Transductive Pairwise Classification

Transduktive Paarweise Klassifizierung
Bachelor-Thesis von Ji-Ung Lee aus Viborg, Denmark
Oktober 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Transductive Pairwise Classification
Transduktive Paarweise Klassifizierung

vorgelegte Bachelor-Thesis von Ji-Ung Lee aus Viborg, Denmark

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten: Eneldo Loza Mencía

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den Oktober 20, 2013

(Ji-Ung Lee)

Abstract

Semi-supervised machine learning algorithms are able to handle labeled and unlabeled data. This allows them to benefit from a bigger dataset and therefore more training examples than common supervised approaches. For this reason it seems natural to adapt semi-supervision to supervised learning approaches such as the learning by pairwise comparison using label preferences approach, which can be further specialized to pairwise classification. This bachelor thesis compares pairwise classification using a supervised classifier with pairwise classification using a semi-supervised classifier. The used classifier is the support vector machine (SVM), an ordinary supervised classifier. The concept of semi-supervision is adapted to the support vector machine by using transductive inference. A matter of special interest is the algorithm for finding label assignments of the unlabeled data in *SVMlight*. After giving a theoretical background of preference learning, pairwise classification and support vector machines, I will introduce the transductive pairwise classification approach and compare it with the supervised pairwise classification approach in various evaluations on multi-class datasets.

Contents

1 Introduction

With the rapid growth of the world wide web *knowledge engineering* is a field gaining more and more importance nowadays. Every day huge amounts of data are flowing into the web unfiltered. For processing those huge amounts of data in a fast and useful way many approaches have been made. Big companies like *Google*[1] are trying to improve their information processing in their struggle over market domination. In addition there are many *Open Source*[2] projects which require a fast information processing like *DBpedia*[3]. Furthermore, many online retrailers use *recommender systems* to provide their customers with the latest products fitting to their interests [6].

To handle such big amounts of data properly, it is important to extract only the relevant information out of it. This is done in a preprocessing step which generates the input data for machine learning algorithms in a proper form.

It is common sense that water is wet.

Attributes:	{It is common sense that water is wet .}	{common 1 is 2 it 1 sense 1 that 1 water 1 wet 1}
Class label:	general knowledge	general knowledge

Figure 1.1: For the sentence above, a various number of representations is possible.

Figure 1.1 shows various possibilities of preprocessing the data. It is reasonable that the number of commas might be a bad attribute for classifying a text into categories. One might notice the loss of the ordering of the words in the second representation. For data where the ordering itself is important it could be stored additionally with the attributes. In this thesis, the used data is already preprocessed in a machine learning applicable way. A specification of the data is given in Section 4.1.1. The preprocessing itself is not an issue in this thesis and will not be discussed further.

After the preprocessing, the input data consists of the *attributes* of one data object also called an *instance* and an assignment to a *class* which is called a *label* [4]. A whole dataset is then defined by a set of attributes and a set of class labels and the data, which is often represented as *feature vectors*.

Depending on the amount of information available in the given input data, various approaches of machine learning like *classification*, *clustering* or *ranking* are applicable. In common *classification* the task is to find the class label which specifies a given instance the best [3]. In *ranking*, which is part of *preference learning*, the goal is to find partial orders in datasets, similar to some given orders in the input data [9]. *Classification* and *ranking* can be further differentiated by the number of classes in the input data. For Example a very common subtask for *classification* is the *multi-class* classification, where the challenge is to build a classifier that distinguishes between all given classes [5]. A more detailed description of *preference learning* and *ranking* will be given in Chapter 2.1. In *clustering*, the task is to detect clusters in a set of given data points or instances, which then can be described by a specific class [3, 2]. The datasets used in *clustering* are usually not labeled. So the challenge is to find new classes for nearby instances without any further input.

When looking at these various machine learning approaches, one big difference is noticeable. Some

[1] http://www.google.de/ , Date of last access: 26th October 2013
[2] http://opensource.org/ , Date of last access: 26th October 2013
[3] http://dbpedia.org/About , Date of last access: 26th October 2013
[4] In machine learning terms labeled instances are also called *examples*.
[5] The simple case is the binary classification task, i.e. to build a classifier for a learning problem with only two classes.

approaches do require labeled instances as input data and some do not. So those approaches were divided into two big classes of machine learning. *Supervised learning* and *unsupervised learning*. In *supervised learning*, the input data consists of labeled instances. Since assigning the correct class label for each instance (also called *labeling*) has to be done manually by humans, it is *supervised*. The input data in *unsupervised* machine learning tasks is not labeled, thus without a supervisor providing examples for the learning algorithm. So tasks like *clustering* are *unsupervised learning* tasks, while *classification* and *preference learning* are *supervised learning* tasks.

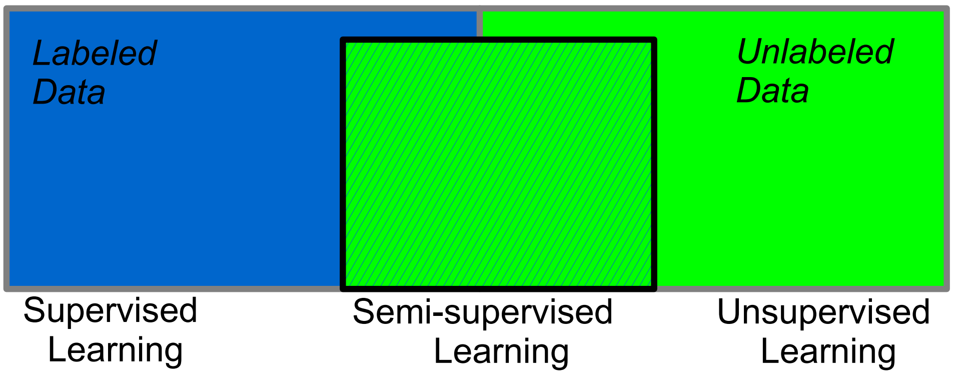


Figure 1.2: *Semi-supervised learning* approaches use labeled and unlabeled data.

One may notice that correctly labeled data is expensive because they have to be labeled manually and the resources are limited. But using only unlabeled data is also impracticable since unsupervised approaches cannot solve all machine learning tasks [13] [6]. So in the last two decades there were many approaches made trying to find a way to make use of datasets with only a few labeled data points and many unlabeled data points [13] as shown in Figure 1.2. Those approaches were categorized in a new category called *semi-supervised learning* and they have shown to be very useful in some cases [13]. Since they use both kinds of input data, they can be seen as a cost-benefit trade-off. This raises the question whether a machine learning task using a supervised learning approach could be solved more efficiently by using a semi-supervised learning approach.

[6] One example would be filtering spam mails. Since there is a big variety of them, at least some examples have to be given.

2 Background

2.1 Preference Learning

Learning preferences is a task which occurs in many fields. One example would be a search engine. A search engine which recommends exactly one solution for a query is not very applicable. The solution might be wrong or the query could be not well formulated. So the requirement of a good search engine is to find a list of results generated by the user's query. Furthermore, those results should be ordered by relevance to the query, meaning that some results have a higher preference than others.
In many machine learning tasks like classification, the training dataset often consists of instances and class labels. However, in preference learning the training dataset could also contain a finite set of preferences. The goal of preference learning is now to predict a correct ranking[1] using these preferences. Depending on the input data, J. Fürnkranz and E. Hüllermeier provide a general classification of preference learning in three big tasks: *label ranking*, *instance ranking* and *object ranking* [6].

2.1.1 Label Ranking

In this bachelor thesis I will focus on the task of label ranking described by J. Fürnkranz and E. Hüllermeier [9]. The purpose of label ranking is to learn an ordering of the labels according to a given instance. So be X a set of instances and Y a set of class labels. Generally, the goal then is to find a ranking function $f_x(Y)$ which provides a ranking for the set of labels Y for the given instance $\mathbf{x}$.
The input data consists of three things:

- a set of n training instances $\{ x_i \mid i = 1,2,...,n \}$, typically represented by a feature vector
- a set of l class labels $\{ y_i \mid i = 1,2,...,l \}$
- preferences for each instance x_i in the form $y_1 \succ_{x_i} y_2$ which means, that for instance x_i, y_1 is preferred against y_2

Usually the given preferences are a set of pairwise preferences. As mentioned before the output data is a ranking of the class labels for each instance x in the form $y_i \succ_x \ldots \succ_x y_j$, so any permutation of the set of class labels Y for each instance.

From another point of view, the task of finding an ordering of the labels can be seen as a more generalized way of classification. In classification we try to find the most relevant class label for an instance. Now, instead of only computing the most relevant class, we find a ranking of the class labels for each instance. Just as one can be seen as a generalized version of the other, Fürnkranz and E. Hüllermeier describe classification to be a specialized form of label ranking [6]:

"*Classification: A single class label y_i is assigned to each example x_l. This implicitly defines the set of preferences $\{y_i \succ_{x_l} y_j | 1 \le j \ne i \le k\}$.*"

[1] Please keep in mind that I will be using the term "ranking" the same way as J. Fürnkranz and E. Hüllermeier [6]. So when I describe something as "ranking" this does not imply that a total order of preferences exists.

2.1.2 Learning by Pairwise Comparison

To learn preferences or a ranking function, many strategies have been created. One way to find a utility function is using regression learning or ordered classification, both are already well established approaches in machine learning [6]. Some try a model-based approach by making certain assumptions about the relation structure before initiating the training phase. One example would be the assumption of a lexicographic order to learn lexicographic preference models like Yaman et al. described [16]. Others try a local aggregation approach in some ways similar to the nearest neighbor algorithm [1]. The approach, which will be introduced and used in this thesis, is to learn a binary preference relation for a set of given class labels [6].

Learning by pairwise comparison is an approach to train preference relations provided by J. Fürnkranz and E. Hüllermeier in [9] which will be explained in the following. The idea of decomposition of multi-class learning problems is already very common in conventional classification. Furthermore, there are many ways for decomposing multi-class problems into a set of pairwise problems, as *all pairs*, *1-vs-1* or *round robin* learning. Motivated by a good performance of those techniques in conventional classification, the approach was to adapt pairwise comparison to preference learning. This approach was especially interesting for label ranking, which can be mapped to conventional classification as shown in Section 2.1.1.

Pairwise Classification

In pairwise classification the key idea is to decompose a multi-class problem with a set of k-classes into $k(k-1)/2$ sets of two classes. So a set of class labels $Y = \{y_1, y_2, \dots, y_k\}$ is transformed into many sets of binary problems in the form of $Y_1 = \{y_1, y_2\}$, $Y_2 = \{y_1, y_3\}$, ..., $Y_{k(k-1)/2} = \{y_{k-1}, y_k\}, \subset Y$. After the decomposition, for each pair of class labels $Y_1, \dots, Y_{k(k-1)/2}$ a separating model $M_{i,j}$, $1 \leq i < j \leq k$ is trained. Since every set of class labels Y_l now only contains two labels, the set of training instances X used as the input for each model can be reduced to a smaller set $X_l \subset X$. X_l, only containing the instances labeled with class labels of Y_l. Every model $M_{i,j}$ now separates the objects with label y_i from objects with label y_j.

$$M_{i,j}(x) = \begin{cases} y_i & \text{if } x \in y_i \\ y_j & \text{if } x \in y_j \end{cases} \tag{2.1}$$

After training the $k(k-1)/2$ models, a query instance $x \in X$ is given to all these models for classification. Now every model $M_{i,j}$ predicts either the class label y_i or y_j for the instance and the resulting prediction is calculated by combining all outcomes. A simple way to get the overall prediction would be a non-weighted voting. Each prediction of the models $M_{i,j}$ is counted as a vote for label y_i or y_j. The resulting class label would be the one with the most votes.

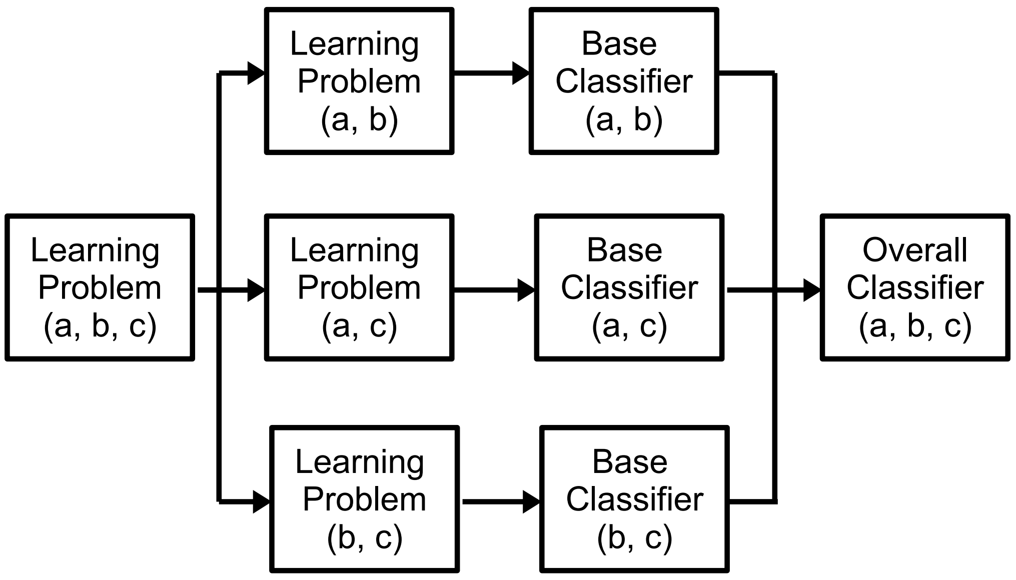


Figure 2.1 (a) Building the overall classifier

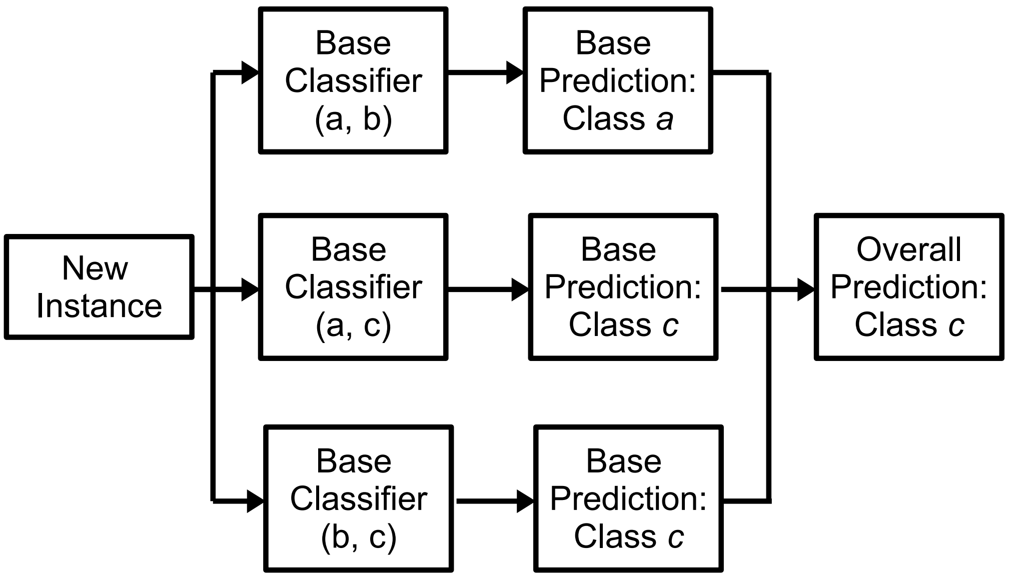
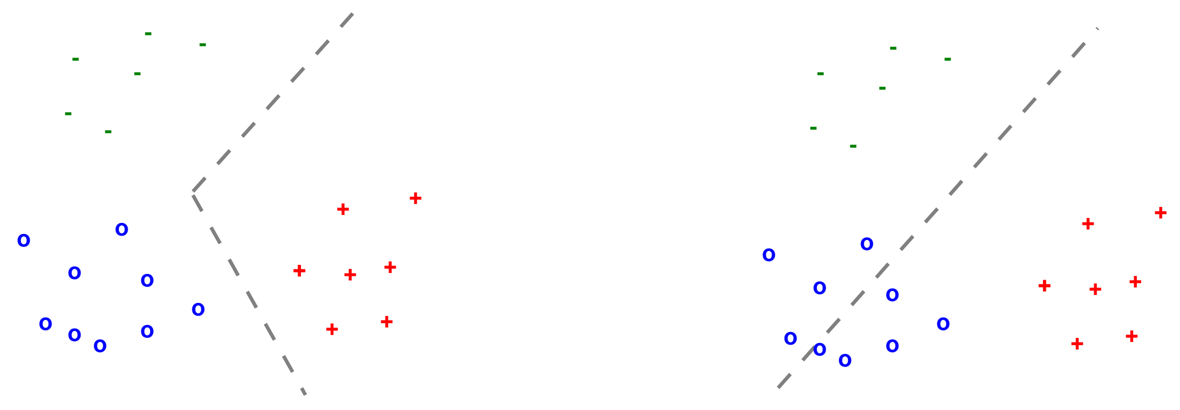

Base Classifier (a, b)
Base Prediction: Class *a*
New Instance
Base Classifier (a, c)
Base Prediction: Class *c*
Overall Prediction: Class *c*
Base Classifier (b, c)
Base Prediction: Class *c*

Figure 2.1 (b) Predicting new instances

Figure 2.1 Pairwise classification

Figure 2.1 (a) illustrates the building phase of an overall classifier using the pairwise classification approach. A learning problem with the classes $\{a, b, c\}$ is decomposed and a separating model for each binary learning problem is build. Figure 2.1 (b) shows how new instances are predicted. Every base classifier predict one of the two classes they are trained for. Afterwards a combination of the predictions is used to generate the prediction of the overall classifier. For the evaluations in this thesis a simple voting strategy is used, i.e. the class with the most votes is predicted.

There are some reasons given by J. Fürnkranz and E. Hüllermeier [9] for choosing the pairwise classification approach against other decomposition techniques like the *one-vs-all* approach. In *one-vs-all* learning a single model is trained for each label, which can cause very complex models. One big advantage decomposing the learning problem in pairwise classification tasks are the resulting subproblems, which are much less computationally complex than approaches like *one-vs-all*. At first glance, pairwise classification seems to have a big disadvantage due to the amount of separating models which have to be trained compared to *one-vs-all* classification. Since there are $k(k-1)/2$ pairs of class labels, $k(k-1)/2$ models have to be trained instead of only k models like in the *one-vs-all* approach. But experiments showed that the benefits from smaller training sets for each of those problems are often bigger than the loss [8]. Furthermore, pairwise classification problems are usually much easier to solve, since the only task is to find a model separating only two class labels. So a very simple base learner is sufficient for building the classifier. An illustration of this benefit can be seen in Figure 2.2.



One-vs-all classification: Even with only three different classes a more complex classifier is necessary.

Pairwise classification: Since every pair of classes are learned separately, a simple separating model is sufficient.

Figure 2.2 A comparison between the one-vs-all approach and pairwise classification.

2.1.3 Performance Measures

To evaluate the performance of a label ranking approach several criteria can be used. In this thesis I am using the following criteria[2] to compare the performance of both approaches [3]. In addition, I will use cross-validation[3] to prevent an *overfitting* of the learner. Overfitting is a well-known machine learning problem which often occurs with small training sets or training sets with an unimportant attribute which is shared by every instance [3]. For Instance, by training a classifier for identifying salmons on only one example, the classifier will be able to recognize only salmons with attributes matching the attributes of the training example. So salmons which slightly differ in some attributes such as color of skin or shape will not be recognized as salmons at all.
Referring to the notation of a separating model $M_{i,j}$ described earlier, I will use $M_{+,-}(x)$ which maps 1 for an instance x classified as positive (e.g. $x \in class\ a$) and -1 for the same instance classified as negative (e.g. $x \notin class\ a$).

[2] For the mathematically formal definitions see [3].
[3] Cross-validation will be explained in 2.1.4

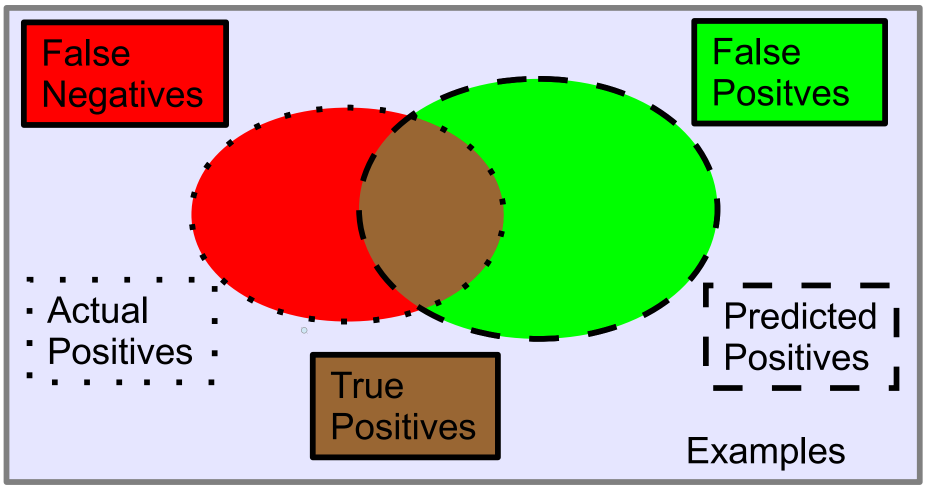


Figure 2.3 An example of the actual positives (dashed) and the predicted positives (dotted). The true positives (brown), false positives (green), true negatives (red) and false negatives (gray) are colored differently.

True Positive Rate

Instances which are classified correctly as positives are called *true positives*. The *true positive rate (tp-rate)* is defined as:

$$\textit{tp-rate} = \frac{|\{x_i : M_{+,-}(x_i) = 1 \land x_i \in X_{positive}, i = 1, \ldots, n\}|}{|X_{positive}|} \tag{2.2}$$

with n being the number of the training examples. So the *tp-rate* is calculated by the number of all correctly classified positives divided by the number of all instances labeled as positives and gives us a proportion of the positives which are correctly classified. In Figure 2.3 the *tp-rate* would be the fraction of the brown part of the dotted ellipse.

False Positive Rate

False positives are the actual negatives classified as positives. The *fp-rate* is defined similar to the *tp-rate*:

$$\textit{fp-rate} = \frac{|\{x_i : M_{+,-}(x_i) = 1 \land x_i \in X_{negative}, i = 1, \ldots, n\}|}{|X_{negative}|} \tag{2.3}$$

So the *fp-rate* can be interpreted as a probability that a negative is classified wrongly as a positive. In Figure 2.3 the false negatives are only in the green part of the dashed ellipse. So the *fp-rate* would be the fraction of the green part to the negative examples (everything except the dotted ellipse on the left side).

Recall

Recall is calculated in the same way as the *tp-rate* and is used in context of search engines or in information retrieval [3].

Precision

Precision is an alternative measure and seems similar to the *tp-rate* at first sight, but is quite different. It is defined as follows:

$$Precision = \frac{|\{x_i : M_{+,-}(x_i) = 1 \land x_i \in X_{positive}, i = 1, \ldots, n\}|}{|\{x_i : M_{+,-}(x_i) = 1, i = 1, \ldots, n\}|} \tag{2.4}$$

A good differentiation of the *tp-rate* and the *precision* is given by P. Flach in [3]:

"*[. . .] while true positive rate is the proportion predicted positives among the actual positives, precision represents the proportion of actual positives among the predicted positives.*"

The *precision* would be the fraction of the brown colored true positives to the sum of the predicted positives represented by the dashed ellipse on the right side when looking at Figure 2.3.

Accuracy

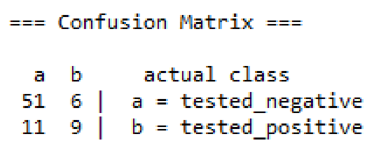
Accuracy is the simplest measure for the performance and is the proportion of the correctly classified instances. It is calculated by:

$$Accuracy = \frac{|\{x_i : M_{+,-}(x_i) = 1 \land x_i \in X_{positive}, i = 1, \ldots, n\}| + |\{x_i : M_{+,-}(x_i) = -1 \land x_i \in X_{negative}, i = 1, \ldots, n\}|}{|X|} \tag{2.5}$$

X is the whole set and the sum in the numerator are the correctly classified instances. In Figure 2.3 the *accuracy* would be fraction of the sum of the brown part and the gray part to the whole example space.

Confusion Matrix

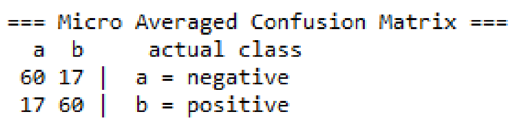
A confusion matrix is a table which illustrates the results of a classification. The confusion matrix holds the results for the classes predicted as positive and negative and the actual class labels. So the sum of one row of a confusion matrix holds the actual positive or negative instances. The sum of one column is the number of predicted positive or negative instances. A confusion matrix can also contain more than two classes. The correctly classified instances can be read directly from the diagonal of the matrix. An illustration is given in Figure 2.4 (a). The correct predictions for each class is an important factor for the performance of a classifier. But to compare the performance on different datasets it is only interesting how good the overall performance was [2]. Therefore the microaveraged confusion matrix sums up the results for each class grouping them into two classes: The correctly predicted positives and the correctly predicted negatives. Figure 2.4 (b) represents the microaveraged confusion matrix to the confusion matrix in Figure 2.4 (a). Note that the number of false positives and false negatives will be equal since every false positive in one class is a false negative in another class.



```
=== Confusion Matrix ===

  a  b    actual class
 51  6 |  a = tested_negative
 11  9 |  b = tested_positive
```

2.4 (a) Confusion Matrix: The sum of the first row would be the number of instances labeled class a. The sum of the first column is the number of the instances, which were predicted class a.



```
=== Micro Averaged Confusion Matrix ===
  a  b    actual class
 60 17 |  a = negative
 17 60 |  b = positive
```

2.4 (b) Microaveraged Confusion Matrix: Since the confusion matrix in 2.4 (a) contains only two classes, the sum of correctly predicted positives and correctly predicted negatives is equal.

Figure 2.4 Confusion Matrices

2.1.4 Cross-validation

Since there are many factors depending on the input data and because I want to avoid overfitting, it is reasonable to create several results on independent training and test sets. If there is enough data, it could be separated into a training set and a test set. The training set is then used to build appropriate models. For the evaluation, the instances of the test set are classified and the predictions are compared to the actual class of the instances. But often there is not enough data to form appropriate test and training sets. This should however not be an excuse to ignore the risk of overfitting a model. Therefore *cross-validation* is an approach which is often applied when using small datasets [3]. In *cross-validation* the dataset is divided into k-parts or *folds*. Then one fold is set aside for testing and the remaining k-1 parts are used as the training set. This procedure is repeated k-times, every time using another fold as the test set. Though the training sets in cross-validation are not completely independent, it is now possible to compute some performance measurements of the learning algorithm for k-folds. The results for each fold i hold the performance of the classifiers trained by the remaining $k - 1$ folds.

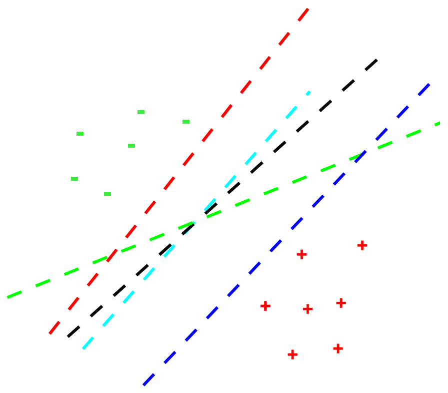
2.2 Support Vector Machines

Among many classifiers *support vector machines (SVM)* are a very popular one. They were first introduced by V. Vapnik [14]. Since then several actual implementations of support vector machines were programmed. For the experiments of this thesis I am using an adaption of *SVMlight*, a SVM implementation by T. Joachims [10] written in C. In the following I will give an introduction into the theory behind support vector machines. Since I use SVMs as base classifiers of the $k(k-1)/2$ binary classification problems, I will only have a look at SVMs for classification tasks with two classes. I will be referring to the classification task of finding separating planes for the classes $\{-1,+1\}$ for a better readability. Please note that in a multi-class classification task with the classes $\{a,b,c\}$, the goal would be to find separating hyperplanes for the subtasks $\{a,b\}$, $\{a,c\}$ and $\{b,c\}$. Each of these subtasks can be mapped to the task of predicting if the instance x_i belongs to class a $(+1)$ or not (-1). For the subtask $\{a,b\}$ a separating model $M_{+,-}$ will predict the following:

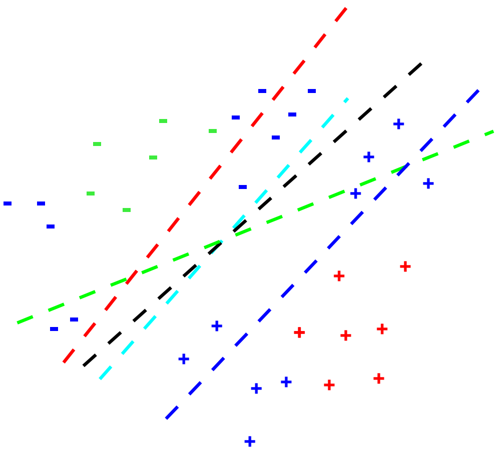
$$M_{+,-}(x_i) = \begin{cases} +1 & \text{if } x_i \in a \\ -1 & \text{if } x_i \notin a \end{cases} \tag{2.6}$$

Note that $x_i \notin a$ implies $x_i \in b$ for the subtask $\{a,b\}$.

2.2.1 The Optimal Hyperplane



2.5 (a) Separating Hyperplanes for vector x: Any of this hyperplanes separates the two subsets correctly for x.



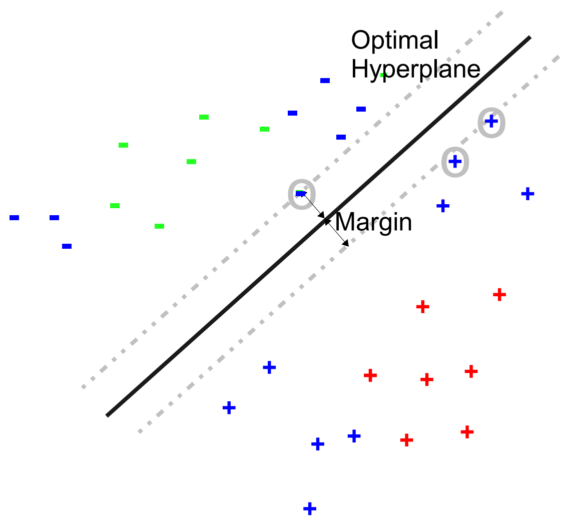
2.5 (b) Separating Hyperplanes including other vectors: When adding new examples z (blue), some of the hyperplanes are suddenly no longer separating hyperplanes for all examples.

Figure 2.5 Separating Hyperplanes

The key idea of the support vector machine is to find an optimal separating hyperplane for two finite subsets of a vector x [15]. So for n data points $\{(x_i,y_i)\}_{i=1}^n$ we have $x_i \in \mathbb{R}^d$ where d is any dimension and $y_i \in \{-1,+1\}$. Basically, when the two subsets are separable, there is an infinite number of separating hyperplanes (Figure 2.5 (a)) and each of them are represented by a function $y(x) = w^T x + w_0$.[4]

[4] A hyperplane can always be described by a function $y(x)$ with the data point x and the normal vector of the hyperplane w.

So each function could be a proper classifier for y_i trained on the vector x and would be applicable for x. But what if we add some new points z with the same $y_i \in \{-1,+1\}$ as examples? Depending on the chosen classifier, the results could be good or bad (Figure 2.5 (b)). So the task is to find an optimal separating hyperplane which gives us the best results for any other point z without having any actual knowledge about it.



The optimal hyperplane separates the two subsets with the maximal margin.

Figure 2.6 The Optimal Hyperplane

To find the optimal separating hyperplane, V. Vapnik defines it by using the *margin*. The margin describes the distance between a decision boundary and the nearest example of a class [3]. Applying the previous description of a hyperplane, the margin can be described using following equation to calculate the distance for any point x_i to the hyperplane:

$$\frac{y(x_i)}{||w||} = \frac{w^T x_i + b}{||w||} \tag{2.7}$$

To find the optimal hyperplane, we have to find the nearest examples x_i which generate the maximal margin. For an easier computation we can now scale the distance $y(x_i)$ of the nearest examples to 1. Now the margin[5] is described by $\frac{1}{||w||}$ and to maximize it, we have to minimize $||w||$. V. Vapnik defines the optimal hyperplane to be the hyperplane with the maximal margin for both classes [15] (Figure 2.6). Further he shows that the optimal hyperplane is unique and therefore can be formulated as an optimization problem.

2.2.2 The Optimization Problem

The goal now is to find an effective method for constructing the optimal hyperplane as defined in the previous section. Therefore V. Vapnik introduced an equivalent formulation of the optimization problem [15]:

[5] Note that we divide the distance of the nearest examples x_i by the length of the normal vector ($||w||$) to normalize the results.

"*Find a pair consisting of a vector ψ_0 and a constant (threshold) b_0 such that they satisfy the constraints*

$$(x_i * \psi_0) + b_0 \geq 1, \quad if\, y_i = 1,$$
$$(x_j * \psi_0) + b_0 \leq -1, \quad if\, y_j = -1, \tag{2.8}$$

and the vector ψ_0 has the smallest norm

$$|\psi|^2 = (\psi * \psi)." \tag{2.9}$$

In other words, we find a vector ψ_0 and a constant b_0 which are used to construct hyperplanes. One hyperplane which describes the upper bound -1 for each example with class y_i = -1 and another hyperplane describing the lower bound +1 for each example with class y_j = +1. Using those two hyperplanes we calculate our ψ_0 which minimizes the second equation $|\psi|^2 = (\psi * \psi)$. [6]
Furthermore, we can ensure that the margin is $\frac{1}{\|w\|}$ since we choose to scale the distance of all nearest examples to 1 and thereby we can say $|y(x_i)| \geq 1$ for all examples. We can describe the problem of maximizing the margin in a common quadratic optimization problem [15, 3]:

$$\begin{gathered}\Phi(w) = min_{w,b}\tfrac{1}{2}\|w\|^2 \\ subject\ to\ y_i(w^T x_i + b) - 1 \geq 0\end{gathered} \tag{2.10}$$

2.2.3 Solving the Optimization Problem

We can solve this quadratic optimization problem using the Lagrange function [15]:

$$L(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \tag{2.11}$$

Now a saddle point has to be found for *L(w, b, α)*. Therefore we derive **(2.11)** for both variables *w* and *b* and get:

$$\begin{gathered}\frac{\partial L(w,b,\alpha)}{\partial w} = w - \textstyle\sum_{i=1}^{n} y_i \alpha_i x_i = 0 \Leftrightarrow w = \sum_{i=1}^{n} y_i \alpha_i x_i \\ \frac{\partial L(w,b,\alpha)}{\partial b} = \textstyle\sum_{i=1}^{n} y_i \alpha_i = 0\end{gathered} \tag{2.12}$$

Using **(2.12)** we can reformulate **(2.11)**:

$$\begin{aligned} L(w,b,\alpha) &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \\ &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i y_i w^T x_i - \sum_{i=1}^{n} \alpha_i y_i b + \sum_{i=1}^{n} \alpha_i \\ &=_{(2.18)} \frac{1}{2}\sum_{i=1}^{n} y_i \alpha_i x_i^T \sum_{j=1}^{n} y_j \alpha_j x_j - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} y_j \alpha_j x_j^T x_i - 0 + \sum_{i=1}^{n} \alpha_i \\ &= -\frac{1}{2}\sum_{i=1}^{n} y_i \alpha_i x_i \sum_{j=1}^{n} y_j \alpha_j x_j + \sum_{i=1}^{n} \alpha_i \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^T x_j \end{aligned} \tag{2.13}$$

[6] Be aware that the vector ψ is the same as the vector w used in **(2.7)**, the normal vector.

To maximize the margin, we now have to find all coefficients α_i^0 with $\alpha_i \geq 0$, $i{=}1,\ldots,$ n and $\sum_{i=1}^n y_i\alpha_i = 0$.[7] V. Vapnik concludes that the values of the α_i^0 which solve the optimization problem correspond to the vectors x_i satisfying the following equation [15]:

$$y_i((x_i^T w_0) + b_0) = 1 \tag{2.14}$$

Those vectors x_i are called *support vectors*. One might notice that the equation above describes all the points with a distance of 1 from the optimal hyperplane and we chose to scale the distance of the closest points of the optimal hyperplane to 1. Going further, we can reformulate **(2.12)** for w_o and we get:

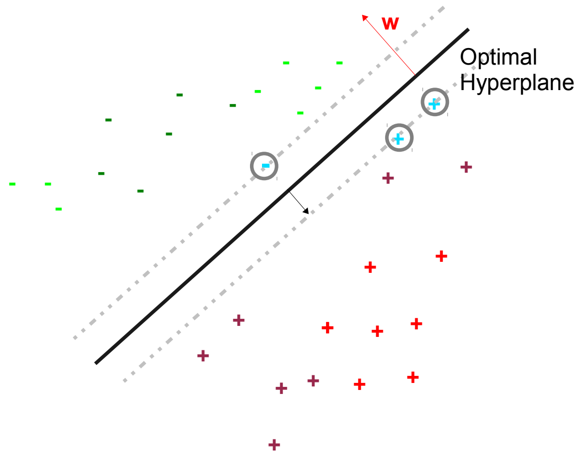
$$w_0 = \textstyle\sum_{i=1}^n y_i \alpha_i^0 x_i$$

Combined with **(2.14)**, V. Vapnik provides following optimal hyperplane [15]:

$$f(x, \alpha_0) = \sum_{i=1}^n y_i \alpha_i^0 (x_s^T x) + b_0 \tag{2.15}$$

with $x_s^T x$ being the inner product of the two vectors.

One important property of the described support vector approach is that there are only a few support vectors needed, e.g. in $\mathbb{R}^2$ the optimal hyperplane is already defined by three data points (Figure 2.7).



The optimal hyperplane is already defined by three data points.

Figure 2.7 Support Vector Approach

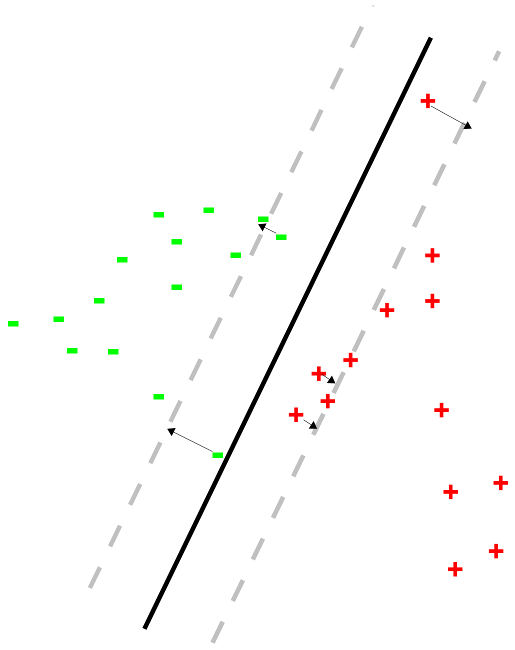
2.2.4 Non-linear separable Datasets

The case described above is only applicable for linear separable datasets. But what if the data is non-linear separable? One solution could be the mapping of the data to a more fitting space, in which the data becomes linear separable. This is also called the *kernel trick* [3]. The basic idea is to apply a kernel

[7] This can also be rewritten as: $\alpha^T y = 0$, with α and y representing all $\alpha_1,\ldots,\alpha_n$ and $y_1,\ldots,y_n$ as a vector. In some literature this is also called the *equation constraint* [10].

function on the data which transforms it into a higher dimension. Since I am only using the linear 'kernel', i.e. the basic approach without any mapping on other spaces, the kernel trick and kernel functions will not be explained.

Another solution for non-linear separable data is to allow a certain error rate. In many learning problems it is often not very feasible to build a classifier which separates the training data perfectly, since this would lead to overfitting. So if allowing errors for some examples lead to more general solutions, it is very desirable to do this. Therefore V. Vapnik provides a soft margin separating hyperplane [15].



The slack variables allow a certain error for some examples, which leads to a wider margin.

Figure 2.8 The soft margin SVM

He introduces the so-called slack variables $\xi_i \geq 0$ for each instance x_i and reformulates the optimization problem for finding a generalized optimal hyperplane [3, 15]:

$$\Phi(w,\xi) = min_{w,b,\xi}\frac{1}{2}|w|^2 + C(\sum_{i=1}^{n}\xi_i) \tag{2.16}$$

The goal is to minimize **(2.16)** by finding vector w under the following constraints:

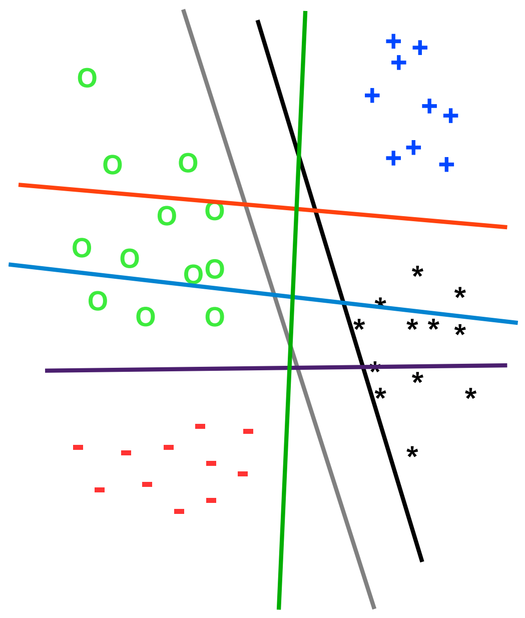
$$\begin{aligned} y_i(w^T x_i + b) &\geq 1 - \xi_i \\ i &= 1,\ldots,n \\ \xi_i &\geq 0 \end{aligned}$$

When looking at those constraints it becomes clear that the ξ_i are variables to correct misclassified examples. The reformulated constraints allow the instance x_i to be inside the margin or for $\xi_i > 1$ even to be on the wrong side of the separating hyperplane. An illustration is given in Figure 2.8 .

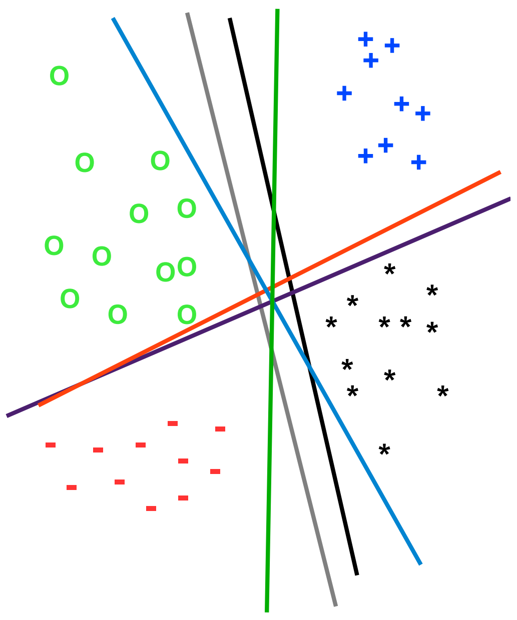
After applying the Lagrangian, this leads to the same equation as in the separable case which has to be maximized:

$$L(w,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^T x_j \tag{2.17}$$

The only things which differ are the constraints $0 \leq \alpha_i \leq C$, $i = 1, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$. Compared to **(2.12)** the constraint to α_i is not only $0 \leq \alpha_i$, but $0 \leq \alpha_i \leq C$. C is referred as the optimal value parameter and it controls how much influence the slack variables have, since it sets a boundary for ξ_i in equation **(2.16)** and for α_i in the constraints. For big C the ξ_i will have a bigger influence and therefore allow only a small margin and for small C the influence of ξ_i will be smaller, resulting in a wider margin. Combined with the *kernel trick*, this also affects the number of support vectors which are necessary to build the support vector machine, since a wider margin will tolerate more misclassification but will also need less support vectors due to a smaller dimension space for the data.



3.1 (a) Inductive approach: For each set of binary classification tasks a separating model is built with maximum margin.



3.1 (b) Transductive approach: All the instances are used in every binary classification task for building the separating model.

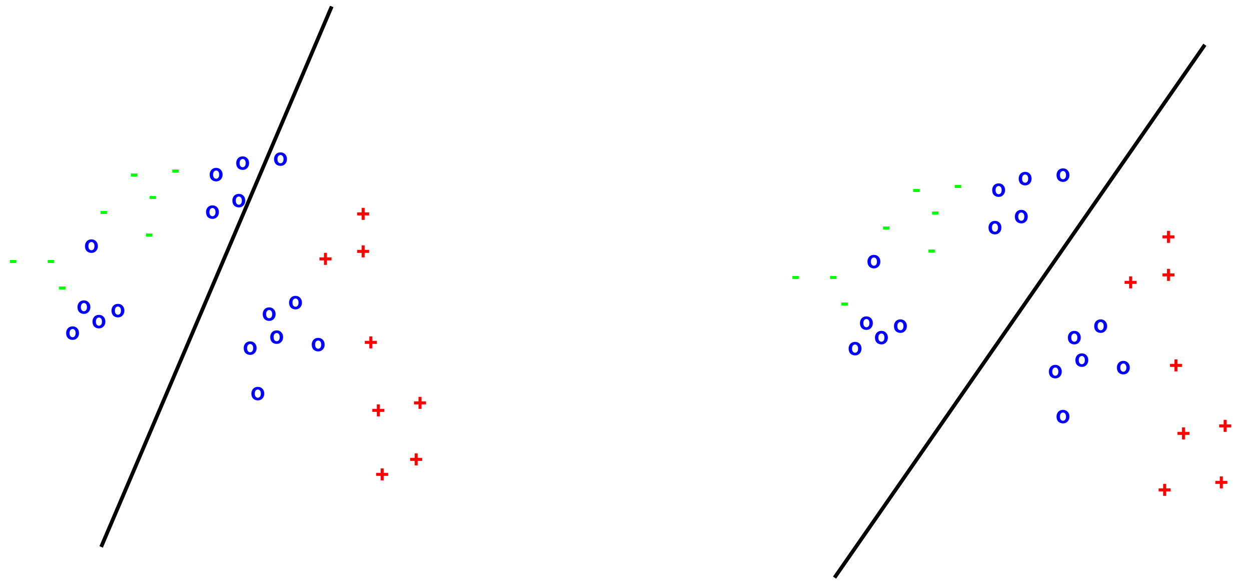
Figure 3.1 Inductive and Transductive Pairwise Classification

With the increasing popularity of semi-supervised learning approaches [13] it is a natural conclusion to adapt well-established supervised learning algorithms to semi-supervised learning. Especially the ability to use unlabeled data, which is one important property of semi-supervised learning algorithms, was a big motivator. Figure 3.1 describes a scenario in which a overall classifier trained on the *semi-supervised approach* might perform better than in the common *supervised approach*.

In the previous chapter I gave some basics about *learning by pairwise comparison* and introduced *support vector machines*, which will be used as base classifier. In this thesis I will compare the performance of overall classifiers using the *learning by pairwise comparison* framework once with common SVMs and once with transductive SVMs as base classifiers. For this task I will give a brief explanation of the theory behind *transductive support vector machines (TSVM)*, then I will explain the implementation of TSVM in *SVMlight*. Finally, I will introduce the new semi-supervised approach called *transductive pairwise classification* and describe the changes applied to the supervised approach.

3.1 Transductive Support Vector Machines

V. Vapnik introduces a new type of inference in addition to inductive inference, the *transductive inference* in [15]. In inductive inference described in the previous chapter, the training data is fully labeled. The goal is then to train a classifier using the given data and evaluate it for other examples to classify them. The key idea of transductive inference is to do the training of a classifier and the evaluation of unlabeled instances in a single step.



3.2 (a) *Inductive SVM*: Only positive and negative examples are considered by the SVM.

3.2 (b) *Transductive SVM*: All the unlabeled examples are considered. It is then attempted to classify them.

Figure 3.2 Inductive and Transductive SVMs

Therefore V. Vapnik describes a support vector method using transductive inference. The resulting SVMs are called *transductive support vector machines*, or just *TSVMs*. Adapting the idea of transductive inference to SVMs, V. Vapnik suggests to additionally consider the test set with k-instances $x_1^*, \dots, x_k^*$ along with the training set $(y_1, x_1), \dots, (y_n, x_n)$ where $y \in \{-1, 1\}$. The goal is then to find a good prediction $y_1^*, \dots, y_k^*$ for the test set with the maximal separating margin for the whole dataset $(y_1, x_1), \dots, (y_n, x_n), (y_1^*, x_1^*), \dots, (y_k^*, x_k*)$.

Similar to the inductive case, this can be formulated the same way as the optimization problem for the linearly separable case:

$$\Phi(w_0(y_1^*, \dots, y_k^*)) = min_{w_0^*} \frac{1}{2} ||w^*||^2 \tag{3.1}$$

$$\begin{aligned} s.t. y_i[(x_i^T w^*) + b] \geq 1, \quad & i = 1, \dots, n \\ y_j^*[(x_i^{*T} w^*) + b] \geq 1, \quad & i = 1, \dots, k \end{aligned} \tag{3.2}$$

and the more general, non-separable case:

$$\Phi(w_0(y_1^*, \dots, y_k^*)) = min_{w_0^*, b, \xi, \xi^*} [\tfrac{1}{2} ||w^*||^2 + C \textstyle\sum_{i=1}^{n} \xi_i + C^* \sum_{j=1}^{k} \xi_j^*] \tag{3.3}$$

$$s.t. y_i[(x_i^T w^*)+b] \geq 1-\xi_i, \quad \xi_i \geq 0, \quad i=1,\dots,n$$
$$y_j^*[(x_i^{*T} w^*)+b] \geq 1-\xi_j^*, \quad \xi_j^* \geq 0, \quad j=1,\dots,k \tag{3.4}$$

Note that w is the normal vector like in the previous chapter. An illustration of the transductive support vector approach is given in Figure 3.2. When trying to use transduction, a new challenge occurs: finding the assignments $(y_1,\dots,y_k)$ with $y_i \in \{-1,1\}$, $i=1,\dots,k$ for $(x_1,\dots,x_k)$ which will generate the best results. For a small set of test examples this problem can be solved by simply trying every combination of assignments [15]. But for larger test sets this approach becomes unusable. Therefore T. Joachims provides a solution which I will explain in the following section.

3.2 Transduction in SVMlight

In this thesis I use *SVMlight* [10], an implementation of support vector machines by T. Joachims. He solves the problem of finding the best assignments for $(y_1^*,\dots,y_k^*)$ by using a local search algorithm [11]. For this the examples x_i^*, x_j^* of the test data are classified with labels y_i^*, y_l^* using the prediction of an inductive SVM trained with the labeled data, generating the outputs $(x_i^*, y_i^*), (x_j^*, y_j^*)$. Then a SVM is trained using those assignments. Afterwards he switches the labeling of two predicted test examples and retrains the classifier with the data $(x_i^*, y_j^*), (x_j^*, y_i^*)$, whereby that one example pair is chosen which minimizes equation **(3.3)**. This is done iteratively until the optimal assignment of $(y_1^*,\dots,y_k^*)$ is found. The algorithm is shown in Figure 3.3.

```
Algorithm TSVM:

  Input:        training examples $((x_1,y_1),\dots,(x_n,y_n))$
                test examples $(x_1^*,\dots,x_k^*)$
  Parameters:   $C, C^*$
                $num_+$
  Output:       $y_1^*,\dots,y_k^*$

$(w,b,\xi,\_) := solve\_svm\_qp([(x_1,y_1)\dots(x_n,y_n)],[],C,0,0);$
$C_-^* := 10^{-5};$
$C_+^* := 10^{-5} \cdot \frac{num_+}{k-num_+};$
$while((C_-^* < C^*)||(C_+^* < C^*))\{$
    $(w,b,\xi,\xi^*) := solve\_svm\_qp([(x_1,y_1)\dots(x_n,y_n)],[(x_1^*,y_1^*)\dots(x_k^*,y_k^*)],C,C_-^*,C_+^*);$
    $while(\exists m,l : (y_m^* * y_l^* < 0)\&(\xi_m^* > 0)\&(\xi_l^* > 0)\&(\xi_m^* + \xi_l^* > 2))\{$
        $y_m^* := -y_m^*;$
        $y_l^* := -y_l^*;$
        $(w,b,\xi,\xi^*) := solve\_svm\_qp([(x_1,y_1)\dots(x_n,y_n)],[(x_1^*,y_1^*)\dots(x_k^*,y_k^*)],C,C_-^*,C_+^*);$
    $\}$
    $C_-^* := min(C_-^* \cdot 2, C^*);$
    $C_+^* := min(C_+^* \cdot 2, C^*);$
$\}$
$return(y_1^*,\dots,y_k^*);$
```

Figure 3.3 The TSVM algorithm

The function *solve_svm_qp* solves the following optimization problem [10]:

$$\Phi(w_0(y_1^*,\dots,y_k^*)) = min_{w_0^*,\xi,xi^*}[\frac{1}{2}||w^*||^2 + C\sum_{i=1}^n \xi_i + C_-^* \sum_{j:y_j^*=-1}^k \xi_j^* + C_+^* \sum_{j:y_j^*=1}^k \xi_j^*] \qquad (3.5)$$

with the constraints **(3.4)**. C_-^* and C_+^* are variables introduced by Morik et. al. [12]. By splitting C^* in two parts for the negatives and positives, C_-^* and C_+^* allow putting more weight on one part or the other. The reason for this is the fact that in text classification there are often more negative than positive examples. So when weighted equally using only one C^*, the negatives will have a bigger influence on the margin than the positives. Since we already have just a few positives this might cause the SVM to overfit on the negative examples, misclassifying the positive examples. So the intention is to allow the SVM a wider margin on the negatives using a small C_-^* and still recognizes the positives quite well.

In the first step of the algorithm w, b, ξ are found using the inductive approach, since with $C_-^* = 0$ and $C_+^* = 0$, **(3.5)** resolves to **(2.16)**. In the next step C_-^* and C_+^* are initialized with some small values. Num_+ is a user-defined value and sets the number of test examples which are initially classified as positives. Those examples are picked by the result of $w^T x + b$. Since $\frac{num_+}{k-num_+} = \frac{1}{\frac{k}{num_+}-1}$ and $0 < num_+ < k$ the fraction will result in a value $x > 0$.

There are several possible cases for the value of the fraction :

- A very simple case is when $num_+ = \frac{k}{2}$. In that case $\frac{num_+}{k-num_+} = 1$ and therefore C_-^* and C_+^* will be equal. So when solving **(3.5)**, the ξ_j^* for the positives and negatives will be weighted the same.
- Another case is when $num_+ < \frac{k}{2}$. In that case $\frac{num_+}{k-num_+} < 1$ and so the initial C_+^* will be smaller and the slack-variables for the positive labeled instances will have less weight.
- In the next case the $num_+ > \frac{k}{2}$. So $\frac{num_+}{k-num_+} > 1$ and the positive slack-variables will be weighted more than the negatives. Also note that in this case the algorithm will terminate faster because C_+^* will reach C^* faster.

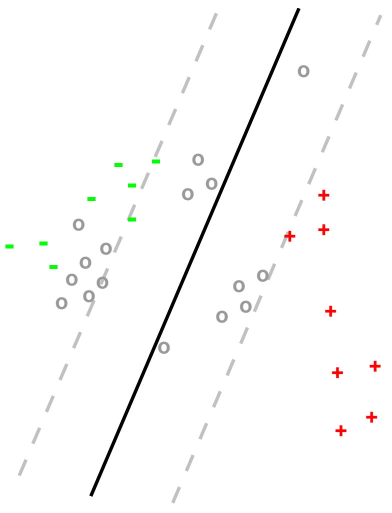
$Num_+ = 0$ and $num_+ = k$ are special cases and internally handled the following way:

- $Num_+ = 0$: The cost-ratio for the positive examples $\frac{num_+}{k-num_+}$ is set to 1. Therefore positive and negative examples are weighted equally. But in the initial assignment none of the examples are assigned to $+1$ and there are no $y_i^* = 1$. Therefore the only positives are the examples already labeled positive.
- $Num_+ = k$: The cost-ratio for the positive examples $\frac{num_+}{k-num_+}$ is set to 0. All unlabeled examples are initially classified as positives, but any slack-variables ξ^* will be ignored, since C_+^* equals zero. Also none of the unlabeled examples will be assigned to -1.

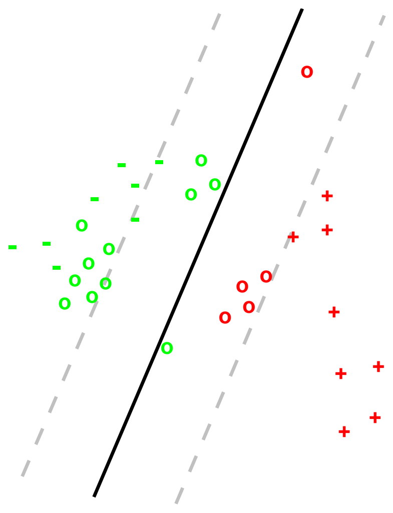
Please note that setting $num_+ = 0$ and $num_+ = k$ will resolve in an approach similar to one-vs-all since the TSVM will try to learn a separating model for exactly all labeled positives or all labeled negatives.

After choosing C, C_-^* and C_+^*, (w, b, ξ, ξ^*) are calculated for the initial assignments of $(y_1^*,\dots,y_k^*)$ (first loop), a positive labeled example is switched with a negative labeled example and new values for (w, b, ξ, ξ^*) are calculated (second loop). The first condition in the second loop ensures that y_m^* and y_l^* have different class labels. The other conditions $(\xi_m^* > 0)$, $(\xi_l^* > 0)$ and $(\xi_m^* + \xi_l^* < 2)$ ensure that the resulting assignment for $y_1^*,\dots,y_k^*$ is the best possible one. This procedure is done with increasing C_-^* and C_+^* in every step until the user-defined value C^* is reached.

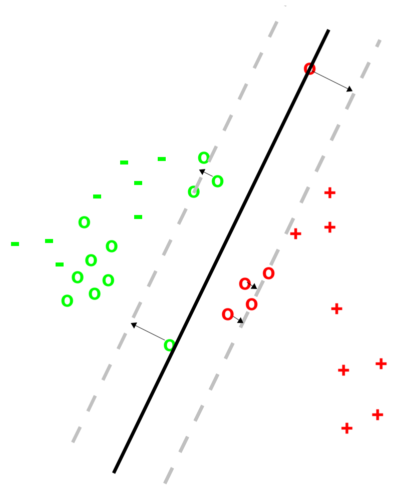
So basically we start with a very big margin due to small C_-^* and C_+^* and calculate the best possible assignment of y_i^*. Afterwards, we reduce the margin and optimize again for some examples which were ignored before due to smaller C_-^*, C_+^*. A proof for the convergence of the algorithm is given in [11].



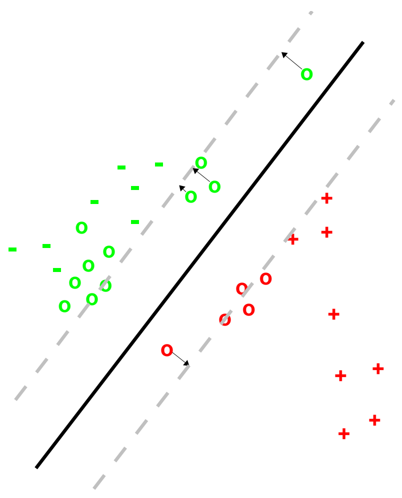
(1) In the first step an inductive SVM is trained for the labeled examples.



(2) The examples that deliver the best results for $w^T x + b$ are assigned to $+1$ (here $num_+ = 5$).



(3) A new inductive SVM is trained with the initial assignments.

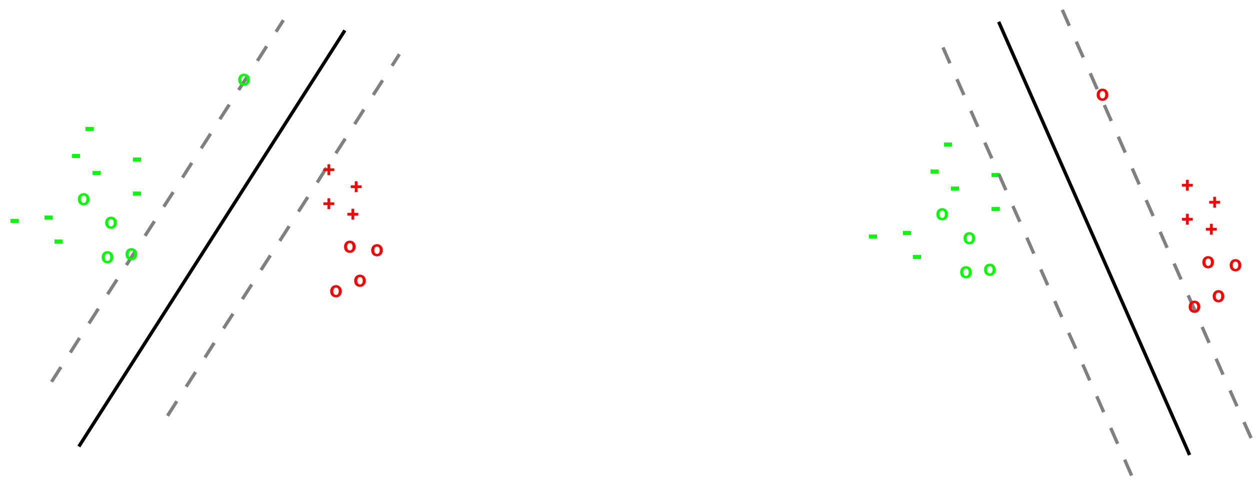


(4) A positive and a negative example are switched, as long as the resulting SVM has a wider margin than before.

Figure 3.4 A graphical example for the TSVM

Figure 3.4 illustrates the first steps of the TSVM algorithm. In the initial phase the inductive SVM is built using only the positive (red colored "+") and negative (green colored "-") examples (1). Num_+ is set to 5 so the unlabeled examples ("O") which are classified as the "most" positive by the SVM are assigned as positives (red colored "O"). All other examples are assigned to the negatives (2). Now the SVM is retrained with the initial assignments of the unlabeled examples (3). Then two examples with different labels are switched and a new SVM is trained, if the resulting margin is bigger than before (4).

One important property of the *TSVMlight* algorithm is that the resulting classifier is not *class-symmetric*. V.Vapnik showed that the inductive SVM always generates the same solution with the best separating margin for a binary classification task $(\mathbf{a}, \mathbf{b})$ regardless of taking all examples of class **a** as positives or the examples of class **b** as positives [15]. Such a classifier is called *class-symmetric* for any binary classification task $(\mathbf{a}, \mathbf{b})$. Since the *TSVMlight* algorithm uses num_+ as the number of instances classified as positives in the initial step, the resulting classifier for a classification task $(\mathbf{a}, \mathbf{b})$ might be different from the classifier for the classification task $(\mathbf{b}, \mathbf{a})$.



(1) TSVM for the task **(+, -)** with $num_+ = 4$.

(2) TSVM for the task **(-, +)** with $num_+ = 4$.

Figure 3.4 Non-symmetry of the *TSVMlight*

As Figure 3.4 shows, the resulting classifier for the task **(+, -)** is different from the classifier for the task **(-, +)**. So this fact has to be considered for the *transductive pairwise classification* approach.

3.3 Transductive Pairwise Classification

Now using TSVMs the question is, if and how much better the overall classifier will perform. While in SVMs good results are strongly depending on optimizing the parameter C [3], in *TSVMlight* there is another parameter p which has big influence on the performance of the classifier. p is the positive ratio for the transduction, i.e. the ratio of the unlabeled examples which will be labeled as positive examples during the training phase. $p \cdot n$ with n being the number of examples is equal to the parameter num_+ described in the previous section.

The *transductive pairwise classification* approach is quite similar to the *learning by pairwise comparison* approach introduced by J.Fürnkranz and E.Hüllermeier [9, 6]. At the beginning a learning problem with k-class labels is divided into $k(k-1)/2$ learning problems with only two class labels each. Then a separating model is built for each of the learning problems. Afterwards the classification of the test examples is done by simple voting like in the supervised approach.
The difference between the supervised approach and the semi-supervised approach is the model building phase for each pair of class labels. In the supervised approach the separating model was built using only the examples of the two class labels [9]. Those examples were used to train a *support vector machine* generating the optimal separating hyperplane for the two classes. Since *transductive support vector machines* are able to build an optimal separating hyperplane with labeled and unlabeled data, I will use all available data as input data for the TSVM. Note that especially all data with other class labels than the two classes which are separated will be used additionally ignoring their class labels and therefore treated as $((y_1^*, x_1^*), \ldots, (y_k^*, x_k^*))$. An illustration of the pairwise classification example using the transductive or

semi-supervised approach is given in Figure 3.6.

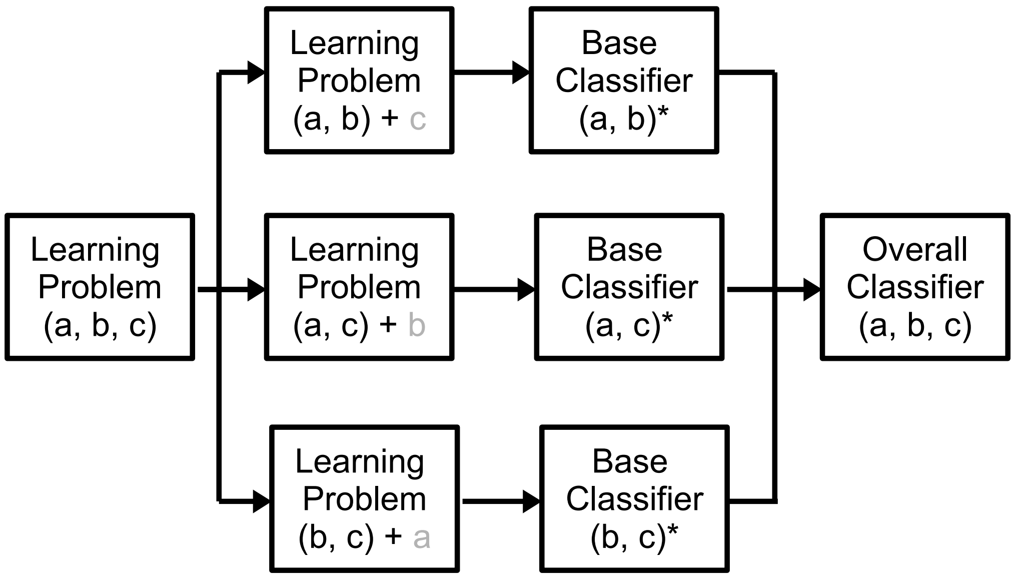

Learning Problem (a, b, c)
Learning Problem (a, b) + c
Learning Problem (a, c) + b
Learning Problem (b, c) + a
Base Classifier (a, b)*
Base Classifier (a, c)*
Base Classifier (b, c)*
Overall Classifier (a, b, c)

Figure 3.6 Transductive Pairwise Classification

As it was shown in the section before, the *TSVMlight* is not class-symmetric. Therefore two separating models will be trained for every binary classification task (a, b), one for (a, b) and one for (b, a). Internally, the decomposition for the TSVM will be done by a *double round robin* decomposition. By doing so, the two separating models might predict different classes for new instances, thereby nullifying the votes of each other, or predict the same class generating a stronger tendency for it. It is also important to realize that the *semi-supervised* approach nullifies the positive effect of the much smaller training sets which resulted by looking at only two classes, since now all available data is used instead of a small training set. So the main question I am trying to solve is if there is a significant improvement of the overall classifier when using transductive support vector machines as base classifier. Depending on the classification task, one might accept a longer training phase for getting better results.

4 Experiments

The goal of the experiments is to compare the performance of the semi-supervised approach with the common supervised approach. The experiments can be divided roughly into three parts. In the first part I will compare the two approaches without any optimization on the SVM. In the second part I will focus on measuring the influence of an additional parameter in the TSVM affecting num_+ described in the previous chapter. In the last part I will optimize the results for both approaches using a cross-validation to find the best input parameters for the SVM and the TSVM and compare them.

4.1 Used Datasets and Set-up

To get proper results, I used several frameworks written for machine learning tasks. Furthermore, I used datasets which are often used to evaluate machine learning algorithms and are widespread for a better comparison of the results with other approaches.

4.1.1 Datasets

The datasets which I am using are some of the multi-class text classification datasets provided by George Forman [4]. A description of the used datasets is given in Figure 4.1 [4].

Dataset	Examples	Attributes	Classes	Dataset	Examples	Attributes	Classes
OHSUMED/oh0	1003	3182	10	TREC/fbis	2463	2000	17
OHSUMED/oh5	918	3012	10	TREC/tr11	414	6429	9
OHSUMED/oh10	1050	3238	10	TREC/tr12	313	5804	8
OHSUMED/oh15	913	3100	10	TREC/tr21	336	7902	6
WebACE/wap	1560	8460	20	TREC/tr23	204	5832	6
TREC/tr31	927	10128	7				

Figure 4.1 Description of benchmark datasets

Since the datasets contain many attributes, each instance is described as a sparse feature vector. Most of these vectors are labeled with the correct class label and used for the training and test phase. An example of such a dataset can be seen in Figure 4.2. As I use weka as an underlying framework, the data files are in a *.arff* format[1].

[1] *ARFF* stands for *Attribute Relative File Format* [7].

```
@RELATION        iris

@ATTRIBUTE       sepallength   REAL
@ATTRIBUTE       sepalwidth    REAL
@ATTRIBUTE       petallength   REAL
@ATTRIBUTE       petalwidth    REAL
@ATTRIBUTE       class         { Iris-setosa,Iris-versicolor,Iris-virginica }

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
...
```

The first part is the header containing information about the data which is going to follow. The actual data begins with the second part (@DATA). Each line is representing a feature vector assigned to a class.

Figure 4.2 An example for an Arff file

4.1.2 Frameworks

To get proper results of the experiments I used several machine learning frameworks which were developed over the time and have proven as stable. First I will give a brief description of the used frameworks and then a scheme of their interaction.

Weka

Weka is a very widespread machine learning environment developed by the *Machine Learning Group at the University of Waikato* [7]. It offers various machine learning algorithms and also a proper data format and learning environment, which is used in *lpcforsos*. For more information about the framework, please read the documentation [7]. The used weka version in this bachelor thesis is a modified version of 3.6.9, which is the latest stable version released.

SVMlight

As base classifier I am using *SVMlight*, an implementation of *support vector machines* written in C by T.Joachims [10]. It already provides various functionalities like the transductive learning mode, which allows the SVM to use transductive inference described earlier in Chapter 3.2. The relevant settings for the *SVMlight* classifier will be named and explained in Section 4.1.4.

LPCforSOS

Learning by Pairwise Comparison for Structured Output Spaces[2] is a machine learning framework developed by the *Knowledge Engineering Group at Technische Universität Darmstadt*. It is an implementation of the learning by pairwise comparison approach described by J.Fürnkranz and E.Hüllermeier [9]. The user can choose between two base classifier environments, the *seco* environment and the *weka* environment. It allows a classification based on pairwise comparison using classifier functions provided by weka. It has a built in cross-validation of the overall classifier to avoid overfitting. The results for each fold are presented in a confusion matrix and a micro-averaged confusion matrix. Further it calculates the performance measures described in Chapter 2.1.3.

[2] http://sourceforge.net/projects/lpcforsos/ , Date of last access: 26th October 2013

Further Frameworks

Since I am using *SVMlight* which is written in C, I had to adapt it into the base learning environment *weka*. *Weka* is written in *Java*, so therefore I am using a *Java Native Interface* adaption of *SVMlight* called *JniSVMlight* provided by Martin Theobald[3].

For the adaption into *weka 3.6.9* I used the wrapper class *JniSVMLight4Weka*. Furthermore, I am using the *CVParameterSelectionMod* class to optimize the input parameters of the support vector machines in the final experiments in Section 4.4. Both, the *JniSVMLight4Weka* wrapper and the *CVParameterSelectionMod* class were written and provided by the *Knowledge Engineering Group of TU Darmstadt*.

4.1.3 Composition of the Frameworks

As, in this thesis, various frameworks were used together, I will give a brief overview of how the individual parts work together.

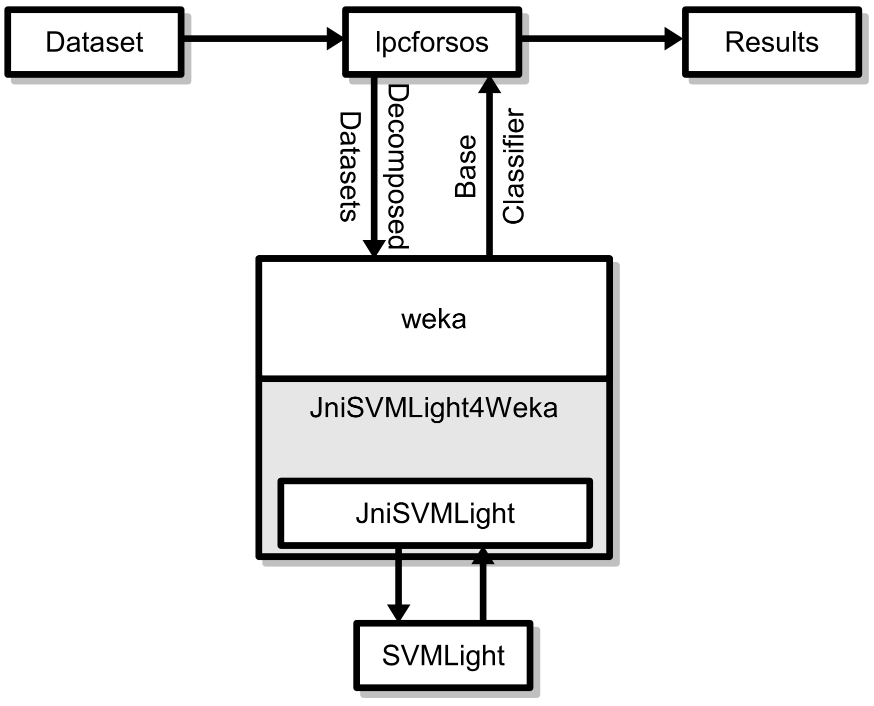


Figure 4.3 Composition of the frameworks

Figure 4.3 gives an illustration of the hierarchy of the single parts. *Lpcforsos* is the overlaying machine learning framework and does the main work. It decomposes the input data and evaluates the performance of the overall classifier by doing a k-fold cross-validation. *Weka* provides the base-learning environment as an underlying framework allowing access to the base classifiers which are used. For the base classifier *SVMlight* is used, with *JniSVMLight4Weka* and *JniSVMlight* for adapting it to *weka*.

For the experiments the input dataset is divided into k-folds in the initial step. Then each fold is further divided in $c(c-1)/2$ datasets with selected instances of a binary classification task. The decomposed datasets are then passed to *weka* which trains the *SVMLight* classifier for the $c(c-1)/2$

[3] http://adrem.ua.ac.be/~tmartin/ , Date of last access: 26th October 2013

datasets. After building the base classifiers, each of the *k*-folds is used as a test set for the remaining classifiers as described in Section 2.1.4. The final prediction for one instance is done by simple majority voting as described in Chapter 2.1. In the last step the predictions for each fold are evaluated and the overall performance is calculated. A scheme of one evaluation run describing the supervised approach is given in Figure 4.4. The changes I made for the semi-supervised approach affect the decomposition step. The decomposed datasets for each fold now also includes all instances without the two class labels of the binary classification problem. Those instances are included unlabeled and passed to the TSVM.

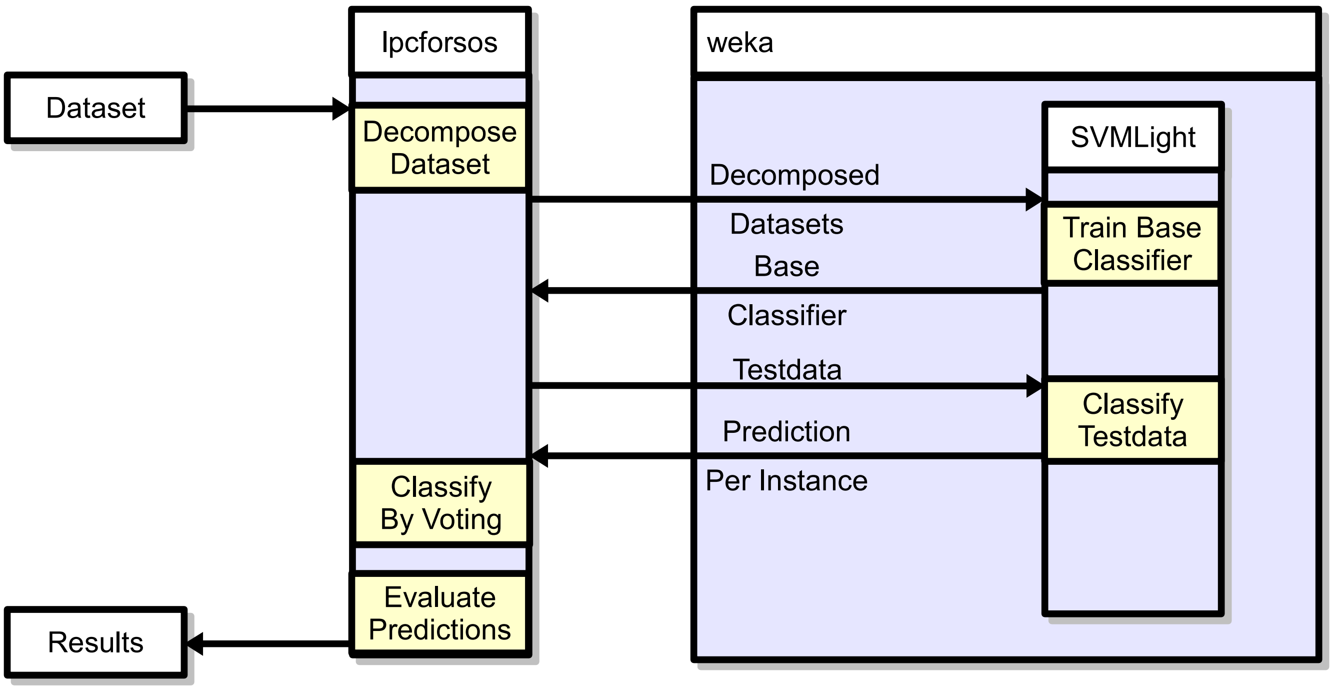


Figure 4.4 Scheme for one evaluation

4.1.4 Set-up

For the evaluation of the new approach I am using the following set-ups for the *Learning by Pairwise Comparison* framework and the *support vector machine*.

LPCforSOS

All relevant input options for the learning by pairwise comparison framework are taken from a xml file. The input options used in this thesis for the first two parts of the experiments are listed in the following *config.xml* file (Figure 4.5):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
  <decompositionType>PairwiseDecomposer</decompositionType>
  <aggregationType>Voting</aggregationType>
  <dataset>datasets/oh0.arff</dataset>
  <numberOfFolds>10</numberOfFolds>
  <seedForEvaluation>2</seedForEvaluation>
  <baseLearnerEnvironment>WekaBaseLearner</baseLearnerEnvironment>
  <classifierName>weka.classifiers.functions.JniSVMLight4Weka</classifierName>
  <classifierOptions>-tm false -c 0.0 -j 1.0 -cost 0.0 -p 0.5 -w 0.1 -e_ab 1.0E-15 -
      e_eq 0.0 -e_di 0.1 -m 40 -maxiter 100 -optprec 0.0 -b true -i false -t 0 -z 1 -d
       3 -g 1.0 -r 0.0 -s 1.0 -n true -ni 0 -v 0</classifierOptions>
  <transduction>false</transduction>
  <isSymmetricClassifier>true</isSymmetricClassifier>
  <decimalPrecision>4</decimalPrecision>
</configuration>
```

Figure 4.5 An example for the config.xml file

As specified in Chapter 3.3, a learning problem with k-classes will be separated into $k(k-1)/2$ binary learning problems. The *PairwiseDecomposer* additionally gives the possibility to choose among several decomposition options to provide the input data for the classifier in a proper format. In this thesis the used classes are *MulticlassDecomposition* and *MulticlassTransDecomposition* which are both classes written especially for decomposing datasets with more than two classes. Furthermore, the *MulticlassTransDecomposition* class provides the instances not related to a binary classification problem unlabeled for the base classifier. To use this class the option *transduction* has to be set *true*. The decomposition is done using the round robin decomposition for multi-class learning problems provided by J. Fürnkranz [5]. Other options will be explained as follows:

- *aggregationType*: The method the overall classifier is using in order to classify the test examples. In this thesis the used option is always simple voting of each base classifier.
- *dataset*: The used dataset for the evaluation.
- *numberOfFolds*: The number of folds which will be used for evaluation. For the experiments in this thesis, the value was set to 10[4].
- *seedForEvaluation*: The seed which is used for the random number generator. For more information see *java.util.Random.class*[5].
- *baseLearnerEnvironment*: The underlying framework for the base learner. *SeCo* and *weka* are possible. The used base learner environment for the experiments is *weka*.
- *classiferName*: The name of the classifier which is used. Since I only use *SVMlight* as a base classifier, it is always set to *JniSVMLight4Weka*.
- *classiferOptions*: The user-defined options, which are relevant for the classifier defined in *classifierName*. The options for *JniSVMLight4Weka* will be explained in the next section.
- *transduction*: An additional check for the options set for the base classifier in *classifierOptions*. It has to be set to the same value as in *classifierOptions*, either *true* or *false*.

[4] The value $k = 10$ is only a convention. But to get proper results, each fold should contain at least 30 instances [3].
[5] http://docs.oracle.com/javase/6/docs/api/java/util/Random.html , Date of last access: 26th October 2013

- *isSymmetricClassifier*: Additional input information about the classifier to simplify the computation by doing a *single round robin* decomposition. Should be set to *true*, if the used classifier is *class-symmetric*, i.e. a classifier M produces the same outputs for the binary set (y_1, y_2) as for the set (y_2, y_1) [5].
- *decimalPrecision*: Sets the decimal precision for the rates and losses. For the experiments, the precision is always set to 4.

The only changing options in the config.xml for each experiment are *dataset* and *transduction*. The options for the base classifiers will either be set by the parameter optimization mod or be constant. For the final experiments there will be some more input parameters introduced in Section 4.4.

JniSVMLight4Weka / SVMlight

There are several options available for the support vector machine. The options explained here are the input options for *JniSVMLight4Weka* which are very similar to the input options for *SVMlight*[6] but are named a little different. The following options are available for *JniSVMLight4Weka* :

- *c*: Sets the c parameter of the *SVMlight* classifer, $c \leq 0$.
- *tm*: train SVM in transduction mode, $tm \in \{true, false\}$
- *j*: A cost-factor by which training errors on positive examples (C_+) outweighs errors on negative examples (C_-) [12]. It allows the introduction of any prior knowledge about the input data to the training phase. It is not used in this thesis and thus always set to 1.0 .
- *cost*: The individual upper bounds for each variable. It is an array used internally and not set.
- *p*: The fraction of unlabeled examples to be classified into the positive class (default is the ratio of positive and negative examples in the training data). This value defines the num_+ value.
- *w*: The epsilon width of tube for regression (default 0.1). *(unused option)*
- *e_ab*: The tolerable error on alphas at bounds, i.e. $0 \leq \alpha \pm e\_ab \leq c$
- *e_eq*: Sets the tolerable error to allow on the equation constraint: $\sum_{i=1}^{n} \alpha_i y_i = e\_eq$, described in Section 2.2.3
- *e_di*: The tolerable error for distances used in stopping criterion $y(w^T x + b) - 1 = e\_di$, cf. Equation **(2.12)** (default 0.001).
- *m*: The size of cache for kernel evaluations in MB (default 40). Since I am using the linear kernel, there is no cache necessary and thus always set to 0 internally.
- *maxiter*: Number of iterations after which the optimizer terminates, if there was no progress in the second loop (maxdiff).
- *optPrec*: The precision of the solver, set to e.g. 1e-21 if convergence problems occur.
- *b*: Usage of biased hyperplane. If true, use $w^T x + b = 0$ and $w^T x = 0$ otherwise. $b \in \{true, false\}$
- *i*: The debug mode. If set to true, classifier outputs additional info to the console. $i \in \{true, false\}$
- *t*: Sets the kernel type of the *SVMlight* classifer. Available are *LINEAR*, *POLYNOMIAL*, *RBF* and *SIGMOID*. $t \in \{0, 1, 2, 3\}$. In this thesis the value is set to 0.

[6] For the input options for the *SVMlight*, see *http://svmlight.joachims.org/ , Date of last access: 26th October 2013*.

- *z*: Sets the type of the SVM. The only available one is classification (value 1).
- *d*: Set the degree of the polynomial. *(unused option)*
- *g*: Set the gamma value of the rbf kernel function. *(unused option)*
- *r*: The c parameter in polynomial and sigmoid kernels. *(unused option)*
- *s*: The s parameter in polynomial and sigmoid kernels. *(unused option)*
- *n*: The instance normalization, $n \in \{true, false\}$. If set to *true* the *Normalize* class will be used to apply the *weka* normalization filter [7] This filter maps to a $[0,1]$ space by default.
- *ni*: Sets the normalization method for single instances. Available are *none*, *L1* and *L2*. These options are chosen by $ni \in \{0,1,2\}$[7]. Instead of using the weka filter, this option will use the instance normalization provided by *JniSVMLight*.
- *v*: Sets the verbosity level, i.e. the level of *SVMlight* debugging infos (default 1).

The options marked as unused are the options which do not affect the support vector machine at all, since the kernel used in the experiments is only the linear one. In the earlier evaluations the parameters which affect error tolerances and weights were set to the default values. But some parameters had to be set to worse values in order to attain an acceptable training time. As an example, *maxiter* was set to only 100 and *e_di* was set to 0.1 instead of the default value (0.001). The important input parameter *c* is optimized with a parameter optimization by cross-validation in the latter experiments. For the normalization of the instances I used only the *Normalize* class (*n* option) provided by weka.

4.2 First Experiments

After integrating *SVMlight* into the *lpcforsos* framework I had to do some modifications to be able to use the transduction mode. One issue was to modify the *lpcforsos* framework to provide all the irrelevant instances of a multi-class dataset unlabeled for training the base classifier. So now, when using transduction, the TSVM is trained on all labeled instances of the binary classification task and every other instance unlabeled. Also the *double round robin* decomposition for multi-class datasets was implemented for the transductive SVM. After integrating *SVMlight* and enabling the transduction mode, I initially ran some tests without optimizing any of the input parameters *c* or num_+ (for transduction) for the support vector machines. The reason was that I wanted to compare the performance of the two approaches directly regardless of the overall performance compared to other classifiers like *J48* or *ZeroR*, which are also available in *weka* [7]. The configuration for the first experiments is illustrated in Figure 4.6 .

[7] *L1* is the linear normalization of the instances and *L2* the L2-normalization

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
  <decompositionType>PairwiseDecomposer</decompositionType>
  <aggregationType>Voting</aggregationType>
  <dataset>datasets/oh0.mc.arff</dataset>
  <numberOfFolds>10</numberOfFolds>
  <seedForEvaluation>2</seedForEvaluation>
  <baseLearnerEnvironment>WekaBaseLearner</baseLearnerEnvironment>
  <classifierName>weka.classifiers.functions.JniSVMLight4Weka</classifierName>
  <classifierOptions>-tm true -c 0.0 -j 1.0 -cost 0.0 -p -1.0 -w 0.1 -e_ab 1.0E-15 -
      e_eq 0.0 -e_di 0.1 -m 40 -maxiter 100 -optprec 0.0 -b true -i false -t 0 -z 1 -d
       3 -g 1.0 -r 0.0 -s 1.0 -n true -ni 0 -v 0</classifierOptions>
  <transduction>true</transduction>
  <isSymmetricClassifier>true</isSymmetricClassifier>
  <decimalPrecision>4</decimalPrecision>
</configuration>
```

Figure 4.6 The configuration for the first experiments. Only the transduction mode and the symmetric classifier options were turned on/off.

One significant difference was the longer training phase for the *transductive support vector machines*, which was expected (cf. Chapter 3.3). In addition, c was set to 0.0 for the evaluations of the SVM and the TSVM. By doing so, the default value of c is used which is defined as $\frac{1}{\|average(x)\|}$, with $\|average(x)\|$ being the average length of all examples. For p the default value was also used which is defined as the ratio of positive and negative examples of the binary class problem [10][8].

The total macro averaged results from the first test runs are summarized in Figure 4.7 . Since the first two experiments ran on an *Intel (R) Pentium (R) Dual CPU T3400 @ 2.16 GHz* with *4.00 GB RAM*, I could not use the datasets with many attributes or many examples. Those were *fbis* and *wap*.

Dataset	Accuracy	Precision	Fp-rate
oh0	77.08%	80.22%	4.27%
oh5	74.49%	79.82%	4.03%
oh10	73.05%	75.41%	4.35%
oh15	70.63%	73.78%	4.49%
tr11	59.63%	57.17%	14.81%
tr12	46.94%	45.23%	21.46%
tr21	71.72%	60.53%	60.07%
tr23	56.31%	59.81%	30.81%
tr31	81.22%	80.84%	7.89%

Performance SVM

Dataset	Accuracy	Precision	Fp-rate
oh0	25.34%	36.80%	4.81%
oh5	41.62%	52.59%	5.03%
oh10	32.57%	47.20%	4.62%
oh15	26.61%	34.97%	5.32%
tr11	9.94%	14.69%	2.63%
tr12	13.43%	12.60%	3.42%
tr21	68.14%	48.69%	64.35%
tr23	59.29%	63.34%	25.87%
tr31	9.80%	31.29%	2.34%

Performance TSVM

Figure 4.7 The results of the first evaluations

The results from the first experiments were surprisingly bad for the transductive support vector approach. While for some classes the *tp-rate* was pretty good, for the most classes the overall classifier was pretty bad. An example for dataset *oh0* is given in Figure 4.8 . It shows the macro averaged performance of the overall classifier for the SVM and the TSVM with p set to default. Also the overall performance of the TSVM was much worse than the performance of the inductive SVM in almost every aspect as the *tp-rates* in Figure 4.7 show. The TSVM also had a worse *precision* compared to the SVM. Only for the dataset *tr23* the transductive approach had a slightly better performance than the inductive approach.

[8] p is the parameter which is used for calculating num_+ in *SVMlight*

Furthermore, the *fp-rate* of the TSVM approach is often similar or smaller than of the SVM approach. Thus the TSVM approach seems to classify much more examples as negatives than the SVM approach.

Class	**Tp-rate**	**Precision**	**Fp-rate**
6-Ketoprostaglandin-F1-alpha	78.95%	84.91%	0.85%
Brain-Chemistry	69.01%	80.33%	1.29%
Creatine-Kinase	60.53%	77.97%	1.40%
England	90.06%	63.92%	11.19%
Ethics	82.613%	92.23%	0.90%
Fundus-Oculi	91.91%	78.62%	3.92%
Heart-Valve-Prosthesis	94.33%	76.89%	6.80%
Larynx	42.42%	96.5%	0.11%
Mexico	29.41%	78.95%	0.42%
Uric-Acid	42.86%	88.89%	0.32%

Performance SVM

Class	**Tp-rate**	**Precision**	**Fp-rate**
6-Ketoprostaglandin-F1-alpha	100.00%	17.54%	28.33%
Brain-Chemistry	47.89%	60.71%	2.363%
Creatine-Kinase	42.11%	27.12%	9.28%
England	0.00%	0.00%	0.00%
Ethics	22.61%	96.30%	0.11%
Fundus-Oculi	5.88%	100.00%	0.00%
Heart-Valve-Prosthesis	5.15%	100.00%	0.00%
Larynx	60.61%	%	3.63%
Mexico	92.162%	12.21%	35.50%
Uric-Acid	0.00%	0.00%	0.00%

Performance TSVM

Figure 4.8 The overall performance for each class of the dataset oh0.

When comparing the overall confusion matrices illustrated in Figure 4.9, it is clearly noticeable that the diagonal of the SVM confusion matrix contains much more values than the diagonal of the TSVM confusion matrix. As explained in Section 2.1.3, the diagonal of a confusion matrix contains the correctly classified instances. So without any optimization on parameter p, the TSVM performs much worse than the SVM. Also it is noticeable that the predictions of the TSVM focus on only two classes (*6-Ketoprostaglandin-F1-alpha*, *Mexico*).

a	b	c	d	e	f	g	h	i	j	class label
45	2	1	4	0	0	4	0	0	1	a = 6-Ketoprostaglandin-F1-alpha
1	49	2	6	1	7	2	1	0	2	b = Brain-Chemistry
3	3	46	9	0	6	9	0	0	0	c = Creatine-Kinase
1	0	4	163	5	4	2	0	2	0	d = England
0	0	0	15	95	0	4	0	1	0	e = Ethics
0	0	0	7	0	125	4	0	0	0	f = Fundus-Oculi
2	1	1	5	1	1	183	0	0	0	g = Heart-Valve-Prosthesis
1	0	0	7	0	7	23	28	0	0	h = Larynx
0	1	0	28	1	4	2	0	15	0	i = Mexico
0	5	5	11	0	5	5	0	1	24	j = Uric-Acid

4.9 (a): The overall confusion matrix of the SVM for dataset oh0.

a	b	c	d	e	f	g	h	i	j	class label
57	0	0	0	0	0	0	0	0	0	a = 6-Ketoprostaglandin-F1-alpha
56	34	1	0	0	0	0	0	1	0	b = Brain-Chemistry
33	0	32	0	0	0	0	1	10	0	c = Creatine-Kinase
56	5	10	0	0	0	0	8	132	0	d = England
9	1	8	0	26	0	0	2	69	0	e = Ethics
43	12	16	0	1	8	0	8	48	0	f = Fundus-Oculi
67	3	44	0	0	0	10	15	55	0	g = Heart-Valve-Prosthesis
13	0	1	0	0	0	0	40	12	0	h = Larynx
3	0	1	0	0	0	0	0	47	0	i = Mexico
39	1	5	0	0	0	0	0	11	0	j = Uric-Acid

4.9 (b): The overall confusion matrix of the TSVM for dataset oh0.

Figure 4.9 The overall confusion matrices for the SVM and the TSVM.

4.3 Second Experiments

After getting such bad results for the transductive approach from the first experiments I decided to vary p (the parameter affecting num_+) and did several test runs with different p parameters. Since the number of unlabeled examples which are classified as positives depend on p, the ratio of positive and negative examples could be a wrong set-up for certain datasets.

E.g. consider a dataset with 100 instances and the class labels $\{a, b, c, d\}$. Assume that a partial order of the classes $a < b < c < d$ exists, but is unknown and there be 10 instances labeled class a, 20 instances labeled class b, 30 instances labeled class c and 40 instances labeled class d. Then a base classifier for the binary classification problem (a, b) with the default value p will try to use 50 percent ($\frac{10}{20} = 0.5$) of the unlabeled instances as positives, which would be $(30 + 40) \cdot 0.5 = 35$ instances. The resulting factor would be $\frac{num_+}{k-num_+} = \frac{35}{65} = \frac{7}{13} \approx 0.54$. Now when calculating the initial C^*_+ and C^*_- it is obvious that $C^*_+ < C^*_-$. Therefore the slack-variables for class b ($\xi_{y_i=-1}$) will have more influence than the $\xi_{y_i=1}$, the slack-variables for class a. The resulting classifier will have a wider margin on class a than on class b. This will cause the classifier to overfit on the negative examples (class b). Also the *TSVMlight* will initially classify half of the unlabeled instances as positives and the other half as negatives. It is reasonable that the resulting classifier will perform bad for the classes c and d.

So I tried various values for p. Again, I did not do any optimization on the c parameter and used the default value as in the first experiments. The results of the second experiments including some results from the first experiments are summarized in Figure 4.10. The configuration was the same as in Figure 4.6 , now with various parameters for $p \in \{0.0, 0.5, 1.0\}$ or *default (d)*. The goal of this experiments was to see how big the influence the p parameter for the results is. So I decided to look at the two special cases with $p = 0.0$ and $p = 1.0$ and the case that positives and negatives are weighted the same ($p = 0.5$). Due to computation time and the fact, that for $p = 0.0$ and $p = 1.0$ the *TSVMlight* results in an approach similar to one-vs-all, I decided to set the option *isSymmetricClassifier* to *true* thus, only applying a round robin decomposition for the subtasks. Furthermore, for $p = 0.5$ the resulting separating models should be quite *class-symmetric*. The difference between the number of instances labeled as positives (num_+) and the number of instances labeled as negatives (num_-) is 1 in the worst case (we have an odd number of unlabeled instances). So for datasets with many examples like in this thesis, the prediction of the classifier should not differ very much for a binary learning problem. For this reason *isSymmetricClassifier* was set to *true* to save computation time by applying only a round robin instead of a double round robin decomposition. The results are visualized in Figure 4.10. For a better comparison the results of the SVM are also listed and indicated by $p = -$.

The results in Figure 4.10 show that the performance is very dependent on how parameter p is chosen. As an example, the overall performance of the TSVM is much better than the performance of the inductive SVM when using 0.0 or 1.0 for p in dataset *oh0*. But with the same parameter $p = 0.0$ the overall classifier performs much worse in the dataset *tr11*. It is hardly possible to find a good overall p parameter for every dataset. I also ran several experiments with various datasets (*oh0*, *tr21*) varying the p parameter from 0.0 to 1.0 in ten steps with a step size of 0.1. Those results are shown in Figure 4.11[9].

Dataset	p	Accuracy	Precision	Fp-rate
oh0	-	77.08%	80.22%	4.27%
oh0	d	25.34%	36.80%	4.81%
oh0	0.5	72.78%	83.00%	2.00%
oh0	0.0	81.06%	84.09%	3.84%
oh0	1.0	81.16%	84.43%	3.70%
oh5	-	74.49%	79.82%	4.03%
oh5	d	41.62%	52.59%	5.03%
oh5	0.5	74.30%	80.57%	2.53%
oh5	0.0	75.26%	81.82%	3.76%
oh5	1.0	76.78%	81.47%	3.55%
oh10	-	73.05%	75.41%	4.35%
oh10	d	32.57%	47.20%	4.62%
oh10	0.5	70.19%	78.08%	2.86%
oh10	0.0	75.52%	76.11%	3.77%
oh10	1.0	75.81%	77.36%	3.94%
oh15	-	70.63%	73.78%	4.49%
oh15	d	26.61%	34.97%	5.32%
oh15	0.5	68.78%	75.67%	3.09%
oh15	0.0	71.95%	77.80%	4.58%
oh15	1.0	74.04%	78.56%	4.13%
tr11	-	59.63%	57.17%	14.81%
tr11	d	9.94%	14.69%	2.63%
tr11	0.5	57.74%	58.22%	13.48%
tr11	0.0	36.00%	69.25%	4.41%
tr11	1.0	58.90%	57.67%	14.35%

Figure 4.10 The results of the second evaluations

P	Accuracy	Precision	Fp-rate
0.1	57.44%	80.68%	3.19%
0.2	53.95%	72.97%	3.84%
0.3	56.94%	70.68%	4.03%
0.4	59.64%	72.43%	3.49%
0.5	72.78%	83.00%	2.00%
0.6	64.21%	74.72%	3.18%
0.7	55.47%	74.20%	3.34%
0.8	58.42%	72.38%	3.50%
0.9	65.71%	76.82%	3.44%

Various p with dataset oh0

P	Accuracy	Precision	Fp-rate
0.1	72.61%	64.22%	57.09%
0.2	72.89%	67.27%	53.93%
0.3	70.53%	59.11%	57.43%
0.4	74.99%	70.34%	50.95%
0.5	74.39%	69.59%	51.60%
0.6	71.98%	65.19%	57.03%
0.7	70.51%	59.47%	62.07%
0.8	68.72%	50.67%	64.88%
0.9	70.21%	54.10%	62.76%

Various p with dataset tr21

Figure 4.11 Performance of the TSVM for the datasets *oh0* and *tr21*

[9] Note that for evaluations with $p \neq 0.5$ the option *isSymmetricClassifier* was set to **false** for the reasons explained in Section 3.3.

As can be seen in Figure 4.11, the *precision* is quite high in all the evaluations for dataset oh0, being over 60% for the most p parameter, so the overall classifier was quite accurate for the positives among the predicted positives. But the *tp-rate* is pretty bad for the most values of p. Only for the cases of 1.0, 0.0 and 0.5 the classifier did perform quite well. With 1.0 and 0.0 being the extreme cases, I chose to use 0.5 as a default p value for the final experiments. For dataset *tr21* $p = 0.5$ seems also to be a quite good choice when looking at the results.

4.4 Final Experiments

The goal of the final experiments was to generate the best results for both approaches with an acceptable training time. Therefore the c parameter of every base classifier should also be optimized. I wanted to generate evaluations for the following settings:

- The inductive approach with a parameter selection using cross-validation on parameter c.
- The transductive approach with the default p of 0.5 and a parameter selection using cross-validation on parameter c.

4.4.1 Modifications

To find the ideal c parameter, I used the *CVParameterSelectionMod* class developed by the *Knowledge Engineering Group* at *Technische Universität Darmstadt*. This class allows to pick some input options for a chosen classifier and to specify a set of values for each option which is optimized. The set of tested values is chosen by a start-value and stop-value and by the number and type of steps from the first to second. In this thesis I am using exponential steps for the selected parameter. For finding the best c, the optimizer class tests every parameter and evaluates the classifier trained on the given parameter using a k-fold cross-validation and then picks the parameter generating the best results.

For the final experiments I tried to find the optimal c parameter by doing a 3-fold cross-validation. The values which I tried for c were from 2^{-5} to 2^5 in ten exponential steps, i.e. $\{2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4, 2^5\}$. Using the *CVParameterSelectionMod* class should generate better results due to a good c parameter for the SVM and TSVM, but it also needs a lot more training time. Since every possible parameter is evaluated, for the base classifier with optimizing c, $11 \cdot 3 = 33$ (number of tested parameters times the number of folds) classifiers have to be built instead of one.
To turn on the *CVParameterSelectionMod*, an additional configuration option has been added to the config.xml file. The option *cvMod* is now available and to be set for better results.

For the final experiments another option was added in the configuration file, the *sparseStorage* option, which allows *lpcforsos* to store the instances in a sparse data format. Instead of saving every value of an instance in a single array, the instances are now split and stored in two arrays. For datasets with many attributes but only a few values, the sparse storage saves a lot of memory space since the zeros are saved implicitly. An example for sparse storage is illustrated in Figure 4.12[10].

1	0	0	0	0	3	0	0	0	2	1	0	0	1	0

Normal Storage: The instance values are stored in one array at their actual position.

[10] Please note that the sparse storage might result in a longer training and testing time, caused by an additional array access. So if the memory size is not an issue, one might choose to use the normal storage.

0	5	9	10	13
1	3	2	1	1

Sparse Storage: Instead of using only one array, an index and a value array are used. The index array holds the position of the entries and the value array their actual value.

Figure 4.12 Sparse Data Storage

The additional options in the config.xml for the final experiments are:

- *cvMod*: A boolean value for turning on the parameter optimization. When set to *true* the *c* parameter will be optimized.
- *sparseStorage*: An option for using the sparse storage. The instances will be stored sparse, when this option is set *true*.

Some options for the SVM and the TSVM were also modified and will be explained as follows:

- *e_di* was now set to the default value to improve the performance of the overall classifier.
- As a conclusion from the results of the previous experiments *p* in Section 4.3 was always set to 0.5.

4.4.2 Set-up

After the modifications made in the previous section, the final configuration was as follows (Figure 4.13):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
  <decompositionType>PairwiseDecomposer</decompositionType>
  <aggregationType>Voting</aggregationType>
  <dataset>datasets/oh0.mc.arff</dataset>
  <numberOfFolds>10</numberOfFolds>
  <seedForEvaluation>2</seedForEvaluation>
  <baseLearnerEnvironment>WekaBaseLearner</baseLearnerEnvironment>
  <classifierName>weka.classifiers.functions.JniSVMLight4Weka</classifierName>
  <classifierOptions>-tm false -c 0.0 -j 1.0 -cost 0.0 -p 0.5 -w 0.1 -e_ab 1.0E-15 -
      e_eq 0.0 -e_di 0.001 -m 10 -maxiter 100 -optprec 0.0 -b true -i false -t 0 -z 1
     -d 3 -g 1.0 -r 0.0 -s 1.0 -n true -ni 0 -v 0</classifierOptions>
  <transduction>false</transduction>
  <cvMod>true</cvMod>
  <isSymmetricClassifier>true</isSymmetricClassifier>
  <decimalPrecision>4</decimalPrecision>
  <sparseStorage>true</sparseStorage>
</configuration>
```

Figure 4.13 The *config.xml* for the final experiments.

Since $p = 0.5$ I decided to set the option *isSymmetricClassifier* to *true* again, for less memory consumption and a shorter computation time.

The final experiments ran on a cluster provided by the *Knowledge Engineering Group* of *Technische Universität Darmstadt*. The technical details are given online[11]: "*It consists of 12 nodes with two AMD Opteron(tm) Processor 250 each. One half of the nodes offer 8 GB RAM (6 x 8 GB = 48 GB RAM) while the rest houses 4 GB RAM (4 x 6 GB = 24 GB RAM), providing 72 GB RAM in total.*"

[11] http://www.ke.tu-darmstadt.de/resources/ComputingCluster , Date of last access: 26th October 2013

4.4.3 Results

The results for the eleven evaluated datasets were not really different from the first evaluations. While in some cases the performance of the TSVM was pretty good compared to the SVM, most results were significantly worse.

Dataset	Accuracy	Precision	Fp-rate
oh0	79.06%	81.10%	3.63%
oh5	74.50%	78.64%	3.75%
oh10	73.62%	75.45%	4.12%
oh15	71.73%	74.98%	4.00%
tr11	75.59%	74.55%	5.91%
tr12	69.94%	73.95%	9.32%
tr21	78.57%	72.22%	37.07%
tr23	70.62%	70.02%	15.50%
tr31	81.23%	80.61%	7.84%
fbis	68.58%	67.44%	5.30%
wap	78.27%	79.07%	2.68%

Performance SVM

Dataset	Accuracy	Precision	Fp-rate
oh0	73.09%	80.95%	2.27%
oh5	75.06%	80.86%	2.55%
oh10	72.76%	76.63%	2.98%
oh15	70.31%	77.93%	5.75%
tr11	59.91%	73.17%	6.83%
tr12	61.96%	66.51%	9.82%
tr21	76.77%	73.07%	36.40%
tr23	68.62%	66.53%	14.54%
tr31	76.58%	80.81%	8.18%
fbis	53.67%	74.46%	2.21%
wap	56.15%	72.69%	1.21%

Performance TSVM

Figure 4.14 Comparison of the performance in the final experiments

As can be seen in Figure 4.14 the *tp-rate* of the overall classifier using the SVM is between 0.7 and 0.8. Also Figure 4.14 shows that the *precision* is in quite the same level like the *tp-rate*. The overall classifier using the TSVM is doing good for one half of the datasets (*oh5*, *oh10*, *oh15*, *tr21*, *tr23*) but worse than the SVM for the other half of the datasets. Furthermore, it is noticeable that the TSVM did not do better in any case and only equally good for the dataset *oh5*. In every other evaluation the performance was worse. In addition, the training time of the TSVM approach was at least ten times longer than the training time of the SVM approach.

5 Conclusion and Outlook

The results of the experiments have shown that the pairwise classification using inductive support vector machines is more practicable and generating better results than the approach using transductive support vector machines. Especially when using large datasets, the transductive support vector machines require a lot more training time. In addition, the memory consumption is significantly higher when using TSVMs. The original benefit of the pairwise classification approach was the smaller datasets for the training phase of each separating model. With using TSVMs all the examples could be used for the training phase which generated extremely large decomposed datasets. The training time, as well, increased drastically since the TSVM needed a lot more time than the SVM. So the one benefit against one-vs-all learning with more complex but less separating models was nullified.

In Summary, a pairwise classification approach using *TSVMlight* is not practical on this basic level using a linear kernel and a simple voting strategy. For further research, support vector machines should be made more efficient. Instead of the linear kernel, a *polynomial* or an *rbf* kernel could be used. And when using these more complex kernels, one should consider extensive testing and evaluation of the optimal input parameters for those kernels. Another possibility would be to use a better voting strategy. Since we do not have equally large amounts of examples for every class, the vote of a separating model trained on fewer labeled examples should have less weight for the overall prediction. One idea could be to use the fraction of the number labeled example to the number of unlabeled examples used for building the base classifier.

The two suggested improvements described above affect the supervised and the semi-supervised approach. An improvement for the semi-supervised approach could also be the optimization of the p parameter, which I set to 0.5. The reason for my choice of $p = 0.5$ was the strong dependency of the *TSVMlight* performance on the p parameter which was explained in Section 3.2 and shown in the experiments. A *TSVMlight* approach with using different p parameters for each separating base model might perform much better than the transductive approach using a fixed p. However one should keep in mind, that for $p \neq 0.5$ the resulting *TSVMlight* is not class-symmetric. As a consequence, a double round robin decomposition should be applied for generating the subsets for the base classifier, which will result in a higher memory consumption and a longer training time.

The p parameter strongly affects the performance of the transductive support vector machine and since the number of unlabeled examples which are labeled as positives does not change during the *TSVMlight* algorithm, one could try to modify the algorithm of finding the best assignments for y_i^* itself. Instead of letting the user choose the number of the examples to label as positives, the initial assignment of the unlabeled examples could be done by a *k-nearest-neighbor* algorithm using the labeled examples. Afterwards the algorithm of *TSVMlight* could be applied to optimize the result on the initial y_i^*.

Some other research could be done in adapting the semi-supervised approach to other tasks like multi-label classification. In multi-label classification, the examples have several class labels instead of one like in the multi-class task. It may be possible that the TSVM will handle those examples better than the SVM due to some meta information embedded in the data structure.

Since the performance of the overall classifier using TSVMs was also strongly depending on the used dataset, one might try to find out for which datasets this approach will perform well. Maybe this approach might perform better for smaller datasets or datasets with a certain structure. So with a little prior knowledge, the *semi-supervised pairwise classification* approach might be a better alternative to the ordinary *supervised pairwise classification*. Note that with smaller datasets the overall training time will not differ very much for both approaches.

Bibliography

[1] K. Brinker and E. Hüllermeier. Case-based label ranking. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 566–573. Springer Berlin Heidelberg, 2006.

[2] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[3] P. Flach. *Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[4] G. Forman and I. Cohen. Learning from little: Comparison of classifiers given little training. In *PKDD*, pages 161–172, 2004.

[5] J. Fürnkranz. Round robin classification. *J. Mach. Learn. Res.*, 2:721–747, Mar. 2002.

[6] J. Fürnkranz and E. Hüllermeier. Preference learning: An introduction. In J. Fürnkranz and E. Hüllermeier, editors, *Preference Learning*, pages 1–17. Springer Berlin Heidelberg, 2011.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[8] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[9] E. Hüllermeier and J. Fürnkranz. Learning from label preferences. In *Proceedings of the 14th international conference on Discovery science*, DS'11, pages 2–17, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] T. Joachims. Advances in kernel methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.

[11] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[12] K. Morik, P. Brockhausen, and T. Joachims. Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 268–277, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[13] N. Pise and P. Kulkarni. A survey of semi-supervised learning methods. In *Computational Intelligence and Security, 2008. CIS '08. International Conference on*, volume 2, pages 30–34, 2008.

[14] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[15] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[16] F. Yaman, T. J. Walsh, M. L. Littman, and M. desJardins. Democratic approximation of lexicographic preference models. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1200–1207, New York, NY, USA, 2008. ACM.