Co-Training of
Version Space Support Vector Machines

Stijn Vanderlooy

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Computer Science: section mathematics

transnational University Limburg

Supervisors:
dr. R.L. Westra
dr. E.N. Smirnov
dr. I.G. Sprinkhuizen-Kuyper

2004-2005
I am grateful to have the opportunity to thank many people who have contributed in different ways to this thesis.

I would like to thank dr. R.L. Westra and dr. E.N. Smirnov for making me enthusiastic about a wide variety on topics of data mining and machine learning. I am very grateful to dr. R.L. Westra for giving me support and advice during the last year of my study. I also have to thank dr. I.G. Sprinkhuizen-Kuyper and dr. E.N. Smirnov for their enthusiasm, inspiration and effort to explain things clearly to me.

I wish to thank all my teachers. They made for me computer science and mathematics interesting, fascinating and fun.

I am indebted to my student colleague and friend Goele Hollanders. She is always with me in difficult times and I would like to thank her for the emotional support, friendship and caring! To my other friends, thanks for the support and the recreative times when I was not working for school.

Lastly, and most importantly, I wish to thank my parents, Val`ere Vanderlooy and Rita Hendrikx and my sister Nathalie Vanderlooy. They supported me, taught me, loved me and asked at the right moments “Is it finished yet?”. I dedicate this thesis to them.
Abstract

One of the main problems when machine-learning classifiers are employed in practice is to determine whether classifications assigned to new instances are reliable. Recently, version spaces were proposed to be considered as an approach to reliable instance classification. The unanimous-voting classification rule of version spaces does not misclassify, i.e., instance classifications become reliable.

Version spaces were proposed to be implemented using support vector machines. The resulting combination is called version space support vector machines (VSSVMs). The experiments with VSSVMs showed 100% accuracy at the cost of coverage reduction.

The problem of coverage reduction is a serious problem for VSSVMs when they are applied in practice. In this thesis we propose to solve this problem by applying co-training on VSSVMs. The key idea is to generate informative instances for the VSSVMs and then to label these instances by those VSSVMs that can classify them. The new-labeled instances are added to the training data and all the VSSVMs are retrained. In this way VSSVMs become smaller, i.e., their coverage grows.

We implement four models for co-training of VSSVMs. The first two models are based on the basic co-training algorithm. The experiments of these two models show that co-training in its original form can harm the accuracy and coverage. Our analysis reveals that these results are due to the fact that the version-space algebra is not preserved for VSSVMs. The last two models partially overcome this problem. These models learn so-called on-top VSSVMs from new-labeled data that classify only those instances that cannot be classified by the base VSSVMs. In this way the problem with the version-space algebra is left to the on-top VSSVMs.

We show that the problem with version-space algebra cannot be solved completely by adding many-level on-top VSSVMs. Therefore, we propose to apply the volume extension approach. The approach is to extend the volume of the co-trained VSSVMs so that the instance misclassification is blocked and the coverage is maximized.
Contents

1 Introduction ........................................... 5
   1.1 Approaches to Reliable Instance Classification ........ 5
      1.1.1 Bayesian Framework ............................. 6
      1.1.2 Meta Learning ..................................... 6
      1.1.3 Typicalness Framework .......................... 6
      1.1.4 Version Space Support Vector Machines ........ 6
   1.2 Co-Training .......................................... 7
   1.3 Problem statement .................................... 7
   1.4 Thesis outline ....................................... 7

2 Combining Support Vector Machines and Version Spaces ..... 9
   2.1 Task of Reliable Instance Classification ............. 9
   2.2 Support Vector Machines ................................ 10
      2.2.1 Hyperplanes and Margins ......................... 10
      2.2.2 Training of Linear Support Vector Machines ...... 11
      2.2.3 Training of Non-linear Support Vector Machines .. 12
   2.3 Version Spaces ....................................... 14
      2.3.1 Definition ......................................... 14
      2.3.2 Classification Rule ................................ 14
      2.3.3 Analysis of the Classification Rule .............. 16
      2.3.4 Volume-Extension Approach ....................... 17
   2.4 Version Space Support Vector Machines ............... 18
      2.4.1 Definition ......................................... 18
      2.4.2 Classification Algorithm .......................... 19
      2.4.3 Example ............................................ 20
      2.4.4 Volume-Extension Approach ....................... 20
   2.5 Conclusion of Chapter 2 ................................ 22

3 Co-Training .............................................. 23
   3.1 Algorithm ............................................. 23
   3.2 Basic Assumptions ..................................... 24
3.3 Limitations ......................................................... 25
  3.3.1 Labeled Data ............................................... 25
  3.3.2 Selection ....................................................... 26
3.4 Discussion .......................................................... 27

4 Co-Training of Version Space Support Vector Machines .... 28
  4.1 Informative Instances ........................................ 28
    4.1.1 Definition .................................................. 29
    4.1.2 Optimization Formulation ............................... 30
    4.1.3 Optimization Algorithm ................................. 30
    4.1.4 Initial Guess ................................................. 31
  4.2 Co-Training Models ............................................ 32
    4.2.1 First Model ............................................... 32
    4.2.2 Second Model .............................................. 45
    4.2.3 Third Model ............................................... 46
    4.2.4 Fourth Model ............................................... 51

5 Conclusion ........................................................... 54
  5.1 Thesis Conclusion ............................................. 54
  5.2 Future Research ................................................ 55
Chapter 1

Introduction

In the last ten years machine-learning classifiers have been applied to various classification problems. Nevertheless, only few classifiers have been employed in real applications, especially in critical domains. The main reason is that it is difficult to determine whether a classification assigned to a particular instance is reliable or not. Solving the problem of reliable instance classification is an important task to integrate machine-learning classifiers in diverse real-life applications.

There are several approaches to the task of reliable instance classification. An overview of the main approaches to the task of reliable instance classification is given in section 1.1. One of the most successful among these approaches is the version space support vector machine. Section 1.2 introduces co-training. A combination of version space support vector machines and co-training leads to the problem statement of this thesis formulated in section 1.3. Section 1.4 gives the thesis outline.

1.1 Approaches to Reliable Instance Classification

There are several approaches to reliable instance classification [2, 12, 15, 20, 23, 25]. Most of these approaches output confidence values for each instance classification. These confidence values are used in a filter mechanism. When the confidence of a classification is above a certain threshold a class label is assigned to the instance. Otherwise the instance is left unclassified.

The main approaches to reliable instance classification are the Bayesian framework [15], meta-learning [2], the typicalness framework [20], and version space support vector machines [25]. These approaches are briefly discussed in the next four subsections.
1.1. APPROACHES TO RELIABLE INSTANCE CLASSIFICATION

1.1.1 Bayesian Framework

The Bayesian framework [14] is one of the first used for reliable instance classification. The reason is that posterior class probabilities can be considered as estimates for the reliability of instance classifications. The correctness of the confidence values depends on the correctness of the prior probabilities. The latter are difficult to estimate. Therefore the Bayesian framework is often misleading in practice.

1.1.2 Meta Learning

Meta learning offers a meta approach that learns a meta classifier to predict whether a particular instance classification is (in)correct [2]. An extension of this approach dynamically chooses the most correct classifier in an ensemble of classifiers [23].

The main problem with the meta classifier approach is that it is impossible to learn a correct meta classifier. Therefore the approach can result in instance misclassifications.

1.1.3 Typicalness Framework

The typicalness framework is introduced in [11, 13, 20]. It assumes that the instances and their class labels are independently and identically distributed (iid). Given training data and a test instance, the confidence prediction for each possible label of this instance is indicated by the likelihood of the sequence consisting of the data and the instance. The more typical this sequence is in the iid sense, the higher is the confidence in the predicted label.

To measure the typicalness of a sequence, a function is needed that measures the strangeness of a specific labeled instance in the sequence. This function has to be designed for each type of classifiers used. Therefore the typicalness framework depends heavily on the machine-learning algorithm used.

1.1.4 Version Space Support Vector Machines

Recently version spaces were proposed to be considered as an approach to reliable instance classification [25]. The key idea is to construct version spaces containing the hypothesis of the target concept or its close approximations. In this way the unanimous-voting classification rule of version spaces does not misclassify, i.e., instance classifications become reliable.
1.2. CO-TRAINING

The unanimous-voting rule can be implemented by testing version spaces for collapse. This can be done by any learning algorithm. In the corresponding paper support vector machines are used. The combination is called version space support vector machines. Experiments show accuracy of 100% at the cost of a reduction of the coverage.

Version space support vector machines play a central role in this thesis and will be explained in detail in section 2.4.

1.2 Co-Training

Co-training [4, 16, 19, 18, 21] is a technique to boost the accuracy of a set of classifiers. To achieve this goal unlabeled instances are used. Each classifier assigns labels to unlabeled instances for which it is certain that the prediction is correct. The new labeled instances are then added to the training data and the classifiers are retrained.

Co-training is discussed in detail in chapter 3.

1.3 Problem statement

Version space support vector machines are a successful approach to reliable instance classification. However when they are applied in practice we face the coverage reduction problem: many instances are left unclassified. To overcome this problem we propose to apply co-training on version space support vector machines. The problem statement of this thesis reads as follows:

Is it possible to apply co-training for version space support vector machines to improve their coverage?

1.4 Thesis outline

This thesis is outlined as follows. Chapter 2 provides necessary background. It formalizes the classification task and the task of reliable instance classification. In addition the chapter provides a brief introduction to the approaches used in this thesis: support vector machines, version spaces, and version space support vector machines. It is shown that version space support vector machines are a successful approach to reliable instance classification.

Chapter 3 reviews co-training. A formal framework for co-training is briefly given and the limitations are summarized.
Chapter 4 provides our solution to the problem statement. The solution is to co-train version space support vector machines in order to improve their coverage. Four models of co-training are proposed, experimented and analyzed.

The last chapter 5 summarizes the results and provides an answer to the problem statement.
Chapter 2

Combining Support Vector Machines and Version Spaces

This chapter provides an overview of three main approaches used in this thesis. To introduce the approaches the task of reliable instance classification is formalized in section 2.1. Then in sections 2.2, 2.3 and 2.4 we consider in detail support vector machines, version spaces and version space support vector machines. In the last section 2.5 we show the importance of solving the coverage reduction problem for applicability of version space support vector machines.

2.1 Task of Reliable Instance Classification

The classification task is defined as follows. Assume that we have an instance space $X$ and some set of possible classifications $Y$. A training instance is represented as $(x, y)$ where $x \in X$ is the feature vector and $y \in Y$ is the class label. In this thesis we consider binary classification tasks, i.e., $Y = \{+1, -1\}$.

An instance $x_i$ is positive (negative) if and only if $y_i = +1$ ($y_i = -1$). The set of all positive (negative) training instances is denoted by $I^+$ ($I^-$). Given this training data $\langle I^+, I^- \rangle$ of a target concept and a hypothesis space $H$, the classification task is to find a hypothesis $h : X \rightarrow Y$ that correctly classifies unseen instances in $X$.

The task of reliable instance classification extends the classification task. The task is to find a hypothesis $h$ that correctly classifies unseen instances in $X$ and when the correct classification cannot be determined $h$ outputs 0, i.e., the instance cannot be unclassified.
2.2 Support Vector Machines

Support Vector Machines (SVMs) are rooted in statistical learning theory and were first introduced by Vapnik and co-workers in [5]. A good tutorial is provided by Burges [6]. The SVM is a training algorithm for classification and regression tasks. Subsection 2.2.1 introduces the hypothesis space of SVMs and shows the principle of hypothesis selection. Subsections 2.2.2 and 2.2.3 describe the training algorithm and its parameters.

2.2.1 Hyperplanes and Margins

The hypothesis space of SVMs is the space of hyperplanes. Given training data, a separating hyperplane is a hyperplane that separates the positive and the negative training instances, i.e., the training error is zero. Figure 2.1 shows two linear separating hyperplanes. Because the instance space is $\mathbb{R}^2$ the hyperplanes are lines.

If $d_+$ is defined as the shortest distance from a hyperplane to a positive instance and if $d_-$ is defined accordingly, a shaded area can be created for each hyperplane. The width of this area is the margin of the hyperplane and is formally defined as $d_+ + d_-$. In the figure hyperplane 1 has the largest margin. The hyperplane with the largest margin is considered to be the least complex one because this hyperplane can absorb the most noise in the data. To illustrate this finding, consider figure 2.2 which is the same as figure 2.1 except that only two instances are shown. Furthermore a circle is around these instances. This circle represents noise: the real (unknown) position of the instance is somewhere inside the circle. It is clear that hyperplane 1 can
absorb the most noise which means that it is the least complex hyperplane and therefore it is preferable.

Figure 2.2: Hyperplane 1 can absorb more noise than hyperplane 2.

SVMs are learning machines that search for hyperplanes with maximum margin. These hyperplanes are considered as optimal. Below we consider linear and non-linear SVMs.

2.2.2 Training of Linear Support Vector Machines

Linear SVMs are SVMs that try to find an optimal linear hyperplane. An example of a SVM optimal hyperplane is given in figure 2.3. Black dots are the positive training instances and circles represent the negative training instances. The equation of the optimal linear hyperplane is $w \cdot x + b = 0$ with $w$ the normal and $b$ is called the threshold. Note that the hyperplane goes through the origin when $b = 0$. Given this equation and some positive constant $a$, all black instances $x_i$ satisfy $w \cdot x_i + b \geq a$ and all white instances $x_i$ satisfy $w \cdot x_i + b \leq -a$.

The instances in figure 2.3 that lie on hyperplanes $H_1$ and $H_2$ are called support vectors. The support vectors are the only instances that determine the position of the optimal hyperplane.

When the optimal coefficients $w$ and $b$ have been found a new instance $x$ is classified by the SVM optimal hyperplane as follows:

$$y = sgn(w \cdot x + b)$$

The above classification rule means that the SVM optimal hyperplane is oriented: the class label depends on which side of the hyperplane the instance lies.
2.2. SUPPORT VECTOR MACHINES

The problem of finding the optimal value for the normal $w$ is a convex quadratic programming problem. The solution of the problem depends on the training data and a constant $C$. To find a linear hyperplane the problem formulation is:

$$
\max \quad -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j + \sum_{i=1}^{l} \alpha_i \\
\text{s.t.} \quad \sum_{i=1}^{l} \alpha_i y_i = 0 \\
0 \leq \alpha_i \leq C \forall i
$$

(2.2)

The alphas are the weights associated with the training instances. All instances with non-zero weights are the support vectors. The constant $C$ determines the trade-off between the margin and the amount of training errors of the SVM optimal hyperplane. When $C$ increases from value $C_1$ to $C_2$ the margin will decrease and the amount of classification errors will generally go down. The constant $C$ is therefore called the complexity-error trade-off parameter.

The solution of the normal $w$ is given by $\sum_{i=1}^{l} \alpha_i y_i x_i$. Substituting this value into the classification formula 2.1 gives $\text{sgn}(\sum_{i=1}^{l} \alpha_i y_i x_i \cdot x + b)$ as the rule for the class label for instance $x$. The value of the threshold $b$ can be found by the Karush-Kuhn-Tucker conditions (see [6]).

2.2.3 Training of Non-linear Support Vector Machines

Non-linear SVMs are proposed for the classification task when the training data are not linearly separable. Non-linear SVMs are derived from the
2.2. SUPPORT VECTOR MACHINES

above quadratic programming problem (problem 2.2). The problem can be
generalized to find a non-linear hypothesis of the data by using the Kernel

The training instances only appear as dot products in optimization prob-
lem 2.2. This has led to the idea of using a mapping \( \Phi : X \rightarrow \mathcal{H} \) where \( \mathcal{H} \) is a large dimensional (possible infinite) Euclidian space. In this new space

One problem can arise with the computations in \( \mathcal{H} \) because the higher the order of this space, the more time is consumed for the calculations. SVMs

Of course only kernel functions should be used that are dot products in \( \mathcal{H} \). Mercer’s condition states when this is valid \([6]\). Two widely used kernels

- Gaussian Radial Basis Function (RBF): \( e^{-\gamma||x-y||^2} \)
- Polynomial: \( ((x \cdot y) + \theta)^E \)

where \( \gamma, \theta \) and exponent \( E \) are parameters. The value of \( \theta \) is always 1 in this thesis. If we speak of SVM parameters only \( \gamma \) and \( E \) are considered. \( \gamma \) determines the level of proximity between any two points in the instance space \( X \). When \( \gamma \) increases from \( \gamma_1 \) to \( \gamma_2 \), any two points become more dissimilar. Thus, it becomes easier to separate them by the SVM optimal hyperplane with the same (fixed) \( C \) parameter. The parameter \( E \) of a polynomial kernel

Given a kernel and the kernel parameter \( p \) (\( \gamma \) or \( E \)) the notation \( H(p) \) is used to denote the hypothesis space of the SVM. \( H(p) \) contains all possible oriented hyperplanes that can be build with the specified kernel. The term

\[
\max -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(x_i) \cdot \Phi(x_j) + \sum_{i=1}^{l} \alpha_i (2.3)
\]

s.t. \( \sum_{i=1}^{l} \alpha_i y_i = 0 \)
\[
0 \leq \alpha_i \leq C \ \forall i \]
2.3. VERSION SPACES

$h(p, C, (I^+, I^-))$ represents a SVM hyperplane build on training data $(I^+, I^-)$ using kernel parameter $p$ and the constant $C$. The classification of a new instance $x$ is found by:

$$h(p, C, (I^+, I^-))(x) = \text{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b\right)$$

(2.4)

2.3 Version Spaces

Version spaces are an established approach to the classification task [14, 15, 24]. Version spaces are defined in subsection 2.3.1. The version space classification rule is explained in subsection 2.3.2. A detailed analysis of the rule under different conditions is given in subsection 2.3.3. Subsection 2.3.4 explains the volume-extension approach that solves the problems of the classification rule. In the end it is concluded that version spaces can be considered as an approach to reliable instance classification.

2.3.1 Definition

A hypothesis $h \in H$ is consistent with training sets $I^+$ and $I^-$ if and only if it satisfies the consistency predicate $\text{cons}(h, (I^+, I^-))$ defined as follows:

$$\text{cons}(h, (I^+, I^-)) \iff (\forall x_i \in I^+ \cup I^-)(y_i = h(x_i))$$

(2.5)

In other words, a hypothesis is consistent with the training data if it correctly classifies all these instances. Note that this does not mean that all instances can be classified correctly because unseen instances (non-training instances) are not considered in this definition.

The version space with respect to hypothesis space $H$ and training data $(I^+, I^-)$ is defined as:

$$\text{VS}(I^+, I^-) = \{ h \in H | \text{cons}(h, (I^+, I^-)) \}$$

(2.6)

2.3.2 Classification Rule

The classification rule of version spaces is the unanimous-voting rule. The rule is specified as follows. Given a version space $\text{VS}(I^+, I^-)$ an instance $x$ receives a classification $y \in Y \cup \{0\}$ as follows:

$$y = \begin{cases} 
+1 & \text{if } (\text{VS}(I^+, I^-) \neq \emptyset) \land (\forall h \in \text{VS}(I^+, I^-))(h(x) = +1) \\
-1 & \text{if } (\text{VS}(I^+, I^-) \neq \emptyset) \land (\forall h \in \text{VS}(I^+, I^-))(h(x) = -1) \\
0 & \text{otherwise.}
\end{cases}$$
The unanimous-voting rule assigns class $+1$ ($-1$) to the instance $x$ if $\text{VS}(I^+, I^-)$ is non-empty and all hypotheses $h$ in $\text{VS}(I^+, I^-)$ assign the same class $+1$ ($-1$) to the instance. In all other cases, the class is unknown.

The volume of a version space $\text{VS}(I^+, I^-)$, denoted by $V(\text{VS}(I^+, I^-))$, is the set of all instances that cannot be classified by the version space.

An example of an instance that is left unclassified is given in figure 2.4. The hypothesis space $H$ is the space of all oriented lines in $\mathbb{R}^2$. The blue and red lines are two consistent hyperplanes. Suppose that the red instance $x$ at coordinates $(0.5, 2)$ has to be classified. The blue line classifies the instance as positive and the red line as negative. Thus it is not possible to classify $x$ reliably because some hyperplanes classify $x$ as positive and other hyperplanes classify $x$ as negative. Note that every possible instance in the grey area is an instance that is left unclassified. The grey area is the volume of the version space.

![Figure 2.4: The volume of the version space (given in grey).](image)

Theorems 1 and 2 given below show that the unanimous-voting rule can be implemented if version spaces can be tested for collapse [9, 24]. Theorem 1 states that if the version space $\text{VS}(I^+, I^-)$ is non-empty, then all the hypotheses $h \in \text{VS}(I^+, I^-)$ assign class $+1$ to an instance $x$ if and only if the version space $\text{VS}(I^+, I^- \cup \{x\})$ is empty. Analogously, theorem 2 states that if the version space $\text{VS}(I^+, I^-)$ is non-empty, then all the hypotheses $h \in \text{VS}(I^+, I^-)$ assign class $-1$ to an instance $x$ if and only if the version...
space \( VS(I^+ \cup \{x\}, I^-) \) is empty.

**Theorem 1.** If \( VS(I^+, I^-) \) is non-empty, then:

\[
(\forall x)((\forall h \in VS(I^+, I^-))(h(x) = +1) \leftrightarrow VS(I^+, I^- \cup \{x\}) = \emptyset).
\]

**Theorem 2.** If \( VS(I^+, I^-) \) is non-empty, then:

\[
(\forall x)((\forall h \in VS(I^+, I^-))(h(x) = -1) \leftrightarrow VS(I^+ \cup \{x\}, I^-) = \emptyset).
\]

The problem to test version spaces for collapse is equivalent to the consistency problem. The consistency problem is to determine the existence of a hypothesis in the hypothesis space that is consistent with training data. Thus, the unanimous-voting rule of version spaces can be implemented by any algorithm for the consistency problem [8, 9, 10, 24].

Most of the learning algorithms are imperfect consistency algorithms. This means that for some data they cannot find consistent hypotheses when these hypotheses are in the hypothesis space \( H \). For these algorithms version spaces are defined so that they are non-empty if and only if the algorithm can find a consistent hypothesis. In this way not only the hypothesis space but the algorithm parameters \( P \) as well determine when version spaces are empty. This means that the unanimous-voting rule of version spaces is approximated by the algorithm with respect to parameters \( P \).

**2.3.3 Analysis of the Classification Rule**

The version space unanimous-voting rule can be analyzed along two dimensions. The first dimension is the expressiveness of the hypothesis space. A hypothesis space is expressive if it contains the target function. The second dimension is the noise level of the training data. Combining these two dimensions results in four different cases of the unanimous-voting rule.

**Case 1: Non-noisy Training Data and Expressive Hypothesis Space**

Since the hypothesis space \( H \) is expressive the hypothesis \( h_t \) of the target concept is in \( H \). Since the training data are noise-free the target function \( h_t \) is consistent with the training data \( ⟨I^+, I^-⟩ \). Furthermore \( h_t \) belongs to the version space \( VS(I^+, I^-) \). In this way, if an instance is classified with the unanimous-voting classification rule applied on \( VS(I^+, I^-) \), the instance receives the classification assigned of \( h_t \). This means that the version space unanimous-voting rule outputs only reliable instance classifications.
2.3. VERSION SPACES

Case 2: Noisy Training Data
If the training data are noisy, then the training set \( I^+ \) (\( I^- \)) is a union of a noise-free set \( I^+_n \) (\( I^-_n \)) and a noisy set \( I^+_n \) (\( I^-_n \)). The noisy sets \( I^+_n \) and \( I^-_n \) cause removal of the set \( NVS = \{ h \in VS(I^+_f, I^-_f) | \neg cons(h, (I^+_n, I^-_n)) \} \) from the version space \( VS(I^+_f, I^-_f) \). The resulting version space \( VS(I^+, I^-) \) will continue to classify instances that are classified by the version space \( VS(I^+_f, I^-_f) \), but it will err on at least one instance in the volume of \( NVS \).

Case 3: Inexpressive Hypothesis Space
If the hypothesis space \( H \) is inexpressive, i.e., the hypothesis \( h_t \) of the target concept does not belong to \( H \), then it is possible that the hypotheses in the version space \( VS(I^+, I^-) \) do not approximate the target concept well. This means that there exists at least one instance \( x \) that is incorrectly classified by all the hypotheses in the version space \( VS(I^+, I^-) \).

Case 4: Noisy Training Data and Inexpressive Hypothesis Space
From the analysis of cases 2 and 3, it is easy to see that if the training data are noisy and the hypothesis space \( H \) is not expressive, then:

- the version space \( VS(I^+, I^-) \) misclassifies at least one instance in the volume of the set \( NVS \); and
- the version space \( VS(I^+, I^-) \) can misclassify instances that are in the volume of the version space \( VS(I^+_f, I^-_f) \) if the hypotheses in \( VS(I^+_f, I^-_f) \) do not approximate the target concept well.

2.3.4 Volume-Extension Approach
The volume-extension approach was proposed in [25] to overcome cases 2, 3, and 4. Assume a hypothesis space \( H \) and an implementation of the unanimous-voting rule based on an imperfect consistency algorithm with parameters \( P \). Then, if a version space \( VS(I^+, I^-) \) misclassifies instances, the approach redefines the hypothesis space \( H \) and/or the parameters \( P \) so that the volume of the new version space \( VS'(I^+, I^-) \) grows and blocks instance misclassifications. Below we consider this approach for all the three problematic cases in subsection 2.3.3:

- case 2: since the volume of \( NVS \) is the region of instance misclassification for the version space \( VS(I^+, I^-) \), we redefine the hypothesis space
2.4. VERSION SPACE SUPPORT VECTOR MACHINES

$H$ or the parameters $P$ so that the volume of the new version space $VS'(I^+, I^-)$ comprises maximally the volume of $NVS$;

- case 3: if there is an instance that is misclassified by the version space $VS(I^+, I^-)$, then we redefine $H$ or $P$ so that the new version space $VS'(I^+, I^-)$ includes at least one hypothesis that does classify the instance as the target concept. In this case, the unanimous-voting rule guarantees that the instance will not be classified;

- case 4: since case 4 is a union of cases 2 and 3, the explanations of the solution for these two cases hold here as well.

To apply the approach by redefining hypothesis space $H$ to a new hypothesis space $H'$ we have to guarantee for arbitrary data $\langle I^+, I^- \rangle$ that if there is a consistent hypothesis $h \in H$, then there is a consistent hypothesis $h' \in H'$. It can then be proven that the volumes of the new version spaces $VS'(I^+, I^-)$ then comprise those of the version spaces $VS(I^+, I^-)$ [25].

To apply the volume-extension approach by redefining the parameters $P$ of the imperfect consistency algorithm in the unanimous-voting-rule implementation we have to find dependencies checking for each two parameter sets $P$ and $P'$ if the volumes of version spaces $VS'$ comprise those of version spaces $VS$. In section 2.4.4 we provide an example for SVMs in the context of version space support vector machines.

Cases 2, 3, and 4 presented in subsection 2.3.3 are problems for version spaces. These cases can be overcome by applying the volume-extension approach. This result and case 1 allow to conclude that version spaces can be considered as an approach to reliable instance classification.

2.4 Version Space Support Vector Machines

Version space support vector machines (VSSVMs) are an approach to reliable instance classification [26, 25]. They are a combination of version spaces and SVMs. Subsection 2.4.1 formally defines VSSVMs. The classification rule is explained in subsection 2.4.2. An example is given in subsection 2.4.3. Finally, subsection 2.4.4 shows how the volume-extension approach can be applied for VSSVMs.

2.4.1 Definition

VSSVMs are version spaces that can be tested for collapse with SVMs. The hypothesis space of VSSVMs is the hypothesis space $H(p)$ of SVMs. SVMs
2.4. VERSION SPACE SUPPORT VECTOR MACHINES

Input: An instance \( x \) to be classified
- Training data sets \( I^+ \) and \( I^- \)
- Kernel and its parameter \( p \) (optional)
- The parameter \( C \) of SVM

Output: classification of \( x \)

Build a hyperplane \( h(p, C, \langle I^+, I^- \rangle) \);
if \( \neg \text{cons}(h(p, C, \langle I^+, I^- \rangle), \langle I^+, I^- \rangle) \) then return 0;
Build a hyperplane \( h(p, C, \langle I^+ \cup \{x\}, I^- \rangle) \);
if \( \neg \text{cons}(h(p, C, \langle I^+ \cup \{x\}, I^- \rangle), \langle I^+ \cup \{x\}, I^- \rangle) \) and
\( \text{cons}(h(p, C, \langle I^+ \cup \{x\}, I^- \rangle), \langle I^+ \cup \{x\}, I^- \rangle) \) then return +1;
if \( \text{cons}(h(p, C, \langle I^+ \cup \{x\}, I^- \rangle), \langle I^+ \cup \{x\}, I^- \rangle) \) and
\( \neg \text{cons}(h(p, C, \langle I^+ \cup \{x\}, I^- \rangle), \langle I^+ \cup \{x\}, I^- \rangle) \) then return -1;
return 0.

Figure 2.5: The classification algorithm of VSSVM.

are not perfect consistency algorithms. The VSSVM is therefore defined as follows. Given \( H(p) \), a constant \( C \) and training data \( \langle I^+, I^- \rangle \) the version space support vector machine \( \text{VS}^p_C(I^+, I^-) \) is:

\[
\begin{cases} \{ h \in H(p) | \text{cons}(h, \langle I^+, I^- \rangle) \} \quad \text{if } \text{cons}(h(p, C, \langle I^+, I^- \rangle), \langle I^+, I^- \rangle) \\ \emptyset \quad \text{otherwise.} \end{cases}
\] (2.7)

Version-space algebra is not preserved due to this definition [8, 9, 24]. This implies that theorems 1 and 2 do not always hold for VSSVMs. To approximate instance classification with VSSVMs it is assumed that the theorems hold so that definition 2.7 can be used for collapse testing.

VSSVMs are version spaces. Hence, the inductive bias of VSSVMs is the restriction bias [15, 24]. The kernel parameter \( p \) defines the hypothesis space \( H(p) \) of VSSVMs and the parameter \( C \) determines when VSSVMs are empty in \( H(p) \). Hence, the restriction bias of VSSVMs is controlled by these two parameters.

To apply SVM the training data \( \langle I^+, I^- \rangle \) are only needed. Hence, the training data are the version-space representation of VSSVMs.

2.4.2 Classification Algorithm

The classification algorithm is given in figure 2.5. The algorithm input is:
- training sets \( I^+ \) and \( I^- \) (the representation of the VSSVM \( \text{VS}^p_C(I^+, I^-) \)), an
instance \( x \) to be classified, kernel and its parameter \( p \) and the parameter \( C \) of the SVM. The algorithm outputs the classification of \( x \): +1, −1, or 0 in the case of no classification.

The classification algorithm starts by building a hyperplane \( h(p, C, \langle I^+, I^- \rangle) \). If \( h(p, C, \langle I^+, I^- \rangle) \) is inconsistent with \( \langle I^+, I^- \rangle \), then \( \text{VS}_C^p(I^+, I^-) \) is empty according to definition 2.3.2. Thus, according to the unanimous-voting rule the algorithm returns 0. If the hyperplane \( h(p, C, \langle I^+, I^- \rangle) \) is consistent with \( \langle I^+, I^- \rangle \), then \( \text{VS}_C^p(I^+, I^-) \) is non-empty. In this case the algorithm builds SVM hyperplanes \( h(p, C, \langle I^+ \cup \{x\}, I^- \rangle) \) and \( h(p, C, \langle I^+ \cup \{x\}, I^- \rangle) \). If \( h(p, C, \langle I^+ \cup \{x\}, I^- \rangle) \) is inconsistent with \( \langle I^+ \cup \{x\}, I^- \rangle \) and \( h(p, C, \langle I^+ \cup \{x\}, I^- \rangle) \) is consistent with \( \langle I^+ \cup \{x\}, I^- \rangle \), then \( \text{VS}_C^p(I^+ \cup \{x\}, I^-) \) is empty and \( \text{VS}_C^p(I^+ \cup \{x\}, I^-) \) is non-empty. This means that the algorithm assigns class +1 to \( x \) since theorem 1 is assumed to hold. If the class +1 cannot be assigned, the algorithm checks analogously if it can assign class −1. If both classes cannot be assigned, the algorithms returns 0.

### 2.4.3 Example

The classification algorithm is illustrated for the space \( H \) of all oriented lines in \( \mathbb{R}^2 \) and training data: \( I^+ = \{(1, 0), (2, 0), (1, 1), (2, 1)\} \) and \( I^- = \{(-1, 0), (-2, 0), (-1, 1), (-2, 1)\} \) (see figure 2.6). For \( C = +\infty \), only the points to the right of the three line segments through the training points \((1, 0)\) and \((1, 1)\) will be classified as positive and the corresponding region to the left of the three line segments through the training points \((-1, 0)\) and \((-1, 1)\) will be classified as negative. This is true because \( C = +\infty \) means that no classification error may occur. The region between the line segments is the volume of the VSSVM. Running the algorithm with \( C = 30 \) results in the classifications in figure 2.6: positively classified: +, negatively classified: ∗, and not classified: □. It is clear from the figure that for \( C = 30 \) the volume of the VSSVM is smaller than for \( C = +\infty \).

### 2.4.4 Volume-Extension Approach

The volume-extension approach can be applied for VSSVMs via the parameters of a RBF SVM: the constant \( C \) and \( \gamma \) [25]. The underlying assumption is that the parameters are monotonic with the probability that the SVM hyperplane \( h(\gamma, C, \langle I^+, I^- \rangle) \) is consistent with data \( \langle I^+, I^- \rangle \). Below we show that increasing the values of these parameters increases the volume of VSSVMs.

Assume a RBF SVM. The above assumption implies that for two values \( \gamma_1 \) and \( \gamma_2 \) of the parameter \( \gamma \) so that \( \gamma_1 < \gamma_2 \) and arbitrary \( \langle I^+, I^- \rangle \) the prob-
ability that \( h(\gamma_2, C, \langle I^+, I^- \rangle) \) is consistent with \( \langle I^+, I^- \rangle \) is higher than the probability that \( h(\gamma_1, C, \langle I^+, I^- \rangle) \) is consistent with \( \langle I^+, I^- \rangle \). Now if we assume that \( h(\gamma_1, C, \langle I^+, I^- \rangle) \) is consistent with \( \langle I^+, I^- \rangle \) and for each instance \( x \) either of hyperplanes \( h(\gamma_1, C, \langle I^+ \cup \{x\}, I^- \rangle) \) and \( h(\gamma_1, C, \langle I^+, I^- \cup \{x\} \rangle) \) is consistent with their data, then it can be proven that \( V(VS_{\gamma_1}^C(I^+, I^-)) \subseteq V(VS_{\gamma_2}^C(I^+, I^-)) \) \([25]\). Thus, the initial assumption that \( \gamma \) is monotonic with the probability that \( \text{cons}(h(\gamma, C, \langle I^+, I^- \rangle), \langle I^+, I^- \rangle)) \), implies that the volume of VSSVM is monotonic with the parameter \( \gamma \). Analogously, it can be shown that the volume of VSSVM is monotonic with the parameter \( C \).

Applying the volume-extension approach means to find \( C \) and \( \gamma \) in order to (re)define VSSVMs so that instances are classified reliably. Using the assumptions that the volume of VSSVMs is monotonic with \( C \) and \( \gamma \) we can find minimal values for \( C \) and \( \gamma \) using sequential search or binary search so that instances are classified reliably and the volume of VSSVMs is minimized.

Experiments showed that the volume-extension approach applied by changing the parameter \( E \) of a polynomial kernel does not perform as well when \( C \) is used. In this thesis we use \( C \) to apply the volume-extension approach disregarding what type of kernel is used.
2.5 Conclusion of Chapter 2

We showed in this chapter that version spaces and VSSVMs can be considered as approaches to reliable instance classification. Experiments with VSSVMs (presented in [25]) showed that when:

- data are noise free the accuracy is 100% and the coverage varies between 30% - 88%; and

- data are noisy the accuracy is 100% and the coverage varies between 5% - 78% (using the volume extension approach).

For practical applications it is important to maximize the coverage of VSSVMs. This is possible in two ways. The first way is to find values of the parameters of the kernel used and the constant $C$ so that the coverage is maximized. The second way is to use ensemble techniques. In the next chapter 3 we consider the most important ensemble technique called co-training [4, 16]. In chapter 4 we show how to use co-training for VSSVMs to improve the coverage.
Chapter 3

Co-Training

In many practical machine learning problems it is easier to find unlabeled data than labeled data. Co-training [4, 16, 19, 18] is a well-known approach that uses labeled and unlabeled data to boost the accuracy of learned classifiers.

This chapter provides an introduction to co-training. The idea of co-training and the basic algorithm are given in section 3.1. Section 3.2 shows basic assumptions for co-training and section 3.3 gives practical limitations. The chapter ends with a discussion in section 3.4.

3.1 Algorithm

To illustrate the idea of co-training consider the example of learning to classify web pages in two groups: “course homepages” and “other pages” (see [4]). The course homepages are considered to be the positive instances and the other pages are the negative instances. The labeled training data consists of individual web pages along with their correct classification. The crucial point for co-training is that there are two ways to classify a page: (1) using the words that appear on the page or (2) using the words in the hyperlinks that point to that page. This means that there are two ways in which the feature vector $x$ of each instance can be defined. In many cases the words on the hyperlinks will be sufficient to classify the instances and the words on the page itself will also be sufficient. Each feature set is called a view.

The general algorithm of co-training is as follows: given a set of training data, two independent classifiers $A$ and $B$ are trained each on a different view. This means in the running example that classifier $A$ is trained using the words on the web page and $B$ is trained using the hyperlink words. This results in two weak classifiers due to the small number of labeled instances
3.2. BASIC ASSUMPTIONS

Input: a set $L$ of labeled training instances.
    a set of unlabeled instances $U$.
Output: classifiers $A$ and $B$ that classify new instances.

Use view $a$ on the instances of $L$ to train classifier $A$.
Use view $b$ on the instances of $L$ to train classifier $B$.
Loop until $U$ is empty or until some threshold is reached:
    Allow $A$ to classify for each class $c$ those instances in $U$ for which
    $A$ is most confident that its class label is $c$.
    Allow $B$ to classify for each class $c$ those instances in $U$ for which
    $B$ is most confident that its class label is $c$.
    Add the new-labeled instances to $L$.
Use view $a$ on the instances of $L$ to train classifier $A$.
Use view $b$ on the instances of $L$ to train classifier $B$.

Figure 3.1: Outline of the co-training algorithm.

that are available. Nevertheless the initial classifiers are better than random.
After the training each classifier is allowed to check the pool of unlabeled
data to find the instances for which the classifier is the most certain that
its label prediction is correct. These instances are then added to the pool
of labeled training data and both classifiers are retrained. This process is
repeated until some threshold is reached or until the pool with unlabeled
data has become empty. Figure 3.1 outlines this process.

Analysis of co-training shows that we observe a boost in accuracy of
classifiers $A$ and $B$ when: (1) $A$ and $B$ classify instances correctly and (2)
there exists instances which $A$ classifies with more confidence than $B$ and
vice versa. The latter means that both classifiers can present each other
with useful labeled instances.

3.2 Basic Assumptions

The co-training framework [4] is proposed for the classification task. The
instance space is defined as $X = X_1 \times X_2$ where $X_1$ and $X_2$ are two subspaces
that represent two distinct views on instances. Each instance $x$ is therefore
represented by the pair $(x_1, x_2)$ where $x_1 \in X_1$ and $x_2 \in X_2$.

The first assumption is that each view on its own is sufficient to classify
the instances. This means that $\exists h_1 : X_1 \rightarrow Y$ and $\exists h_2 : X_2 \rightarrow Y$ so that
$\forall x \in X : h_t(x) = h_1(x_1) = h_2(x_2)$ where $h_t$ is the target function. If this
assumption is valid it is said that $X_1$ and $X_2$ are redundantly sufficient. The redundantly sufficient views assumption is very important for co-training and is sometimes called the compatibility assumption \cite{4}.

There is a second assumption on the views which is called class-conditional independency or the uncorrelated views assumption. The assumption is that the distribution over $X$ generates instances for which one classifier is more confident about the label than the other classifier. This ensures that the classifiers can provide each other with useful labeled data. A PAC based proof for the effectiveness of co-training is possible \cite{4}. Interested readers are referred to \cite{17} in which both assumptions on the views are explained in detail and applied to the example of classifying web pages.

3.3 Limitations

3.3.1 Labeled Data

This subsection considers classification tasks where a large number of training instances are required to achieve usable performance levels. Co-training does not scale well in this setting due to mistakes made by the view classifiers. This is especially true at the start of the algorithm when the classifiers are trained on a small set of labeled data. Occasionally the view classifiers may add incorrectly labeled instances to the pool of labeled data. If many iterations of the co-training algorithm are required, the effectiveness of co-training is dulled over time by the degradation of the quality of the labeled data.

The above problem was observed in \cite{21}. To partially overcome this problem a modified co-training algorithm was proposed. This algorithm is given in figure 3.2. The first difference with the original algorithm is that a pool $U'$ is introduced which contains a randomly selected subset of the instances in the complete set $U$. The reason for this is that the computation time becomes very high when $|U|$ is large. The second difference is that the distribution of labeled instances is according to a label distribution $D_L$. This distribution specifies the probability that a randomly chosen instance has class label $c$ and is used to preserve the ratio of labels in $L$.

It is obvious that the amount of initially labeled data is important. When $|L|$ is too large the initial classifiers are too accurate in the sense that co-training only improves accuracy very slowly. When $|L|$ is too small only a part of the learning task is learned because the unlabeled instances that the classifiers label are those that are the most similar to instances which were seen in the small set $L$ at the start of the algorithm. Proceeding in this direction, the classifiers will not find those instances that are helpful.
Input: a set $L$ of labeled training instances.  
a set of unlabeled instances $U$.  
Output: classifiers $A$ and $B$ that classify new instances.

Use view $a$ on the instances of $L$ to train classifier $A$.  
Use view $b$ on the instances of $L$ to train classifier $B$.  
Transfer randomly $u$ instances from $U$ to $U'$.  
For each classifier $T \in \{A, B\}$ repeat $g$ times:  
Select label $c$ at random according to $D_L$.  
Allow $T$ to classify the instance in $U'$ for which $T$ is most  
confident that its class label is $c$.  
Add the new-labeled instance to $L$.  
Use view $a$ on the instances of $L$ to train classifier $A$.  
Use view $b$ on the instances of $L$ to train classifier $B$.

Figure 3.2: Outline of the modified co-training algorithm.

for learning new aspects of the task because most of the time unfamiliar  
instances are needed for this.

3.3.2 Selection

Each classifier is allowed to classify unlabeled instances and place them in the  
pool $U$. Until now it is only considered to choose those instances for which  
the classifier is the most confident that the predicted label is correct. This  
is an attempt to minimize noise and it is not guaranteed that the classifiers  
provide each other with useful labeled instances, i.e., training utility is not  
maximized.

In [27] experiments are performed with different selection methods. The  
results indicate that selection methods that maximize training utility find  
instances that result in better classifiers than those that only minimize error.  
Instances that maximize training utility have to satisfy some criteria, e.g.,  
the confidence of one classifier is greater than that of the other classifier by  
some threshold $n$. In real-life situations it is only a question if the unlabeled  
data pool $U$ contains sufficient number of these instances and it is likely  
that noise is introduced. It is more interesting if both classifiers or some  
oracle could generate instances for which it is known that training utility is  
maximized and training error minimized.
3.4 Discussion

Co-training can be used to boost the accuracy of a learning algorithm by using unlabeled data. Two assumptions provide a best case scenario for co-training. The first assumption is that the views are redundantly sufficient and the second assumption is that views have to be uncorrelated.

In the next chapter 4 we propose to co-train VSSVMs to solve the coverage reduction problem. We now motivate why co-training of VSSVMs is potentially applicable. The first assumption of co-training holds for VSSVMs when the training data are noise-free and each view is expressive, i.e., the target function is in each view. If the training data are noisy and/or if the hypothesis space is not expressive, we still can use the volume-extension approach. Thus, we still classify with the target function on a large part of the data distribution. The second assumption holds when VSSVMs are complementary. This means that unclassified instances of a VSSVM can be classified by other VSSVMs. This assumption is easily reachable. We have to use for co-training of VSSVMs informative instances, i.e., instances that are classified correctly by approximately half of the hypotheses in the VSSVM.

The above discussion shows that the limitations of the original co-training algorithm can be overcome for co-training of VSSVMs. Therefore co-training of VSSVMs can be used to boost the coverage if no additional problems with the VSSVMs are encountered. The next chapter 4 describes and explains the above analysis in detail.
Chapter 4

Co-Training of Version Space Support Vector Machines

This chapter addresses the thesis problem statement. The goal is to solve the coverage reduction problem of VSSVMs.

To solve the problem the approach is to co-train VSSVMs represented in different hypothesis spaces. The algorithm starts by training VSSVMs on the training data. Each VSSVM is allowed to generate informative instances that improve their coverage when added as training instances. After generation, the informative instances are classified by other VSSVMs. The above steps are performed multiple times.

This chapter is organized as follows. Section 4.1 explains the choice of informative instances and how they are constructed. Then four models for co-training of VSSVMs are introduced. Section 4.2 provides descriptions, experiments, and conclusions for each model.

4.1 Informative Instances

This section formalizes the notion of informative instances and shows how they can be constructed. Subsection 4.1.1 defines informative instances for a VSSVM as instances on the SVM optimal hyperplane that represents the VSSVM. Subsection 4.1.1 motivates this definition. The problem of finding informative instances is formulated as an optimization problem that is solved with the steepest descent algorithm. Details are given in subsections 4.1.2 and 4.1.3. The start instance for steepest descent is important for the convergence. This problem is discussed in subsection 4.1.4.
4.1. INFORMATIVE INSTANCES

4.1.1 Definition

In subsection 3.3.2 we explained that co-training benefits the most from instances that maximize training utility. For co-training of VSSVMs training utility means that new-labeled instances result in a decrease of the volume of VSSVM. Consider for example figure 4.1. The blue vertical line is the SVM optimal hyperplane of a VSSVM, \( VSSVM_1 \). The volume of \( VSSVM_1 \) is the grey area. Note that the boundary lies in the middle of the volume if the margin-width is considered as measurement. Negative instances are represented as circles and positive instances as plus signs. Suppose that some instances are generated at random. One of these instances is indicated by a red cross in the figure. \( VSSVM_1 \) sends this instance for classification to another VSSVM, \( VSSVM_2 \). If the latter classifies the instance as negative then the volume of \( VSSVM_1 \) shrinks much more than when the classification is positive. This shows that it is interesting to generate instances that maximizes the reduction of the volume in average. We propose to define informative instances for VSSVMs as those instances that lie on the SVM optimal hyperplane representing these VSSVMs. In this way these instances will cause approximately half of the hypotheses in the VSSVM to be removed when added to the training data.

Figure 4.1: The volume of a VSSVM, SVM boundary and possible informative instance.

\[ \text{In the rest of this chapter the term boundary is used for the SVM optimal hyperplane} \]
4.1. INFORMATIVE INSTANCES

4.1.2 Optimization Formulation

The search for informative instances is formulated as an optimization problem. This means that some performance index \( F(x) \) is defined for which the minima must be found. These minima are instances on the SVM boundary. The performance index that is chosen defines for each instance the distance to this boundary. Therefore, minimum function value that can be reached is zero. The minimization function is derived from the general classification formula of SVMs (formula 2.1) and is defined as:

\[
\left( \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right)^2
\]  

(4.1)

The square ensures that the performance index is continuous. The distance of instance \( x \) to the boundary is measured in kernel space and \( x \) is maintained in instance space. This makes it possible to add the found minimum to the pool of labeled data.

4.1.3 Optimization Algorithm

The optimization algorithm that is used is the steepest descent algorithm [7]. This is an iterative optimization algorithm that works as follows. It starts with some initial guess \( x_0 \) and then updates the guess in stages according to \( x_{k+1} = x_k + \beta_k p_k \). The vector \( p_k \) represents the search direction and the positive scalar \( \beta_k \) is the learning rate which determines the length of the step. The best search direction for an instance is the negative of the gradient in that instance because the gradient gives the direction in which the function increases most rapidly. The learning rate is calculated by line search, i.e., \( F(x) \) is minimized with respect to \( \beta_k \) at each iteration. This \( \beta_k \) is found by an efficient approximate line minimization algorithm. This algorithm consists of two phases: a bracketing phase and a sectioning phase. Both involve the minimization of a cubic interpolation polynomial over a bounded interval. A detailed explanation of the complete line search routine can be found in [7].

Independent of the kernel that is used the gradient of the performance index \( F(x) \) equals to:

\[
\nabla_x \left[ \left( \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right)^2 \right]
\]  

(4.2)

This expression is equal to:

\[
2 \left( \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right) \nabla_x \left[ \sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right]
\]  

(4.3)
and if the last term the last term is expanded, we have:

\[ 2 \left( \sum_{i=1}^{l} y_{i} \alpha_{i} K(x_{i}, x) + b \right) \left( \sum_{i=1}^{l} y_{i} \alpha_{i} \nabla_{x} \left[ K(x_{i}, x) \right] \right) \quad (4.4) \]

The term \( \nabla_{x} \left[ K(x_{i}, x) \right] \) is different for each kernel. For a RBF function \( K(x_{i}, x) = e^{-\gamma ||x_{i} - x||^2} \) the following equation holds:

\[
\nabla_{x} \left[ K(x_{i}, x) \right] = \left[ \frac{\partial}{\partial x_{j}} K(x_{i}, x) \right]_{j=1,...,d} = [K(x_{i}, x)2\gamma(x_{ij} - x_{j})]_{j=1,...,d} \quad (4.5)
\]

For a polynomial kernel \( K(x_{i}, x) = (x_{i} \cdot x)^p \) the solution is:

\[
\nabla_{x} \left[ K(x_{i}, x) \right] = \left[ \frac{\partial}{\partial x_{j}} K(x_{i}, x) \right]_{j=1,...,d} = \left[ p(x_{i} \cdot x)^{p-1} \frac{\partial}{\partial x_{j}}(x_{i} \cdot x) \right]_{j=1,...,d} \quad (4.7)
\]

\[
= \left[ p(x_{i} \cdot x)^{p-1} x_{ij} \right]_{j=1,...,d} \quad (4.8)
\]

For a lower-order terms polynomial kernel \( K(x_{i}, x) = ((x_{i} \cdot x) + 1)^p \) we have:

\[
\nabla_{x} \left[ K(x_{i}, x) \right] = \left[ \frac{\partial}{\partial x_{j}} K(x_{i}, x) \right]_{j=1,...,d} = \left[ p((x_{i} \cdot x) + 1)^{p-1} \frac{\partial}{\partial x_{j}}((x_{i} \cdot x) + 1) \right]_{j=1,...,d} \quad (4.10)
\]

\[
= \left[ p((x_{i} \cdot x) + 1)^{p-1} x_{ij} \right]_{j=1,...,d} \quad (4.11)
\]

4.1.4 Initial Guess

The initial instance of the steepest descent algorithm is important because the performance index \( F(x) \) can have a lot of local optima in which steepest descent can get stuck. Our experiments show that steepest descent performs very well in kernel space.

The importance of the initial guess becomes clearer when the performance index is considered for SVMs with a Gaussian RBF kernel. In this setting, the performance index can be zero in two cases:

- the instance is on the boundary; or
- the instance is not on the boundary and is far away from each support vector.
4.2. CO-TRAINING MODELS

To see why this is true the formula of the Gaussian RBF has to be considered:

\[ K(x, y) = e^{-\gamma||x-y||^2} \]  \hspace{1cm} (4.13)

It is easy to see that if \( x \) and \( y \) are far away from each other the kernel value is very small. Now if the minimization function (formula 4.2) is again considered and instance \( x \) is far away from each support vector \( x_i \), then each \( K(x, x_i) \) is very small. This yields a value of the minimization function close to the minimum and the optimization algorithm then incorrectly concludes that \( x \) is on the boundary.

For the success of the VSSVM co-training the steepest descent algorithm may not take steps in each iteration that lead away from each support vector. This can be done by placing the initial guesses so that the steepest search direction leads to the boundary. If a non-empty VSSVM is considered the boundary goes through the space that separates each positive support vector from each negative support vector. Therefore, a good choice for the initial guesses are the midpoints from the lines that connect a positive and a negative support vector.

4.2 Co-Training Models

This section gives a detailed description of different co-training models that are implemented and tested. In total four models are described.

4.2.1 First Model

Description

The first co-training model starts with training two VSSVMs, \( VSSVM_1 \) and \( VSSVM_2 \), in different hypothesis spaces. Each VSSVM has its own pool of labeled data \( (L_1 \) and \( L_2) \) which is initialized equal to the training set.

No co-training is performed when at least one of the VSSVMs is empty. If both are non-empty \( VSSVM_1 \) generates informative instances. Instances that are classified by \( VSSVM_2 \) form the set \( I_1 \). Then, \( VSSVM_2 \) generates informative instances as well. Instances that are classified by \( VSSVM_1 \) form the set \( I_2 \). The class distribution \( D_L \) is used to preserve the ratio of positive and negative instances.

Informative instances generated for a VSSVM are only used by that VSSVM, i.e., the set \( I_1 \) (\( I_2 \)) is added to \( L_1 \) (\( L_2 \)). When a VSSVM can classify an informative instance the label is compared to the label that the
4.2. CO-TRAINING MODELS

**Input:** a set $L$ of labeled training instances.

**Output:** $VSSVM_1$ and $VSSVM_2$ that classify new instances.

$L_1 = L, L_2 = L$.

Use $L_1$ to train $VSSVM_1$ and $L_2$ to train $VSSVM_2$.

Loop for $n$ iterations or until some threshold is reached:

- Generate informative instances $I_1$ for $VSSVM_1$.
- Let $VSSVM_2$ classify instances in $I_1$ according to $D_L$.
- Add the largest subset of these new-labeled instances to $L_1$ so that there is no disagreement with the predictions of $VSSVM_1$ and there is no collapse of $VSSVM_1$.
- Generate informative instances $I_2$ for $VSSVM_2$.
- Let $VSSVM_1$ classify instances in $I_2$ according to $D_L$.
- Add the largest subset of these new-labeled instances to $L_2$ so that there is no disagreement with the prediction of $VSSVM_2$ and there is no collapse of $VSSVM_2$.
- Use $L_1$ to train $VSSVM_1$ and $L_2$ to train $VSSVM_2$.

Figure 4.2: Outline of the first co-training model.

other VSSVM assigns. The informative instance is not added to the labeled data pool if $VSSVM_1$ and $VSSVM_2$ classify the instance differently.

It is possible that a VSSVM collapses when informative instances are added to the labeled data pool. The problem is solved by searching for the largest subset of the informative instances that can be added to the labeled data so that there is no collapse. However, it is not guaranteed that this approach gives the largest increase in coverage.

The first co-training model as described above is given in figure 4.2. The classification scheme is illustrated in figure 4.3. It is a variant of unanimous-voting. The scheme uses $VSSVM_1$ and $VSSVM_2$ to predict the class label of an instance. If the predictions coincide the label is assigned. If the predicted labels are different the scheme checks if exactly one VSSVM leaves the instance unclassified. The output is then the prediction of the VSSVM that classifies the instance. When $VSSVM_1$ and $VSSVM_2$ predict different labels the instance is left unclassified because it is not known which label is correct.
4.2. CO-TRAINING MODELS

Input: an instance $x$.
Output: classification for $x$.

$c_1 =$ label predicted by $VSSVM_1$.
$c_2 =$ label predicted by $VSSVM_2$.

If $c_1 = c_2$ then assign this label.
Else:
  If $c_1 = 0$ and $c_2 \neq 0$ then $c_2$ is the label.
  Else if $c_1 \neq 0$ and $c_2 = 0$ then $c_1$ is the label.
  Else the label is 0.

Figure 4.3: Co-training classification scheme for the first model.

Experiment

The VSSVM is implemented in WEKA [28] using the SMO implementation of SVM [22]. The co-training methods are implemented in WEKA for two VSSVMs.

A noise-free artificial training set consisting of 25 instances is used. The feature space is $\mathbb{R}^2$. Figure 4.4 shows the dataset. Instances represented by a plus sign are the positive training instances. The circles are the negative training instances. Most figures in this chapter contain SVM boundaries. These are drawn by the steepest descent algorithm as explained in section 4.1.3. The precision of the boundary depends on the number and location of the support vectors. Instances of interest (test instances or informative instances) are always marked as a red circle or cross.

During the experiments we allowed at most five informative instances for each VSSVM to be added in a co-training iteration. This is 20% of the number of instances in the training set. The label distribution states that no more than three negative and two positive informative instances may be added to the corresponding pool.

The SVMs that are used in the VSSVMs have both lower-order polynomial kernels. The parameters for the first SVM ($SVM_1$) are $C = 1.0$ and $E = 2.7$. The second SVM ($SVM_2$) is defined by $C = 1.0$ and $E = 3.0$. $SVM_1$ ($SVM_2$) is used for testing collapse of $VSSVM_1$ ($VSSVM_2$).

The method for evaluation is the leave-one-out (LOO) method. The statistics to compare the results are the number of correctly, incorrectly and unclassified instances.

A limitation is that the coverage can increase without the LOO method
4.2. CO-TRAINING MODELS

![Image](image.png)

Figure 4.4: Artificial dataset to compare the co-training algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>18</td>
<td>72%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>6</td>
<td>24%</td>
</tr>
</tbody>
</table>

Table 4.1: Results of the first model without co-training.

to notice. Figure 4.5 gives an example. The red cross at the right of the SVM optimal hyperplane of $VSSVM_1$ is the test instance. The black dot on the boundary is an informative instance and the grey area is the volume of $VSSVM_1$. Suppose that $VSSVM_2$ classifies the informative instance as negative. The volume of $VSSVM_1$ then shrinks a lot but the test instance is still left unclassified.

The results from the first model with no co-training and after one co-training iteration are summarized in tables 4.1 and 4.2. They show that the coverage and accuracy after co-training are decreasing. This is due to

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>17</td>
<td>68%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>7</td>
<td>28%</td>
</tr>
</tbody>
</table>

Table 4.2: Results of the first model after one co-training iteration.
the instance \( x_{\text{loo}} \) from the data with coordinates (0.3, 0.15) that becomes unclassified from correctly classified. Below the causes for this negative effect are analyzed. Since the evaluation is realized with LOO the classification of \( x_{\text{loo}} \) is analyzed for its own LOO set. This set consists of all the training instances minus \( x_{\text{loo}} \).

Before co-training \( \text{VSSVM}_1 \) classifies \( x_{\text{loo}} \) as positive and \( \text{VSSVM}_2 \) is not able to classify \( x_{\text{loo}} \). Therefore, the output of the classification scheme is positive which is the correct label.

Co-training is performed because both VSSVMs are non-empty on the LOO dataset. We will now analyze what happens in steps according to the model in figure 4.2.

\( \text{VSSVM}_1 \) uses steepest descent to generate informative instances which \( \text{VSSVM}_2 \) tries to classify. The set \( I_1 \) of labeled informative instances is sent back to \( \text{VSSVM}_1 \). \( \text{VSSVM}_1 \) searches for the largest subset of \( I_1 \) that can be added to \( L_1 \) so that the new \( \text{VSSVM}_1 \) is still non-empty. The only instance \( x_{\text{inf}} \) that can be added to \( L_1 \) is (0.77, 0.84). \( \text{VSSVM}_2 \) classified this instance as negative because:

- \( \text{VSSVM}_2 \) is non-empty when \( x_{\text{inf}} \) is added as negative (figure 4.6); and
- \( \text{VSSVM}_2 \) is empty when \( x_{\text{inf}} \) is added as positive (figure 4.7).

Figure 4.8 shows that \( \text{VSSVM}_1 \) is still non-empty when trained on \( L_1 \) and \( x_{\text{inf}} \) with negative label.
4.2. CO-TRAINING MODELS

Figure 4.6: Boundary of $SVM_2$ representing $VSSVM_2$ when $x_{inf}$ is added as negative.

Figure 7: Boundary of $SVM_2$ representing $VSSVM_2$ when $x_{inf}$ is added as positive.

Figure 4.8: Boundary of $SVM_1$ representing $VSSVM_1$ when $x_{inf}$ is added as negative.
The second step in the co-training model is to repeat the previous steps for $VSSVM_2$. This means that $VSSVM_2$ generates informative instances and $VSSVM_1$ tries to classify them. The only labeled informative instance that can be added to $L_2$ whilst preserving a non-empty $VSSVM_2$ is $(0.52, 0.55)$.

The final step is to classify $x_{l_oo}$ with the new $VSSVM_1$ and $VSSVM_2$. $VSSVM_1$ leaves the instance unclassified because:

- $VSSVM_1$ is empty when $x_{l_oo}$ is added as negative (figure 4.9); and
- $VSSVM_1$ is empty when $x_{l_oo}$ is added as positive (figure 4.10).

Thus, $VSSVM_1$ has lost its power to classify $x_{l_oo}$ due to only one informative instance. $VSSVM_2$ left the instance unclassified before and after co-training (figures 4.11 and 4.12). This means that the instance is left unclassified by the co-training classification scheme.

**Discussion of the First Model**

The first model shows a misclassified instance before co-training. The incorrect label is caused by one of the following cases as discussed in section 2.3.3:

- noisy training data; or
- inexpressive hypothesis space.
4.2. CO-TRAINING MODELS

Figure 4.11: Boundary of $SVM_2$ representing $VSSVM_2$ when $x_{loo}$ is added as negative.

Figure 4.12: Boundary of $SVM_2$ representing $VSSVM_2$ when $x_{loo}$ is added as positive.

Table 4.3: Possible combinations of $VS^p_C(I^+ \cup \{x\}, I^-)$ and $VS^p_C(I^+, I^- \cup \{x\})$ in terms of collapse.

<table>
<thead>
<tr>
<th>Combination</th>
<th>$VS^p_C(I^+ \cup {x}, I^-)$</th>
<th>$VS^p_C(I^+, I^- \cup {x})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$= \emptyset$</td>
<td>$= \emptyset$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$= \emptyset$</td>
<td>$\neq \emptyset$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$\neq \emptyset$</td>
<td>$= \emptyset$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$\neq \emptyset$</td>
<td>$\neq \emptyset$</td>
</tr>
</tbody>
</table>

To explain the behavior of the first model we analyze the model on two levels. The first level is the level of VSSVMs. The second level is the level of the unanimous-voting rule employed as the classification rule for the co-training.

The level of VSSVMs. To analyze the first model on the level of VSSVMs we analyze the VSSVM classification algorithm (see figure 2.5) before and after the co-training phase.

Before the co-training phase we consider the situation when we have training data $<I^+, I^->$ and a non-empty VSSVM $VS^p_C(I^+, I^-)$. To classify an instance $x$, the VSSVM classification algorithm checks VSSVMs $VS^p_C(I^+ \cup \{x\}, I^-)$ and $VS^p_C(I^+, I^- \cup \{x\})$ for collapse. The four possible combinations with respect to collapse of these VSSVMs are presented in table 4.3.

\footnote{For the sake of clarity we use here the notation of VSSVMs introduced in chapter 2}
4.2. CO-TRAINING MODELS

<table>
<thead>
<tr>
<th>Combination</th>
<th>$VS_C^p(I^+ \cup {x}, I^-)$</th>
<th>$VS_C^p(I^+, I^- \cup {x})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1'$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$c_2'$</td>
<td>$\emptyset$</td>
<td>$\neq \emptyset$</td>
</tr>
<tr>
<td>$c_3'$</td>
<td>$\neq \emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$c_4'$</td>
<td>$\neq \emptyset$</td>
<td>$\neq \emptyset$</td>
</tr>
</tbody>
</table>

Table 4.4: Possible combinations of $VS_C^p(I^+ \cup \{x\}, I^-)$ and $VS_C^p(I^+, I^- \cup \{x\})$ in terms of collapse.

All the four combinations from table 4.3 are possible for VSSVMs. The combinations $c_1$ and $c_4$ imply that the instance $x$ is not classified. The combinations $c_2$ and $c_3$ imply that the instance $x$ is classified. We note that the combination $c_1$ means that the version-space algebra is not preserved for VSSVMs. Consider the equation below that represents one of the dependencies from the version-space algebra:

$$VS_C^p(I^+, I^-) = VS_C^p(I^+ \cup \{x\}, I^-) \cup VS_C^p(I^+, I^- \cup \{x\})$$ (4.14)

When the combination $c_1$ holds, the left-hand side of equation 4.14 is non-empty and the right-hand side is empty, i.e., equation 4.14 does not hold. So, the version-space algebra is not preserved.

After the co-training phase the training data $\langle I^+, I^- \rangle$ are updated by adding new instances. The updated data are denoted as $\langle I^+, I^- \rangle$. When we classify an instance $x$ with a non-empty VSSVM $VS_C^p(I^+, I^-)$, we have the same four combinations, see table 4.4. We note that again all the combinations are possible for VSSVMs. The equation:

$$VS_C^p(I^+, I^-) = VS_C^p(I^+ \cup \{x\}, I^-) \cup VS_C^p(I^+, I^- \cup \{x\})$$ (4.15)

does not always hold due the combination $c_1'$.

Combining combinations $c_i$ with combinations $c_j'$ for $i, j \in \{1, 2, 3, 4\}$ allows us to specify all possible sixteen cases of transformation of instance classifications due to co-training. For the sake of simplicity of the presentation we consider only situations when an instance to be classified is positive or it is unclassified. Situations when the instance is classified as negative are symmetric.

Case 1: Correctly Classified Instance becomes Unclassified
If an instance $x$ is classified correctly as positive before co-training the combination $c_3$ from table 4.3 holds. If an instance $x$ is unclassified after co-training
one of the combinations $c'_1$ and $c'_4$ from table 4.4 holds. Thus, since these combinations are valid for VSSVMs, we conclude that a correctly classified positive instance becomes unclassified due to co-training when we have one of the following transformations:

$$c_3 ightarrow c'_1 \quad (4.16)$$
$$c_3 ightarrow c'_4 \quad (4.17)$$

We note that for both transformations the version-space algebra is not preserved. For the transformation $c_3 ightarrow c'_1$ this is due to combination $c'_1$. For the transformation $c_3 ightarrow c'_4$ this is due to the fact that we co-train (i.e., add new training instances) and empty $VS_C^p(I^+ \cup \{x\})$ becomes non-empty $VS_C^p(I'^+ \cup \{x\})$. This transition would not be possible when SVMs were perfect consistency algorithms because an empty version space stays empty when more training data are added.

Let us consider what will happen if we apply the volume-extension approach by increasing or decreasing $C$ before and after co-training. If we apply the volume-extension approach by increasing the parameter $C$ then more version spaces become non-empty. This means that more transitions $VS_C^p(I^+ \cup \{x\}, I^-) \rightarrow VS_C^p(I'^+ \cup \{x\}, I'^-)$ and $VS_C^p(I^+ \cup \{x\}) \rightarrow VS_C^p(I'^+ \cup \{x\})$ that before corresponded to transformation $c_3 ightarrow c'_1$ will correspond to transformations $c_3 ightarrow c'_2$, $c_3 ightarrow c'_3$, $c_3 ightarrow c'_4$, $c_4 ightarrow c'_1$, $c_4 ightarrow c'_2$, $c_4 ightarrow c'_3$, and $c_4 ightarrow c'_4$.

If we apply the volume-extension approach by decreasing the parameter $C$ then more version spaces become empty. This means that more transitions $VS_C^p(I^+ \cup \{x\}, I^-) \rightarrow VS_C^p(I'^+ \cup \{x\}, I'^-) and VS_C^p(I^+ \cup \{x\}) \rightarrow VS_C^p(I'^+ \cup \{x\})$ that before corresponded to transformation $c_3 ightarrow c'_4$ will correspond to transformations $c_1 ightarrow c'_1$, $c_1 ightarrow c'_2$, $c_1 ightarrow c'_3$, $c_1 ightarrow c'_4$, $c_3 ightarrow c'_1$, $c_3 ightarrow c'_2$, and $c_3 ightarrow c'_3$.

Since the transformation $c_3 ightarrow c'_3$ is the desired one, we have to look for the parameter $C$ so that it holds more often.

**Case 2: Incorrectly Classified Instance becomes Unclassified**

Case 2 can be explained analogously to case 1. Since case 2 is desirable, we do not have to apply the volume-extension approach.

**Case 3: Incorrectly Classified Instance becomes Correctly Classified**

If an instance $x$ is classified incorrectly as negative before co-training the combination $c_2$ from table 4.3 holds. If an instance $x$ is classified correctly as positive after co-training combination $c'_3$ from table 4.4 holds. Thus, in this
case we have the transformation:

\[ c_2 \rightarrow c'_3 \quad (4.18) \]

We note that for this transformation the version-space algebra is not preserved. This is due to the fact that we co-train (i.e., add new training instances) and empty \( V_S^p(C, I^+ \cup \{x\}, I^-) \) becomes non-empty \( V_S^p(C, I'^+ \cup \{x\}, I'^-). \)

Since case 3 is desirable, we do not have to apply the volume-extension approach.

**Case 4: Correctly Classified Instance becomes Incorrectly Classified**

Case 4 can be explained analogously to case 3. For case 4 we have the transformation:

\[ c_3 \rightarrow c'_2 \quad (4.19) \]

To avoid case 4, we can apply the volume-extension approach. If we apply this approach before and after co-training by increasing the parameter \( C \) then more version spaces become non-empty. This means that more transitions \( V_S^p(C, I^+ \cup \{x\}, I^-) \rightarrow V_S^p(C, I'^+ \cup \{x\}, I'^-) \) and \( V_S^p(C, I^+ \cup \{x\}, I^-) \rightarrow V_S^p(C, I'^+ \cup \{x\}, I'^-) \) that before corresponded to transformation \( c_3 \rightarrow c_2 \) will correspond to transformations \( c_3 \rightarrow c'_4 \), \( c_4 \rightarrow c'_2 \) and \( c_4 \rightarrow c'_4 \). If we apply the volume-extension approach by decreasing the parameter \( C \), we have the opposite effect: more transitions will correspond to transformations \( c_3 \rightarrow c'_1 \), \( c_1 \rightarrow c'_2 \) and \( c_1 \rightarrow c'_4 \). In both situations we go more to the direction of unclassified instances.

**Case 5: Unclassified Instance becomes Correctly Classified**

If an instance \( x \) is unclassified before co-training one of the combinations \( c_1 \) and \( c_4 \) from Table 4.3 holds. If an instance \( x \) is classified correctly as positive after co-training the combination \( c'_3 \) from Table 4.4 holds. Thus, since these combinations are valid for VSSVMs, we conclude that an unclassified instance becomes correctly classified as positive due to co-training when we have one of the following transformations:

\[ c_1 \rightarrow c'_3 \quad (4.20) \]
\[ c_4 \rightarrow c'_3 \quad (4.21) \]

We note that for transformation 4.20 the version-space algebra is not preserved. This is due to the fact that we co-train (i.e., add new training
4.2. CO-TRAINING MODELS

instances) and empty $\text{VS}_C^p(I^+ \cup \{x\}, I^-)$ becomes non-empty $\text{VS}_C^p(I'^+ \cup \{x\}, I'^-)$.

Since case 5 is desirable, we do not have to apply the volume-extension approach.

**Case 6: Unclassified Instance becomes Incorrectly Classified**

Case 6 can be explained analogously to case 5. For case 6 one of the following transformations occurs:

$$c_1 \rightarrow c'_2$$  \hspace{1cm} (4.22)

$$c_4 \rightarrow c'_2$$  \hspace{1cm} (4.23)

To avoid case 6, we can apply the volume-extension approach. If we apply the volume-extension approach before and after co-training by increasing the parameter $C$ then more version spaces become non-empty. This means that more transitions $\text{VS}_C^p(I^+ \cup \{x\}, I^-) \rightarrow \text{VS}_C^p(I'^+ \cup \{x\}, I'^-)$ and $\text{VS}_C^p(I'^+, I'^- \cup \{x\}) \rightarrow \text{VS}_C^p(I^+, I^- \cup \{x\})$ will correspond to transformations $c_1 \rightarrow c'_4$, $c_2 \rightarrow c'_2$, $c_2 \rightarrow c'_4$, $c_3 \rightarrow c'_2$, $c_3 \rightarrow c'_4$, and $c_4 \rightarrow c'_4$. If we apply the volume-extension approach by decreasing the parameter $C$, we have the opposite effect: more transitions will correspond to transformations $c_4 \rightarrow c'_1$, $c_3 \rightarrow c'_3$, $c_3 \rightarrow c'_2$, $c_2 \rightarrow c'_1$, $c_2 \rightarrow c'_2$, $c_1 \rightarrow c'_1$. In both situations we have introduced the possibility that the unclassified instance stays unclassified with 50% probability (under the assumption that each transformation has the same probability).

These six cases can be divided into two groups according to the probability they occur. In cases 1, 2, 5 and 6 only one VSSVM on the right-hand side of equation 4.14 has to become empty from non-empty or vice versa. In cases 3 and 4 both VSSVMs have to change. The distribution for positive and negative effects is 50% in each group.

The application of the volume-extension approach is not a trivial task. The problem is that while trying to avoid negative cases (e.g., case 1) we avoid positive cases (e.g., case 2). Thus, we first have to determine with an internal cross-validation process a frequency for each type of transformation we observe during co-training. Using the frequencies we can decide how to apply the volume-extension approach: by increasing the parameter $C$ or by decreasing the parameter $C$. The volume-extension approach can be applied by maximizing the accuracy and coverage of the VSSVMs using an internal cross-validation process.

**The level of unanimous-voting.** Co-trained VSSVMs assign a label to an instance with unanimous-voting of the VSSVMs involved. Table
4.2. CO-TRAINING MODELS

<table>
<thead>
<tr>
<th>VSSVM$_1$</th>
<th>VSSVM$_2$</th>
<th>Unanimous-Voting</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>i</td>
<td>u</td>
</tr>
<tr>
<td>c</td>
<td>u</td>
<td>c</td>
</tr>
<tr>
<td>i</td>
<td>c</td>
<td>u</td>
</tr>
<tr>
<td>i</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>i</td>
<td>u</td>
<td>i</td>
</tr>
<tr>
<td>u</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>u</td>
<td>i</td>
<td>i</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

Table 4.5: Analysis of unanimous-voting of two VSSVMs: all possible classifications of VSSVM$_1$, VSSVM$_2$, and unanimous-voting. c stands for correctly classified, i stands for incorrectly classified and u stands for unclassified.

4.5 shows all possible combinations of the classifications combined with the unanimous-voting. Since VSSVMs are better than random, the rows with at least one i (incorrectly classified) have less probability than the other rows. The unanimous-voting does not classify an instance for the rows containing a correct (c) and incorrect classification (i). For the other rows with an incorrect classification, the output is the incorrect label but the probability of these rows are small. Therefore, the unanimous-voting is a stabilization factor.

Conclusion of the First Model

Our experiments and analysis of the first model showed that the version-space algebra is not preserved for VSSVMs. This means that by increasing the size of training data the classification boundaries of VSSVMs do not converge. Instead they diverge on some parts of the instance space. This effect actually causes the positive and negative cases of instance classification transformation described in subsection 4.2.1. To reduce the number of the negative cases as well as the noisy sensitivity of VSSVMs we proposed to use the volume-extension approach. The VSSVMs generated with the approach can have better generalization performance, so the coverage of VSSVMs can be improved. In addition, we showed that the unanimous-voting rule is a stabilization factor for co-trained schemes with VSSVMs.
4.2. CO-TRAINING MODELS

4.2.2 Second Model

Description

The second co-training model is similar to the first model. The only difference is that a binary search is performed for the optimal parameters $p$ and $C$ for the VSSVMs when:

- the VSSVMs are initialized with the training data; and
- the VSSVMs are retrained on the training data plus the labeled informative instances.

Using optimal parameters we try to find more plausible boundaries with enough generalization power so that previously classified instances can still be classified.

The classification scheme used is that of the first model.

Experiment

In the experiments we use VSSVMs represented by SVMs with a polynomial kernel ($SVM_1$) and a RBF kernel ($SVM_2$). We use different kernels because the SVM parameters are determined with binary search. The parameter $\gamma$ ($E$) is first searched in the range $[0, 20]$. If the corresponding VSSVM is non-empty a binary search for $C$ is performed in the range $[0, 10000]$ so that the VSSVM is still non-empty. It is important to choose the parameters as small as possible because the coverage is maximized (see section 2.4.4).

The dataset used for training is the one that is used in the first model. The computation time is high due to the binary search technique. The model runs approximately fifty minutes on a Pentium IV processor to complete one co-training iteration. The method for evaluation is again the LOO method.

Results for no co-training and one co-training iteration are summarized in tables 4.6 and 4.7. They show that the coverage and accuracy after co-training are decreasing. Although not clear from the tables the changes that occur after one iteration are:

- a correctly classified instance becomes unclassified; and
- an unclassified instance becomes correctly classified; and
- a correctly classified instance becomes incorrectly classified.

For the first and second cases both VSSVMs agree on the classification. In the last case the instance is left unclassified by $VSSVM_1$ before and after co-training, and the incorrect classification is due to $VSSVM_2$ that changes the classification of the instance after co-training.
4.2. CO-TRAINING MODELS

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>24</td>
<td>96%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
</tbody>
</table>

Table 4.6: Results of the second model without co-training.

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>23</td>
<td>92%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
</tbody>
</table>

Table 4.7: Results of the second model after one co-training iteration.

**Discussion of the Second Model**

If the results for no co-training from the first model (table 4.1) are compared with the same results from the second model (table 4.6), it is clear that a binary search heuristic gives better results for the coverage. This result was expected because the SVM is very parameter dependent so the collapse of VSSVMs is also very parameter dependent. The results of the second model after co-training show that the coverage and accuracy are decreasing as well.

The behavior of the second model can be explained on the level of VSSVMs and on the level of the unanimous-voting rule employed as the classification rule for the co-training. This coincides with the discussion of the first model in subsection 4.2.1.

**Conclusion of the Second Model**

The second model tries to preserve the performance of the VSSVMs by searching for the optimal parameters. However, the version-space algebra is still not preserved for VSSVMs. Therefore, the same conclusion of the first model holds for the second model.

### 4.2.3 Third Model

**Description**

The third model is another extension of the first one. The only difference with the first model (figure 4.2) is that the instances in $I_1$ and $I_2$ are not added to $L_1$ and $L_2$, respectively. This preserves the VSSVMs so their generalization performance is not lost as in the first two models.
Input: an instance $x$.

Output: classification for $x$.

$c_1 =$ label predicted by $VSSVM_1$.

If $c_1 = 0$ then $c_1 =$ label predicted by $iVSSVM_1$.

$c_2 =$ label predicted by $VSSVM_2$.

If $c_2 = 0$ then $c_2 =$ label predicted by $iVSSVM_2$.

If $c_1 = c_2$ then assign this label.

Else:

If $c_1 = 0$ and $c_2 \neq 0$ then $c_2$ is the label.

Else if $c_1 \neq 0$ and $c_2 = 0$ then $c_1$ is the label.

Else the label is 0.

Figure 4.13: Co-training classification scheme for the third model.

$I_1$ and $I_2$ are used to train two new VSSVMs, $iVSSVM_1$ and $iVSSVM_2$. These new VSSVMs are called on-top VSSVMs. An on-top VSSVM is used to classify instances that are in the volume of its base VSSVM.

The classification scheme for this model is outlined in figure 4.13. It is a variant of unanimous-voting. First $VSSVM_1$ may try to classify the instance. If it leaves the instance unclassified and $iVSSVM_1$ is non-empty then $iVSSVM_1$ is used to predict the label. The same procedure holds for $VSSVM_2$. If the predictions of the ensembles ($VSSVM_1$ and $iVSSVM_1$) and ($VSSVM_2$ and $iVSSVM_2$) coincide the label is assigned. If the predicted labels are different the scheme checks if exactly one ensemble leaves the instance unclassified. The output is then the prediction of the ensemble that classifies the instance. When ensembles predict different labels the instance is left unclassified because it is not known which label is correct.

Experiment

The on-top VSSVM $iVSSVM_1$ ($iVSSVM_2$) is represented by $iSVM1$ ($iSVM2$) which has the same parameters as $SVM_1$ ($SVM_2$) used in $VSSVM_1$ ($VSSVM_2$). The parameters are those from the first model. The training data is the artificial dataset used for the previous two models and testing is done with the LOO method.

The results for no co-training are the same as those from the first model (table 4.1). The results after one co-training iteration are given in table 4.8. Although not clear from this table the results indicate that:
4.2. CO-TRAINING MODELS

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>19</td>
<td>76%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>6</td>
<td>24%</td>
</tr>
</tbody>
</table>

Table 4.8: Results of the third model after one co-training iteration.

- an unclassified instance becomes correctly classified; and
- an incorrectly classified instance becomes unclassified.

Thus, for this dataset the model is able to increase the coverage and to filter noise. In the following two sections both cases are analyzed.

Unclassified Instance becomes Correctly Classified.
The instance $x_{\text{loo}}$ that becomes correctly classified is $(0.15, 0.2)$. Since the evaluation is realized with LOO the classification of $x_{\text{loo}}$ is analyzed for its own LOO set.

When no co-training is performed $VSSVM_1$ and $VSSVM_2$ leave $x_{\text{loo}}$ unclassified. Both VSSVMs are non-empty on the LOO set.

When co-training starts it checks if $VSSVM_1$ is able to classify $x_{\text{loo}}$. This is not the case so $VSSVM_1$ constructs informative instances and $VSSVM_2$ tries to classify them. The labeled set $I_1$ is then used to train $iVSSVM_1$. $iVSSVM_1$ classifies $x_{\text{loo}}$ as negative (figures 4.14 and 4.15).

The next step in co-training is that $VSSVM_2$ generates informative instances. However, $VSSVM_1$ cannot classify the informative instances from $VSSVM_2$ so $iVSSVM_2$ is empty. Therefore, the output of the model is the classification of $iVSSVM_1$ and this is the correct label.

Incorrectly Classified Instance becomes Unclassified.
The instance $x_{\text{loo}}$ that becomes unclassified is $(0, 0.25)$. Analysis of the no co-training result shows that $VSSVM_1$ does not classify $x_{\text{loo}}$ and $VSSVM_2$ classifies $x_{\text{loo}}$ as positive. This is the wrong label for $x_{\text{loo}}$. Both VSSVMs are non-empty on the LOO set for $x_{\text{loo}}$.

Co-training uses $iVSSVM_1$ because $x_{\text{loo}}$ is left unclassified by $VSSVM_1$. $iVSSVM_1$ is trained on $I_1$ and classifies $x_{\text{loo}}$ as negative (the correct label). Figures 4.16 and 4.17 show this decision. In these figures are the axis chosen differently. $iVSSVM_1$ and $VSSVM_2$ contradict each other on the label. Therefore, the classification algorithm leaves $x_{\text{loo}}$ unclassified.
4.2. CO-TRAINING MODELS

Figure 4.14: Boundary of \( iSVM_1 \) representing \( iVSSVM_1 \) when \( x_{loo} \) is added as negative.

Figure 4.15: Boundary of \( iSVM_1 \) representing \( iVSSVM_1 \) when \( x_{loo} \) is added as positive.

Figure 4.16: Boundary of \( iSVM_1 \) representing \( iVSSVM_1 \) when \( x_{loo} \) is added as negative.

Figure 4.17: Boundary of \( iSVM_1 \) representing \( iVSSVM_1 \) when \( x_{loo} \) is added as positive.
4.2. CO-TRAINING MODELS

Discussion of the Third Model

The third model keeps the base VSSVMs intact so that there is no negative effect on their performance due to co-training. The informative instances are used to train new VSSVMs which are called on-top VSSVMs. An on-top VSSVM is used to classify instances that are in the volume of the base VSSVM. The results from the experiments show that the third model is able to increase the coverage (i.e., to assign a correct class label to a previously unclassified instance) as well as to filter noise (i.e., to label an instance as unclassified when it is classified incorrectly before co-training). However, it is also possible that this model harms coverage and accuracy. This is due to the fact that most of the negative cases for VSSVMs described in subsection 4.2.1 can hold for the on-top VSSVMs. To overcome this problem we propose to apply the volume-extension approach to the base and on-top VSSVMs as described in subsection 4.2.1. By using the approach for the base VSSVMs, we first guarantee correctly labeled informative instances and only after that we train on-top VSSVMs.

Unanimous-voting is a stabilization factor for the third model as well. The explanation coincides with that from subsection 4.2.1. However, due to the on-top VSSVMs it is not possible in the third model that an incorrectly classified instance becomes correctly classified after co-training (case 3 from subsection 4.2.1). When VSSVM\(_1\) and VSSVM\(_2\) assign the same but incorrect label no on-top VSSVMs are used and the incorrect classification is kept. When one base VSSVM does not classify and the other misclassifies then the best case scenario is that the on-top VSSVM contradicts with the predicting VSSVM so that the instance becomes unclassified. An analogue reasoning can be made to see that a correctly classified instance cannot become incorrectly classified (case 4 from subsection 4.2.1).

Conclusion of the Third Model

In this section we introduced our third model for co-training with VSSVMs. The main advantage of this model compared with the previous two models is that the coverage and accuracy of the base VSSVMs are guaranteed to be preserved. The on-top VSSVMs can improve the coverage of the co-trained schemes because they classify instances in the volume of the base VSSVMs. Of course the coverage can be improved only if the negative cases for VSSVMs described in section 4.2.1 are overcome. This can be done by applying the volume-extension approach.
4.2. CO-TRAINING MODELS

4.2.4 Fourth Model

Description

The third model was the first co-training approach that did not harm coverage on the artificial dataset. However, the second model showed better results when no co-training is performed (see tables 4.1 and 4.6). Therefore, it is interesting to combine the second and third model.

The fourth model is similar to the third except that a search is performed for the optimal SVM parameters when:

- the base VSSVMs are trained with the training data; and
- the on-top VSSVMs are trained on the labeled informative instances.

The classification scheme is that of the third model.

Experiment

The training data is the artificial dataset used for the previous three models and the testing is done with the LOO method. Experiments showed that one co-training iteration of this model does not harm or benefit the coverage (which was already very large). Table 4.9 shows this result. To test the fourth model more co-training iterations are performed. This is tested using two scenarios.

In the first scenario the base VSSVM, VSSVM\textsubscript{1}, generates informative instances which are classified by iVSSVM\textsubscript{2}. If iVSSVM\textsubscript{2} is not available we check if iVSSVM\textsubscript{1} is available to generate informative instances. These instances are then classified by VSSVM\textsubscript{2}.

It is possible that a non-empty iVSSVM\textsubscript{2} cannot classify any of the informative instances from VSSVM\textsubscript{1}. Thus, no useful labeled instances are added. The second scenario overcomes this problem using the technique of the first scenario when iVSSVM\textsubscript{2} is not available, i.e., it checks if there is a non-empty iVSSVM\textsubscript{1} to generate informative instances. These instances are classified by VSSVM\textsubscript{2}. Figure 4.18 gives an outline of the second scenario.

Note that we always use the combination base VSSVM and on-top VSSVM of the other ensemble. It is not allowed that the base VSSVMs are used to classify instances for each other because this already happened in the first iteration. The combination of two on-top VSSVMs is also not allowed because their performance is not complementary on the artificial dataset.

Experimental results with more co-training iterations showed no improvement of the coverage. Analysis of the instance that is left unclassified shows
If \( iVSSVM_2 \) is non-empty then
VSSVM\(_1\) generates informative instances \( I_1 \).
Use \( VSSVM_2 \) to classify these instances.

If nothing was labeled and \( iVSSVM_1 \) is non-empty then
\( iVSSVM_1 \) generates informative instances \( I_1 \).
Use \( VSSVM_2 \) to classify these instances.

Else if \( iVSSVM_1 \) is non-empty then
\( iVSSVM_1 \) generates informative instances \( I_1 \).
Use \( VSSVM_2 \) to classify these instances.

**Figure 4.18:** Second scheme for more co-training iterations.

<table>
<thead>
<tr>
<th></th>
<th>Nr. of instances</th>
<th>% of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctly Classified Instances</td>
<td>24</td>
<td>96%</td>
</tr>
<tr>
<td>Incorrectly Classified Instances</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Unclassified Instances</td>
<td>1</td>
<td>4%</td>
</tr>
</tbody>
</table>

**Table 4.9:** Results of the fourth model.

that co-training cannot progress after the third iteration. The reason is that the VSSVMs are not able to classify informative instances for each other.

**Discussion of the Fourth Model**

Our experiments with the fourth model show no progress and no decline. Co-training already stops after three iterations. Thus, both ensembles of VSSVMs are not as complementary as desired although they are based on a RBF SVM and a lower-order polynomial SVM. Visual inspection enforces this hypothesis because it is seen that the informative instances from all VSSVMs lie close to each other or lie more or less on the same curve. This means that the second assumption of the co-training model (uncorrelated views) does not hold.

The discussion of the third model also holds for the fourth model as well. So, we propose to use the volume-extension approach for the base and/or on-top VSSVMs.

**Conclusion of the Fourth Model**

The fourth model tries to boost the coverage of the third model by adding the binary search heuristic for the optimal SVM parameters. The coverage
and the accuracy of the base VSSVMs are guaranteed to be preserved and on-top models try to improve the coverage and accuracy. However, the search for the optimal parameters does not solve the problem that the version-space algebra is not preserved. Therefore, we have to apply the volume-extension approach.

We believe that the fourth model in general will give better results than the third model due to the parameter sensitivity of VSSVMs.
Chapter 5

Conclusion

In this chapter we present our answer to the research statement and recommendations for future research. The answer is provided in section 5.1. The recommendations are summarized in section 5.2.

5.1 Thesis Conclusion

To answer our problem statement we proposed four models for co-training of version space support vector machines (VSSVMs). The first two models are based on the basic co-training algorithm. The experiments of these two models showed that co-training in its original form can harm the accuracy and coverage of VSSVMs. Our analysis reveals that these results are due to the fact that the version-space algebra is not preserved for VSSVMs. Therefore, we concluded that by increasing the size of training data on some parts of the instance space the classification boundaries of VSSVMs do not converge, instead they diverge. This result has positive and negative effects on instance classification. To reduce the negative effects we introduced the next two models for co-training of VSSVMs. These models learn so called on-top VSSVMs from new-labeled data. The on-top VSSVMs are allowed to classify only those instances that cannot be classified by the base VSSVMs. In this way the problem with the version-space algebra is left for the on-top VSSVMs.

The problem with the version-space algebra cannot be solved by adding many-level on-top VSSVMs (see our experiments with the fourth model). Therefore, we proposed to apply the volume-extension approach for co-trained VSSVMs. Applying the approach is not a trivial task. First, we have to determine with an internal cross-validation process a frequency for each type of desirable and undesirable instance-classification transformations
due to co-training. Then, using these frequencies we have to decide how to apply the volume-extension approach: by increasing or decreasing the constant $C$. Once the direction of $C$ is set, the volume-extension approach has to be applied by maximizing the accuracy and coverage of the VSSVMs using an internal cross-validation process.

The results presented so far are closely related to our problem statement. For the sake of completeness we repeat the statement:

*Is it possible to apply co-training for VSSVMs to improve their coverage?*

Summarizing our results we provide an answer to the problem statement as follows:

*Co-training of VSSVMs can improve their coverage if the version-space algebra is preserved. When it is not preserved, the volume-extension approach has to be applied.*

## 5.2 Future Research

We foresee three directions for future research. The first direction has to overcome the problem with the version-space algebra for VSSVMs. We plan to analyze VSSVMs and SVMs for different kernels to identify the main reasons for this problem. Using our analysis a new SVM-like algorithm has to be designed that guarantees preservation of the version-space algebra. One idea for such an algorithm is to attach weights for each instance correctly classified by a SVM and then to use them in the optimization problem to find an optimal SVM.

The second direction for future research is to test experimentally the volume-extension approach when it has to improve the co-training with VSSVMs. Using the experiments we have to determine how to apply the volume-extension approach in general: (1) on the level of base VSSVMs and/or on the level of on-top VSSVMs and (2) before and/or after co-training.

The last direction for future research is to investigate how to construct complementary VSSVMs. The complementary VSSVMs maximize the training utility of informative instances and decreases the computation time of co-training. This research is related to the construction of new types of informative instances. One possibility is to construct instances that connect dense regions in the training data. It is believed that this type of informative instances will not cause drastic changes in the classification boundaries of VSSVMs.
Bibliography


