

List of Algorithms

List of Definitions

Notation overview

- $|x|$: Absolute of a real value x.
- $A \wedge B$: Logical conjunction.
- $A \vee B$: Logical disjunction.
- $A \cap B$: Intersection of sets A and B.
- $A \cup B$: Union of sets A and B.
- $A \times B$: Cartesian product of sets A and B.
- $\min S$: Gives the minimum value in a set S.
- $\max S$: Gives the maximum value in a set S.
- S/x: A set S with the element x excluded.
- $|S|$: Cardinality of a set S.
- $\vec{a} = (a_1, ..., a_n)$: Vector.
- $\vec{a} \cdot \vec{b}$: Dot product.
- $f : A \to B$: Function mapping elements of A to elements of B.
- $\hat{c}(x)$: Approximation of function $c(x)$.
- $\oplus$: Positive label.
- $\ominus$: Negative label.
- $\mathbb{B} = \{\oplus, \ominus\}$: Binary label set.
- $[x]_\pm$: Assignment of binary label. If $x > 0$ then $[x]_\pm = \oplus$. Otherwise, $[x]_\pm = \ominus$.
- TP, TN, FP, FN: True Positives, True Negatives, False Positives and False Negatives, respectively.
- `bp`: Base pair unit.
- $\mathbb{S}_{\texttt{nt}}$: Set of all nucleotide sequences.
- $S : A...B$: Substring of $S \in \mathbb{S}_{\texttt{nt}}$ from nucleotide index A up to and including nucleotide index B.
- $\lambda(S)$: Length of sequence S.
- $O(s)$: Set of all motif occurrences in sequence s.
- $\tau(o)$: Type of motif occurrence o.
- $O_m(s)$: Set of all occurrences of motif m in sequence s.

- $\pi_\alpha(o)$: Index of first nucleotide of motif occurrence o.
- $\pi_\beta(o)$: Index of nucleotide after the last of motif occurrence o.
- $\pi_\gamma(o)$: Position of center nucleotide of motif occurrence o. $\pi_\gamma(o) = \frac{\pi_\alpha(o)+\pi_\beta(o)}{2}$. $\pi_\gamma(o) \in \mathbb{R}$.
- $\delta_\alpha(o_1, o_2)$: Distance between ends of motif occurrences o_1 and o_2.
- $\delta_\gamma(o_1, o_2)$: Distance between centers of motif occurrences o_1 and o_2.
- $f'(x)$: Derivative of function $f(x)$.
- $\frac{\partial f}{\partial x}$: Partial derivative of f with respect to x.
- $\nabla f(\vec{x})$: Gradient of $f(\vec{x})$.

Bibliography

[1] L. Ringrose, M. Rehmsmeier, J.-M. Dura, and R. Paro, "Genome-wide prediction of polycomb/trithorax response elements in drosophila melanogaster," *Developmental Cell*, pp. 759–771, 2003.

[2] J. Zeng, B. D. Kirk, Y. Gou, Q. Wang, and J. Ma, "Genome-wide polycomb target gene prediction in drosophila melanogaster," *Nucleic Acids Research*, 2012.

[3] D. L. Nelson and M. M. Cox, *Lehninger: Principles of Biochemistry, fifth edition.* W.H. Freeman and Company, 2008.

[4] S. J. Marygold, P. C. Leyland, R. L. Seal, J. L. Goodman, J. Thurmond, V. B. Strelets, R. J. Wilson, and the FlyBase consortium, "FlyBase: Improvements om the bibliography," *Nucleic Acids Research*, vol. 41, 2013.

[5] I. Eidhammer, I. Jonassen, and W. R. Taylor, *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis.* John Wiley & Sons, Ltd., 2004.

[6] L. Ringrose and R. Paro, "Polycomb/trithorax response elements and epigenetic memory of cell identity," *Development*, 2007.

[7] J. Müller and J. A. Kassis, "Polycomb response elements and targeting of polycomb group proteins in drosophila," *Elsevier*, 2007.

[8] L. Ringrose, "Polycomb comes of age: genome-wide profiling of target sites," *Elsevier*, 2007.

[9] C. E. Horak and M. Snyder, "ChIP-chip: A genomic approach for identifying transcription factor binding sites," *Elsevier*, 2002.

[10] E. R. Mardis, "ChIP-seq: welcome to the new frontier," *Nature Methods*, 2007.

[11] Y. B. Schwartz, T. G. Kahn, D. A. Nix, X.-Y. Li, R. Bourgon, M. Biggin, and V. Pirrotta, "Genome-wide analysis of polycomb targets in drosophila melanogaster," *Nature Genetics*, 2006.

[12] O. Katarzyna, L. Guitiérrez, J. Gagneur, C. Girardot, A. K. Sengupta, E. E. Furlong, and M. Jürg, "Dynamic regulation by polycomb group protein complexes controls pattern formation and the cell cycle in drosophila," *Developmental Cell*, 2008.

[13] B. Schuettengruber, M. Ganapathi, B. Leblanc, M. Portoso, R. Jaschek, B. Tolhuis, M. v. Lohuizen, A. Tanay, and G. Cavalli, "Functional anatomy of polycomb and trithorax chromatin landscapes in drosophila embryos," *PLoS Biology*, 2009.

[14] Y. B. Schwartz, T. G. Kahn, P. Stenberg, K. Ohno, R. Bourgon, and V. Pirrotta, "Alternative epigenetic chromatin states of polycomb target genes," *PLoS Genetics*, 2010.

[15] D. Enderle, C. Beisel, M. B. Stadler, M. Gerstund, P. Athri, and R. Paro, "Polycomb preferentially targets stalled promoters of coding and noncoding transcripts," *Genome Research*, 2011.

[16] L. Ringrose and R. Paro, "Epigenetic regulation of cellular memory by the polycomb and trithorax group proteins," *Annual Reviews*, 2004.

[17] J. H. Ho, E. Bishop, P. V. Karchenko, N. Nègre, K. P. White, and P. J. Park, "Chip-chip versus chip-seq: Lessons for experimental design and data analysis," *BMC Genomics*, 2011.

[18] T. Fiedler and M. Rehmsmeier, "jpredictor: a versatile tool for the prediction of *cis*-regulatory elements," *Nucleic Acids Research*, 2006.

[19] A. Hauenschild, L. Ringrose, C. Altmutter, R. Paro, and M. Rehmsmeier, "Evolutionary plasticity of polycomb/trithorax response elements in *Drosophila* species," *PLoS Biology*, 2008.

[20] X. Xiao, Z. Li, H. Liu, J. Su, F. Want, X. Wu, H. Liu, Q. Wu, and Y. Zhang, "Genome-wide identification of polycomb target genes in human embryonic stem cells," *Gene*, 2013.

[21] L. Hamel, *Knowledge Discovery with Support Vector Machines*. Wiley Series on methods and applications in data mining, John Wiley & Sons, Inc., Hoboken, New Jersey, 2009.

[22] T. M. Mitchell, *Machine Learning*. McGraw-Hill Book Co, 1997.

[23] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," 2010. National Taiwan University, Taipei 106, Taiwan.

[24] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[25] P. Baldi, S. Brunak, Y. Chauvin, C. A. F. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification: an overview," *Bioinformatics*, 2000.

[26] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," 2006.

[27] G. Hripcsak and A. S. Rothschild, "Agreement, the f-measure and reliability in information retrieval," *J Am Med Inform Assoc*, pp. 12:296–298, 2005.

[28] T. Fawcett, "An introduction to roc analysis," *Elsevier*, 2005.

[29] M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American Society of Information Science*, pp. 45(1):12–19, 1994.

[30] E. Garbarine, J. DePasquale, V. Gadia, R. Polikar, and G. Rosen, "Information-theoretic approaches to svm feature selection for metagenome read classification," *Elsevier*, 2010.

[31] J. W. Nicol, G. A. Helt, S. G. J. Blanchard, A. Raja, and A. E. Loraine, "The integrated genome browser: free software for distribution and exploration of genome-scale datasets," *Bioinformatics*, 2009.

[32] "UCSC genome bioinformatics." `https://genome.ucsc.edu/FAQ/FAQformat.html`. Accessed: 2013-05-13.

[33] R. A. Adams and C. Essex, *Calculus: A complete course*. Pearson Canada Inc., Toronto, Ontario, 2010.

[34] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Elsevier*, 2007.

[35] M. Kalicinski, "RapidXml." `http://rapidxml.sourceforge.net/`, 2009. Accessed: 2013-05-08.

[36] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[37] N. D. Jay, S. Papillon-Cavanagh, C. Olsen, G. Bontempi, and B. Haibe-Kains, "mrmre: an r package for parallelized mrmr ensemble feature selection," *Submitted*, p. ., 2012.

[38] H. Chen and P. C. Boutros, "VennDiagram: a package for the generation of highly-customizable venn and euler diagrams in R," *BMC Bioinformatics*, vol. 12:35, 2011.