

# Masterarbeit

vorgelegt an der  
Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt in  
der Fakultät Informatik und Wirtschaftsinformatik zum Abschluss  
eines Studiums im Studiengang Informationssysteme

## Integration of transfer learning into the Probabilistic Classification Vector Machine

**Erstprüfer:** Prof. Dr. rer. nat. habil. Frank-Michael Schleif

**Zweitprüfer:** Prof. Dr.-Ing. Frank Deinzer

**Abgabetermin:** 12.10.2017

**Eingereicht von:** Christoph Raab, Würzburg

## **Eidesstattliche Erklärung**

Hiermit versichere ich, dass ich die vorgelegte Masterarbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Würzburg, den 12.10.2017

## Kurzfassung

In der Mustererkennung gilt die Klassifizierung, basierend auf überwachtem Lernen, als ein wichtiges Paradigma und findet Anwendungen in vielen Bereichen. Auf Grund von bereits kategorisierten Trainingsdaten, können Algorithmen, die auf überwachtem Lernen basieren, ein Modell konstruieren. Mithilfe dieses Modells können Vorhersagen erstellt werden. Die Qualität der Vorhersage ist jedoch schlecht, wenn die Trainingsdaten in diesem Zielbereich nicht ausreichen oder aus einem anderen Bereich entstanden sind und daher verschieden sind. Dies ist üblich in realen Anwendungsfällen. Die Probabilistic Classification Vector Machine ist zwar in der Lage viele Probleme der momentanen Standardlösung zu beheben, kann jedoch die Unterschiede in den Bereichen nicht ausgleichen. Um das Problem des Wissenstransfers zu beheben, wurde die Probabilistic Classification Vector Machine durch das Transfer-Lernen erweitert. Dies basiert auf dem Transfer-Kernel-Ansatz. Außerdem wurden das daraus entstehende Problem der aufwändigen Parametersuche durch einen heuristischen Ansatz ersetzt. Die Leistungsunterschiede der Algorithmen wurden gegenübergestellt und zudem mit momentanen Transfer-Learning-Lösungen verglichen. Die Tests wurden auf verschiedenen, für die Wissenschaft üblichen, Datensätzen durchgeführt. Die Unterschiede wurden mit Hilfe der Kreuzvalidierung und dem Friedman Test festgestellt.

## Abstract

One prominent paradigm in the field of pattern recognition which has many applications in a broad range of disciplines is classification based on supervised learning. Algorithms, which are based on supervised learning create their model based on pre-classified training data. This model can predict future events and can classify new information. However, a problem that classification suffers from is that the domain of interest, where the prediction takes place, has no sufficient training data. Furthermore, if the only available training data is drawn from another domain, hence different, then it will result in a critical performance drop. Domain differences are quite common in real-world scenarios. The probabilistic classification vector machine is able to ease some critical issues from the current state of the art solution but suffers from the transfer problem. To tackle this issue, we will extend the probabilistic classification vector machine with transfer learning through a transfer kernel approach and solve the resulting problem of parameter determination through a heuristic estimation. We have determined the performance differences of these two algorithms and compared it to other transfer learning approaches based on well-known datasets and a real-world dataset. The difference is determined via cross-validation and the Friedman test.

# Contents

<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Algorithms</b>	<b>ix</b>
<b>List of Abbreviations</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Vector Machines</b>	<b>3</b>
2.1 Support Vector Machine . . . . .	3
2.2 Disadvantages of Support Vector Machines . . . . .	4
2.3 Relevance Vector Machine . . . . .	6
2.4 Probabilistic Classification Vector Machine . . . . .	8
2.4.1 Model Specification . . . . .	8
2.4.2 Prior over Weights . . . . .	9
2.4.3 Expectation Maximization Algorithm . . . . .	10
2.4.4 Properties of Algorithm . . . . .	13
2.5 Conclusion . . . . .	13
<b>3 Transfer Learning</b>	<b>15</b>
3.1 Challenges of Supervised Learning Algorithms . . . . .	15
3.2 Definition of Transfer Learning . . . . .	17
3.3 Measurement of Difference . . . . .	19
3.3.1 Maximum Mean Discrepancy . . . . .	20
3.3.2 Kullback-Leibler Divergence . . . . .	21
3.4 Settings of Transfer Learning . . . . .	21
3.4.1 Inductive-Transfer Learning . . . . .	22
3.4.2 Transductive-Transfer Learning . . . . .	23
3.4.3 Unsupervised-Transfer Learning . . . . .	24
3.5 Homogeneous Transfer Learning . . . . .	24
3.5.1 Instance-Transfer . . . . .	25

3.5.2	Symmetric-Feature-Transfer . . . . .	26
3.5.3	Asymmetric-Feature-Transfer . . . . .	29
3.5.4	Parameter-Transfer . . . . .	31
3.5.5	Relational-Knowledge-Transfer . . . . .	33
3.6	Heterogeneous Transfer Learning . . . . .	34
3.7	Negative Transfer . . . . .	36
3.8	Conclusion . . . . .	37
<b>4</b>	<b>Integration of Transfer Learning</b>	<b>38</b>
4.1	Nyström Approximation . . . . .	39
4.1.1	Nyström Kernel Learning . . . . .	40
4.2	Domain Invariant Kernel Learning . . . . .	41
4.2.1	Learning Approach . . . . .	42
4.2.2	Learning Algorithm . . . . .	44
4.3	PCTKVM Algorithm . . . . .	46
4.3.1	Theta Estimation . . . . .	47
4.3.2	Training Algorithm . . . . .	49
4.3.3	Prediction . . . . .	50
4.3.4	Extensions . . . . .	51
4.4	Conclusion . . . . .	52
<b>5</b>	<b>Benchmark Study</b>	<b>53</b>
5.1	Dataset Description . . . . .	54
5.1.1	Text Dataset . . . . .	54
5.1.2	Image Dataset . . . . .	55
5.2	Study Procedure and Settings . . . . .	57
5.2.1	Data Sampling . . . . .	58
5.2.2	Kernel . . . . .	58
5.2.3	Parameters . . . . .	59
5.3	Performance Metrics . . . . .	60
5.3.1	Area under a ROC Curve . . . . .	61
5.3.2	Accuracy and Error . . . . .	61
5.3.3	Root Mean Squared Error . . . . .	62
5.4	Descriptive Statistics . . . . .	62
5.4.1	Metric Results . . . . .	62
5.4.2	Computational Complexities . . . . .	65
5.4.3	Number of Support Vectors . . . . .	66
5.5	Comparison over one Dataset . . . . .	67
5.5.1	Result and Discussion . . . . .	68
5.6	Comparison over multiple Datasets . . . . .	69

---

5.6.1	Analysis of Variance . . . . .	70
5.6.2	Friedman Test . . . . .	70
5.6.3	Result and Discussion . . . . .	71
5.7	Conclusion . . . . .	73
<b>6</b>	<b>Evaluation and Future Work</b>	<b>74</b>
6.1	Extensions . . . . .	76
<b>7</b>	<b>Conclusion</b>	<b>78</b>
<b>A</b>	<b>Mathematical Notation</b>	<b>80</b>
A.0.1	Terminology . . . . .	81
<b>B</b>	<b>Complete Test Results</b>	<b>83</b>
B.1	Five-Two Comparison . . . . .	83
B.1.1	Time Comparison . . . . .	88
B.2	Complete-Dataset Comparison . . . . .	90
<b>C</b>	<b>Details of Algorithms</b>	<b>95</b>

# List of Figures

2.1	Example of SVM Classification . . . . .	4
2.2	Comparison of SVM and RVM Classification . . . . .	6
2.3	Truncated Gaussian Priors over Weights . . . . .	10
3.1	Comparison of Transfer and Traditional Learning Problems . . . . .	18
3.2	Settings of Transfer Learning . . . . .	22
3.3	The Process of Geodesic Flow Kernel Approach . . . . .	29
4.1	Transfer Kernel Learning Process . . . . .	42
4.2	Scatter Plot of Gaussian- and TKL-Kernel . . . . .	45
4.3	Performance in Dependence of Theta . . . . .	47
4.4	Effect of Width in Gaussian Kernel for Regression . . . . .	48
5.1	Plot of Orgs vs. People Dataset . . . . .	55
5.2	Examples from Image Dataset . . . . .	57
5.3	Plot of mean Error and standard Deviation . . . . .	64
B.1	Plot of Mean Error and Standard Deviation on Image Dataset (CV) .	85
B.2	Plot of Mean Error and Standard Deviation on Reuters Dataset (CV)	85
B.3	Plot of Mean Error and Standard Deviation on the whole Datasets .	91



# List of Tables

5.1	Overview of Key Values of Reuters-21578 dataset . . . . .	55
5.2	Overview of Key Values of Image dataset . . . . .	56
5.3	List of Estimated Thetas . . . . .	60
5.4	Result of Cross-Validation . . . . .	63
5.5	Comparison of Computational Complexities . . . . .	66
5.6	Comparison of Number of Support Vectors . . . . .	67
5.7	Result of the 5 x 2 cv F-Test . . . . .	69
5.8	Mean Ranks under the Performance Metrics . . . . .	72
5.9	Result of Friedman Test with Bonferroni-Dun as Post-Hoc Test . . .	72
A.1	Notation of Transfer Learning . . . . .	81
B.1	Win-Lose-Tie Result of the 5 x 2 cv F Test . . . . .	84
B.2	Complete Cross-Validtion Result of Error . . . . .	84
B.3	Complete Cross-Validtion Result of AUC . . . . .	86
B.4	Number of used Support Vectors from Cross-Validation . . . . .	87
B.5	Comparison of Mean Calculation Time . . . . .	88
B.6	Time Comparison from Cross-Validation Test . . . . .	89
B.7	Complete Dataset Result of Error . . . . .	90
B.8	Complete Dataset Result of AUC . . . . .	92
B.9	Complete Nnumber of used Support Vectors . . . . .	93
B.10	Time Comparison on Whole Dataset . . . . .	94

# List of Algorithms

1	Transfer Kernel Learning . . . . .	44
2	Probabilistic Classification Transfer Kernel Vector Machine (Short) .	50
3	Probabilistic Classification Vector Machine . . . . .	96
4	Probabilistic Classification Transfer Kernel Vector Machine . . . . .	97

# List of Abbreviations

<b>ACC</b>	Accuracy.....	63
<b>ANOVA</b>	Analysis of Variance.....	70
<b>AUC</b>	Area under a ROC Curve.....	61
<b>CDF</b>	Cumulative Distribution Function.....	8
<b>CGI</b>	Carnegie Group, Inc.....	54
<b>CP-MDA</b>	Conditional Probability based Multisource Domain Adaptation.....	76
<b>DA</b>	Domain Adaptation.....	19
<b>DF</b>	Document Frequency.....	54
<b>EM</b>	Expectation-Maximization.....	8
<b>ERR</b>	Error.....	62
<b>GFK</b>	Geodesic Flow Kernel.....	28
<b>IID</b>	independent and identically distributed.....	20
<b>ITL</b>	Inductive Transfer Learning.....	22
<b>JDA</b>	Joint Domain Adaption.....	30
<b>KLD</b>	Kullback-Leibler Divergence.....	19
<b>LOO</b>	Leave One Out.....	33
<b>LS-SVM</b>	Least-Square Support Vector Machine.....	32
<b>MAE</b>	Mean Absolute Error.....	62
<b>MAP</b>	Maximum a Posteriori.....	8
<b>MMD</b>	Maximum Mean Discrepancy.....	19
<b>MMDE</b>	Maximum Mean Discrepancy Embedding.....	28
<b>MMKT</b>	Multi-Model Knowledge Transfer.....	32
<b>PCA</b>	Principal Component Analysis	
<b>PCTKVM</b>	Probabilistic Classification Transfer Kernel Vector Machine.....	38
<b>PCVM</b>	Probabilistic Classification Vector Machine.....	3

<b>PSD</b>	Positive Semi-definite Kernel .....	20
<b>QP</b>	Quadratic Program .....	44
<b>RBF</b>	Radial Basis function .....	58
<b>RKHS</b>	Reproducing Kernel Hilbert Space .....	20
<b>RMSE</b>	Root Mean Square Error .....	62
<b>ROC</b>	Receiver Operating Characteristics .....	61
<b>RT-PCVM</b>	Rotational Transfer-Probabilistic Classification Vector Machine .....	76
<b>RVM</b>	Relevance Vector Machine .....	3
<b>SDP</b>	Semi-Definite Program .....	28
<b>SIFT</b>	Scale Invariant Feature Transform .....	56
<b>SURF</b>	Speeded Up Robust Features .....	57
<b>SVD</b>	Single Value Decomposition .....	41
<b>SVM</b>	Support Vector Machine .....	3
<b>TCA</b>	Transfer Component Analysis .....	27
<b>TFIDF</b>	Term-Frequency Inverse-Document-Frequency .....	54
<b>TKL</b>	Transfer Kernel Learning .....	41
<b>TTI</b>	Text to Image .....	34
<b>TTL</b>	Transductive Transfer Learning .....	23

# Chapter 1

## Introduction

Machine learning insights and the resulting technologies have a broad range of applications and become more important in many disciplines. They are used to identify handwritten numbers from an image or predict whether a hospitalized patient will get a heart attack or not. They are playing a major part in the engineering of autonomous cars or can predict future changes in the stock market.[63, p. 1][84, p. 1] Besides these extreme examples, machine learning can play a role in everyday life. It is used to characterize a new customer on a website, to identify his needs and his wishes regarding a certain product on this site. Another use case is to create an individual advertisement for a customer, based on his behavior to optimize the shopping experience. The World Wide Web is a giant information collective and beyond that changes and proliferates. In research, machine learning is used to classify or cluster this massive amount of information. For example the categorization of new upcoming websites into trustworthy or shady, based on the already analyzed pool of websites, which finally results in spam protection software.[66][76]

As the previous implies, the classification of information is one task in machine learning. Information is sometimes called pattern concerning the task pattern recognition. This pattern can be created by feature extraction based on pre-processed images, speech signals or text-documents. Furthermore, it is possible to create 'theories' based on the previously collected patterns. Concerning pattern recognition, this theory about the data is called model. This model represents the outcome of an algorithm to describe the underlying data. Based on this model, predictions can be made for new incoming patterns.[80][63, p. 2-5]

The question, which algorithm provides the best model can be a challenging question. In 2007, the ten most influential algorithms in various research communities were identified. For example,  $k$ -Means, Support Vector Machines, A priori, Expectation-Maximization, AdaBoost and Naive Bayes. However, these algorithms can not ad-hoc be used for any type of problem concerning the data. These algorithms have prerequisites and a scope of application.[91]

An algorithm, from the category supervised learning, must first learn from previously classified data called source or training data. Based on this training data, it can create a model and finally can predict future events, based on new information, which is called target or test data. Although one would say that this constraint is limiting, supervised learning algorithms are doing a pretty decent job.[88, p. 1][80, p. 7]

From the examples above, the Support Vector Machine is one of these supervised learning algorithms. However, besides the already good performance, there is some research interest concerning some disadvantages of the Support Vector Machine. To tackle these drawbacks, the Probabilistic Classification Vector Machine was proposed. This algorithm also uses supervised learning for creating the regarding model.[19] In the course of this work, we will dive deeper into the techniques and algorithms of these vector machines and discuss improvements concerning the support vector machine.

However, most of the supervised classification algorithms suffer from a certain problem. The problem that training data and test data are different from each other, which is often the case in the previously discussed real-world scenarios. These differences can be expressed in two domains, training and testing. Because of this, the task of transferring knowledge from one domain into the another, has attracted some research interest.[66]

The idea of transfer knowledge results in the task transfer learning. This is one of the fundamental achievements of life on earth. The ability to transfer can be found in many species in life, even while observing insects. In fact, humankind (primates) masters this task before all other species.[12, p. 198-200]

The transfer learning task, which is one reason for many fundamental insights in our world, is currently applied to pattern recognition algorithms.

Of course, the task of transfer learning which is done by a human differs greatly from the transfer learning solution of an algorithm, but the goal is similar: Collect knowledge in one domain and transfer it to another different but related domain. Regarding supervised classifier, it should end up with an improved performance of the algorithm.[66] The technique transfer learning in the pattern recognition process can be interpreted as an extension of feature generation techniques and therefore be considered as pre-processing step. For example, some transfer learning solutions are just extending feature generation methods, like dimensionality reduction.

Transfer learning has been already established in the research community with over 700 published academic articles until 2016.[88, p. 2] Therefore, we will discuss the current state of transfer learning research and explain some ideas about it. Furthermore, we will see how transfer learning can be integrated into the Probabilistic Classification Vector Machine and how it influences the performance.

# Chapter 2

## Vector Machines

In this chapter, we will introduce supervised vector machines. In particular, the Support Vector Machine (SVM), Relevance Vector Machine (RVM) and Probabilistic Classification Vector Machine (PCVM). Although, the SVM is a popular solution for supervised learning, it comes with several disadvantages.[19] To tackle these problems, Tipping et al. proposed the RVM in [81] to align these drawbacks. The idea of the RVM is mainly motivated to improve the SVM[81, p. 1-2]. However, the RVM itself provides some conception assumptions, which are leading in the opinion of Chen et al. to unstable results. Therefore, concerning these issues, they proposed the PCVM. Regarding to Chen et al., the PCVM is an improved version of the RVM.[19]

### 2.1 Support Vector Machine

The SVM is a supervised learning algorithm and is designed for classification or regression tasks.[8, p. 325]

A key property of the SVM is that the model parameters are determined with a convex optimization problem. With this, any local solution is also global. The Lagrange multipliers are used in the dual optimization problem.[8, p. 328-239]

The SVM is a maximum margin classifier with the goal to determine a margin to separate two classes of samples. This margin which should be maximal. Therefore, the margin is the smallest distance between the decision boundary and any sample.[8, p. 327]

The classifier in his simplest form and apart from the choice of a kernel has two model parameters  $\mathbf{w}$  and  $b$ . The first one is modeling the hyperplane which separates the two classes. This is just introduced by the decision boundary. The second parameter  $b$  is called the bias and models the location of the hyperplane.[8, p. 327-328]

If a sample has a certain distance to the hyperplane, then it is considered as a support vector. The support vectors are on both sides of the hyperplane for the two classes.

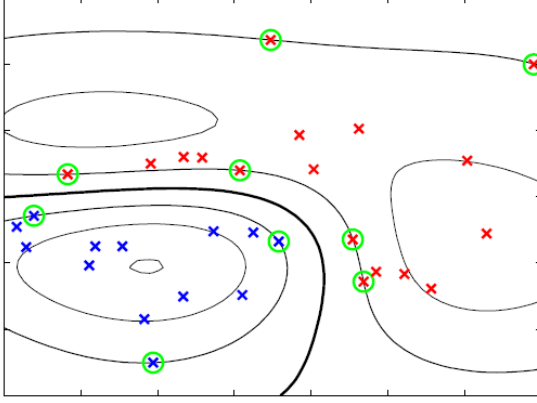


Figure 2.1: Example of the SVM trained with the synthetic dataset with two classes in two dimensions. On the left, the circled data points are the support vectors.[8, p. 331]

The prediction is made by determining the edge of a data point corresponding to the decision boundary. If a new sample point is on one side of the hyperplane, the point belongs to the corresponding class and vice versa. This is expressed as:[8, p. 236;328]

$$y(\mathbf{x}) = \sum_{i=1}^N w_i \phi_i(\mathbf{x}) + b = \mathbf{w}^T \Phi(\mathbf{x}) + b \quad (2.1)$$

Where  $\Phi(\mathbf{x}) = (\phi(\mathbf{x})_1, \dots, \phi(\mathbf{x})_N)$  as feature matrix and  $\phi(\cdot)$  denotes a features space transformation. Furthermore, the hyperplane parameter  $\mathbf{w}$  and the bias  $b$ . The class of a new data point is now obtained with the sign of  $y(\mathbf{x})$ . If  $y(\mathbf{x}) > 0$ , then it belongs to class 1. Also, when it comes to  $y(\mathbf{x}) < 0$ , it belongs to class  $-1$ . Note that if a point in the training state has the function value of  $y(\mathbf{x}) = 1 \vee y(\mathbf{x}) = -1$ , then it is considered as support vector.[8, p. 237]

The SVM has a complexity of  $\mathcal{O}(B^3)$  in the worst case, with  $B$  basis functions.[8, p. 329] To get a visual idea of how the SVM works, the process is shown in figure 2.1. Note that this is a very brief explanation of the SVM, which takes some assumptions. Some concepts, for example, errors and the Lagrange variables, are not defined. This is just to get an idea of how it works.

The SVM is the baseline classifier of every transfer learning method and therefore can be treated as underlying algorithm, except PCVM, PCTKVM, PCTKVM <sub>$\theta_{\text{Est}}$</sub>  and GFK. The latter is using an own approach of  $k$ -nearest-neighbor. In this thesis, the LibSVM is used as implementation. It is created and maintained from Chih-Jen Lin and Chih-Chung Chang and can be downloaded from their university page<sup>1</sup>.

## 2.2 Disadvantages of Support Vector Machines

First the SVM, which is described as baseline classifier in section 2.1, provides an optimal solution through the convex optimization problem [8, p. 325], but suffers from a few disadvantages:

<sup>1</sup><https://www.csie.ntu.edu.tw/~cjlin/libsvm/>



The SVM is non-probabilistic. The problem is that hard binary decisions which are made of the SVM cannot catch the uncertainty for predictions. Furthermore, the probabilistic predictions are considered as crucial when posterior probabilities of a class assignment are adapted to varying class priors and asymmetric misclassification costs.[81, p. 239-240]

To solve this problem, there are some post processing methods developed to match the binary import to a probabilistic output. However, this is considered as unreliable by [81, p. 239-240]. This uncertainty can be interpreted as Bayesian probability.[8, p. 21]

Secondly, the number of support vectors needed to create the margin of the decision boundary grows linearly with the size of the training set. With that the computational complexity and model complexity grows and does not lead to sparse models or fast computing. As a consequence, some post-processing is suggested to reduce the complexity of the model.[19]

An example would be to find a set of 'reduced' vectors with  $R$  entries, which approximating the original set of support vectors  $O$ . Note that these reduced vectors are no training samples and are not necessarily lying on the margin. The goal is then to find the smallest  $R$  with  $R \ll O$  for that the loss in the generalization performance is acceptable. However, this approach is computationally costly.[13]

Therefore, the wanted reduction of model complexity is maybe not yet achieved.

Furthermore, the SVM has several parameters that needs to be tuned by parameter optimization. For example the  $C$  parameter, as cost parameter of the hyperplane[84, p. 420], or the parameters of a kernel function, for example, the width of the Gaussian kernel, section 5.2.2. Parameter optimization is done by a grid search in a certain range. For example, evaluate the performance of the SVM for  $C = 1, 2, 5, 10, \dots, 100, 200$  and select the parameter according to the best performance.[19] Finally, when it comes to the interpretation of the results, the SVM provides a good interpretation of how the margin and the decision boundary is created. However, the points which are considered as support vectors and therefore selected in the model, are not representing the actual class data very well, because they are the closest points from one class to the other class.[8, p. 326]

Here a key feature from the RVM takes place. It creates a sparser model, in comparison with the SVM, and provides a probabilistic estimate of the classes, where the relevance vectors are representing the data.[8, p. 335-356]

This effect of the sparse RVM classification in comparison with the SVM can be seen in figure 2.2. The circled points are model vectors. On the left, the SVM classification is shown. On the left, the RVM model is shown.

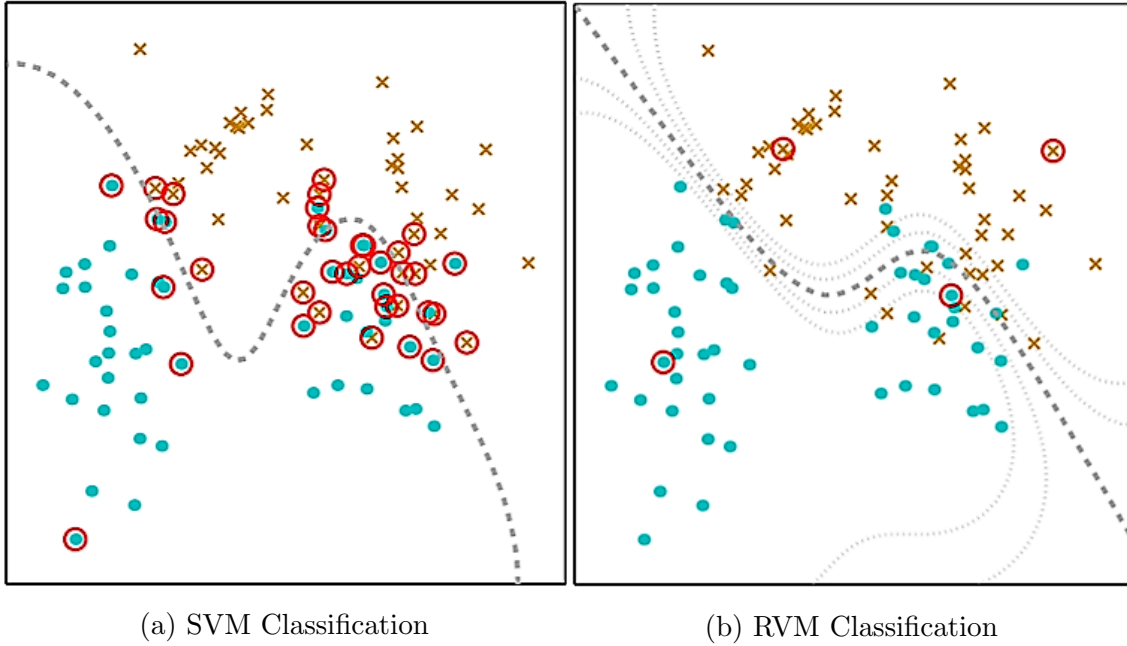


Figure 2.2: An example of classification of the SVM and RVM. The circled points are model vectors. The left shows the SVM classification and the right shows the RVM. It can be seen that the RVM model has fewer model vectors than the SVM. Moreover, the model vectors are a better representation of the data.[9]

## 2.3 Relevance Vector Machine

The RVM is introduced by Tipping in the already noted work [81]. The key idea behind it is to create a classifier with a similar functional form to the SVM with a probabilistic background. It uses a general Bayesian framework.

The RVM uses relevance vectors instead of support vectors. It is found that for many weights the posterior distribution is sharply peaked around zero. That means the probability for a certain weight for the training point is highest nearly the weight zero. The training vectors with the corresponding remaining non-zero weights are called relevance vectors. The weights are representing the importance of a relevance vector.[81, p. 213] Note that although in this thesis the same letter is used for the model vectors ( $\mathbf{w}$ ), there is a significant difference between them. The SVM defines with this parameter the hyperplane and the probabilistic vector machines defines prototypical vectors, which are representing the corresponding class in a probabilistic manner shown in figure 2.2.[81, p. 222]

Another advantage of the RVM against the SVM is, that the kernel function for an SVM has to satisfy Mercer's conditions. The RVM kernel does not have this constraint.[81, p. 213]

Without going deeper in this, a kernel which satisfies the Mercer's conditions is positive semi-definite.[36] The complexity of the algorithm is  $\mathcal{O}(B^3)$  with  $B$  as a number of basis functions and is also similar to the SVM.[81, p. 236-237]

The RVM makes predictions for a new point  $\mathbf{x}$ : [81, p. 211]

$$\mathbf{t} = y(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^N \phi_i(\mathbf{x}) w_i + w_0 = \mathbf{\Phi}(\mathbf{x}) \mathbf{w} + w_0 \quad (2.2)$$

With the bias  $w_0$  and basis functions  $\mathbf{\Phi}(\mathbf{x}) = (\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_N))$ . The weight parameter has the form  $\mathbf{w} = (w_1, \dots, w_N)^T$  and  $\mathbf{t} = t_1, \dots, t_N$  as function value.

In (2.2), the bias is used to move the model out of the origin. The corresponding label of a data point is the sign of the function value  $\mathbf{t}$ . In general,  $\mathbf{t}$  is the regression value. [79, p. 662]

As a consequence, to being able to solve (2.2), the weight  $\mathbf{w}$  with size  $N$  and  $w_0$  must be determined. For classification, the RVM uses the Bayesian theorem. The weights are determined with:

$$p(\mathbf{w}|\mathbf{t}) \propto P(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha}) \quad (2.3)$$

Note that the regression is varying from the classification solution. The RVM for classification gives the probability of a label for a point  $\mathbf{x}$  by applying a logistic sigmoid function, which gives the probabilistic output:

$$\sigma = 1/(1 + e^{-y}) \quad (2.4)$$

With  $y = y(\mathbf{x})$ . Furthermore, the logistic sigmoid function is combined with the Bernoulli distribution of  $P(t|\mathbf{x})$  for the likelihood:

$$P(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^N \sigma\{y(\mathbf{x}_n; \mathbf{w})\}^{t_n} [1 - \sigma\{y(\mathbf{x}_n; \mathbf{w})\}]^{1-t_n} \quad (2.5)$$

Additionally, the prior  $p(\mathbf{w}|\boldsymbol{\alpha})$  is obtained by a zero-mean Gaussian:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^N \mathcal{N}(w_i|0, a_i^{-1}) \quad (2.6)$$

With the inverse hyperparameter vector  $\boldsymbol{\alpha}$  of size  $N + 1$  as the precision of the Gaussian distribution. At this point the parameter  $w_0$  from (2.2) is determined. The parameter  $\boldsymbol{\alpha}$  itself is Gamma distributed. [81, p. 214-215, 218-219]

An idea of a Gamma distribution and the Bernoulli distribution can be obtained from [8, p.686-688].

Because the integral of the Bayesian inference is intractable, the following procedure is used to determine the parameters: The algorithm is trained with the resulting Hessian Matrix from likelihood and prior of 2.3. The most probable weights  $\mathbf{w}_{MP}$  and the corresponding covariance matrix  $\boldsymbol{\Sigma}$  are obtained from the mode of the posterior and the Hessian:

$$\begin{aligned} \boldsymbol{\Sigma} &= (\mathbf{\Phi}^T \mathbf{B} \mathbf{\Phi} + \mathbf{A})^{-1}, \\ \mathbf{w}_{MP} &= \boldsymbol{\Sigma} \mathbf{\Phi}^T \mathbf{B} \mathbf{t} \end{aligned} \quad (2.7)$$

Where  $\mathbf{A} = \text{diag}(\alpha_1, \dots, \alpha_N)$  and  $\mathbf{B} = \text{diag}(\beta_1, \dots, \beta_N)$  with  $\beta_i = \sigma\{y(\mathbf{x}_n)\}[1 - \sigma\{y(\mathbf{x}_n)\}]$ . After the update of  $\mathbf{w}_{MP}$  and  $\Sigma$  the hyperparameters are updated. This is repeated until  $\alpha$  satisfies stable convergence criteria.[81, p. 219]

Although Tipping solved the problems of the SVM with the RVM, Chen et al. proposed that there are some concept problems left.[19] These are discussed in the following sections, especially in section 2.4.2.

## 2.4 Probabilistic Classification Vector Machine

In general, the Probabilistic Classification Vector Machine (PCVM) should provide a more stable solution, sparser model and better performance in comparison with the SVM and RVM.[19]

The PCVM uses a latent variable model because of the assumption that the data is incomplete or latent. Because of this and because that the integral of the Bayesian Inference is, in this case, intractable, it uses an Expectation-Maximization (EM) algorithm, as it is a general solution to get a Maximum a Posteriori (MAP) estimation of parameters based on latent variables. Furthermore, the EM algorithm optimizes the parameters within, and therefore the PCVM does not need a parameter optimization because the free model parameter is optimized within the algorithm.[19]

An Attribute of the EM algorithm is that it is likely to be trapped in local maxima and therefore finds no global solution.[93] Unlike to the SVM which always can determine the global maxima, see section 2.1.

### 2.4.1 Model Specification

The PCVM is made for binary classification[19], unlike the RVM, which has a specification for multi-class.[81, p. 220] But the ideas behind the two concepts are similar as we will see in the following.

In general, it makes predictions for a given test point  $\mathbf{x}$  with the already known prediction function from (2.2) and (2.1).[19] As well as the RVM in section 2.3, the PCVM uses a link function to match the linear to a probabilistic output. In case of the PCVM, it is done with the probit link function defined with:[18]

$$\Psi(\mathbf{x}) = \int_{-\infty}^x N(t|0, 1)dt \quad (2.8)$$

Where  $\Psi$  is the Gaussian Cumulative Distribution Function (CDF), with mean zero and variance one.

With a CDF the probability for a new point  $\mathbf{x}$  of a steady random variable  $X$  can be determined. It is the integral of the density function and gives the probability

$P(X \leq x)$ , which means the probability that  $X$  takes a value equals or less of  $x$ . [78, p. 270]

Therefore, the probabilistic model after integrating the probit link function becomes:

$$l(\mathbf{x}; \mathbf{w}; b) = \Psi \left( \sum_{i=1}^N w_i \phi_{i,\theta}(\mathbf{x} + b) \right) = \Psi(\Phi_{\theta}(\mathbf{x})\mathbf{w} + b) \quad (2.9)$$

Where  $\Phi(\mathbf{x})_{\theta} = (\Phi_{1,\theta}(\mathbf{x}), \dots, \Phi_{N,\theta}(\mathbf{x}))$  as basis functions and  $\mathbf{w} = (w_1, \dots, w_N)^T$  is the weight vector with  $N$  training data points.  $\theta$  is the parameter of the basis function. Because the PCVM uses the Gaussian kernel as basis function, the parameter  $\theta$  is referred as the width of the kernel. [19] The Gaussian kernel is described in the section 5.2.2.

## 2.4.2 Prior over Weights

The three algorithms SVM, RVM and PCVM are determining the weight parameter differently.

It is necessary for a stable solution that the sign of the weight  $w_i$  of a data point  $\mathbf{x}_i$  is equal to the corresponding classes  $y_i \in \{-1, +1\}$ . [19]

The SVM ensures this by defining the weight vectors as  $\mathbf{w} = \{v_1 y_1, \dots, v_n y_n\}$ , where  $v_i$  are non-negative Lagrange multipliers. Therefore, the same sign is guaranteed. [19] But consider, that some of the data points, which are not support vectors, have  $v_i = 0$ . [8, p. 330]

The RVM estimates the weights according to equation (2.3). The likelihood is always positive because the exponential function  $e$  is always positive [39, p. 355] and hence the logistic sigmoid link function it is. Therefore, the sign depends on the prior, which uses the zero-mean Gaussian over all weights and hence providing some problems.

It can happen that training errors occur with the use of the zero-mean Gaussian because a positive relevance vector can get a negative weight assigned and vice versa. This may lead to an unstable result corresponding to non-reliable vectors. As a consequence, the RVM is more likely to overfit noise in comparison with the SVM and PCVM, with same selection of parameters. [19]

At this point the improvements of the PCVM take place. Instead of a zero-mean Gaussian prior, it uses the zero-mean truncated Gaussian as prior: [19]

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \sum_{i=1}^N p(w_i|\alpha_i) = \sum_{i=1}^N N_t(w_i|0, \alpha_i^{-1}) \quad (2.10)$$

Here  $N_t$  is the truncated Gaussian function with  $\boldsymbol{\alpha}$  as inverse variance.

With that the relevance vector is always getting a proper signed weight attached. Because for  $y_i = +1$  the prior is selected from the right-truncated non-negative

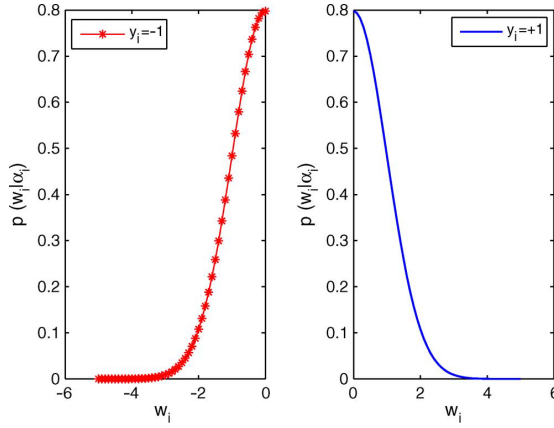


Figure 2.3: The truncated Gaussian prior over weights. On the left side the non-positive left-truncated one in case of a negative class label. On the right side the non-negative right-truncated one for the positive class.[19]

Gaussian and for  $y_i = -1$  it is the left-truncated non-positive Gaussian.[19]

Now to avoid errors corresponding to noise or unreliable points the above is expressed with:[19]

$$p(w_i | \alpha_i) = \begin{cases} 2N(w_i | 0, \alpha_i^{-1}), & \text{if } y_i w_i \geq 0 \\ 0, & \text{if } y_i w_i < 0 \end{cases} \quad (2.11)$$

With that the PCVM assigns for an unreliable point a zero weight and therefore it is no longer considered as relevance vector for the model. The truncated Gaussian prior are illustrated in figure 2.3.

The bias  $b$ , which is referred as  $w_0$  in (2.6), is still determined with the zero-mean Gaussian:[19]

$$p(b | \beta) = N(b | 0, \beta^{-1}) \quad (2.12)$$

The priors in the previous are determined with a Gaussian prior with fixed parameters. These parameters  $\alpha$  and  $\beta$  could also be controlled through a hierarchical hyperparameters with a hyperprior regarding the Bayesian Framework.[19] This means that hyperparameters are controlling the distribution of the prior. [8, p. 71] Because these hyperparameters are controlled by a prior of another distribution, this prior is called hyperprior.[7, .p 423]

The PCVM uses the gamma distribution for modeling the hyperprior, but in practice, these hyperparameters are set to zero.[19]

### 2.4.3 Expectation Maximization Algorithm

The PCVM uses a simple latent variable model within the EM algorithm.[19] A latent variable can either be missing data or is unobservable (hidden).[84, p. 276-277][8, p. 84] Hidden in this context means that this variable cannot directly be observed and his existence is therefore implicit.[10]

The latent variable is a part of the data which is not available (hidden) but is involved in the model. Therefore, the algorithm makes assumptions to deal with the uncertainty of this latent variables.

The standard probabilistic assumption is that the predictor function in (2.2) is affected by noise. Therefore, (2.2) can be reformulated to:[19]

$$h_\theta(\mathbf{x}) = \Phi_\theta(\mathbf{x})\mathbf{w} + b + \epsilon \quad (2.13)$$

With the  $\theta$  parameter for the basis function and the bias  $b$  instead of  $w_0$ . The noise is distributed according to the standard Gaussian as  $\epsilon \sim N(0, 1)$ . Furthermore, because  $\epsilon$  is unobservable thus latent,  $h_\theta(\mathbf{x})$  itself will be considered as latent variable.[19]

The PCVM uses the probit link function, which is originally used for regression.[3]. It assigns a variable  $l = 1$ , when the corresponding  $h_\theta(\mathbf{x}) \geq 0$  and  $l = 0$  for  $h_\theta(\mathbf{x}) < 0$ . [19]. Therefore, to obtain the probit mode, the Gaussian CDF is used:

$$P(l = 1|x, w, b) = p(\Phi_\theta(\mathbf{x})\mathbf{w} + b + \epsilon \geq 0) = \Psi(\Phi_\theta(\mathbf{x})\mathbf{w} + b) \quad (2.14)$$

It states, given the data  $\mathbf{x}$  and the parameter  $\mathbf{w}$  and  $b$ , how likely it is that  $\mathbf{x}$  belongs to the positive class. To get the probability for the other class it is simply  $P(l = 0|x, w, b) = 1 - P(l = 1|x, w, b)$ . [19]

Revisiting (2.2), we need to determine  $\mathbf{w}$  for the model. Again the Bayesian theorem is used to determine most probable parameter values.[19]

The prior for  $\mathbf{w}$  is already obtain through (2.10) and the bias is specified with (2.12) as zero-mean Gaussian. Therefore, only the likelihood has to be specified. Taking the assumption that  $h_\theta(\mathbf{x})$  is known, the Gaussian likelihood could be obtained with:[19]

$$\begin{aligned} p(\mathbf{H}_\theta(\mathbf{x})|w, b) &= N(\mathbf{H}_\theta(\mathbf{x})|\Phi_\theta(x)\mathbf{w} + b, 1) \\ &= (2\pi)^{N/2} \exp\left\{-\frac{1}{2} \|\mathbf{H}_\theta - \Phi_\theta\mathbf{w} + b\mathbf{I}\|^2\right\} \end{aligned} \quad (2.15)$$

With  $\mathbf{H}_\theta(\mathbf{x}) = (h_\theta(\mathbf{x}_1), \dots, h_\theta(\mathbf{x}_N))^T$  and where  $\Phi_\theta = (\Phi_\theta(\mathbf{x}_1)^T, \dots, \Phi_\theta(\mathbf{x}_N)^T)^T$  and  $\Phi_\theta(\mathbf{x}_i) = (\phi(\mathbf{x}_1, \mathbf{x}_i), \dots, (\phi(\mathbf{x}_N, \mathbf{x}_i)))$  forming the kernel. At this point it is important to reintroduce a clear notation.  $\Phi_\theta$  as kernel. Within this kernel,  $\Phi(\mathbf{x}_i)$  is a row and  $\Phi_\theta(\mathbf{x})$  refers to the row corresponding to  $\mathbf{x}$ .

For the complete log-posterior of the parameters  $\mathbf{w}$  and  $b$  the corresponding parameters,  $\alpha$  and  $\beta$  are also considered as latent variables. With that there are three latent variables in the posterior. Instead of the standard posterior, the log posterior is used and therefore has the form:[19]

$$\begin{aligned} \log p(\mathbf{w}, b|\mathbf{y}, \mathbf{H}_\theta, \alpha, \beta) &\propto \log p(\mathbf{H}_\theta|\mathbf{w}, b) + \log p(\mathbf{w}|\alpha) + \log p(b|\beta) \\ &\propto \mathbf{w}^T \Phi_\theta^T (2\mathbf{H}_\theta - \Phi_\theta\mathbf{w}) + 2b\mathbf{I}^T \mathbf{H}_\theta - 2b\mathbf{I}^T \Phi_\theta - b^2 N - \mathbf{w}^T \mathbf{A} \mathbf{w} - \beta b^2 \end{aligned} \quad (2.16)$$

With that we can proceed with the EM algorithm.[19]

Dempster et al originally introduced the EM algorithm. It is made for determining the maximum likelihood for incomplete data [24]. Based on their work, many other

applications are made to determine parameters. For example, Bishop used a EM algorithm within  $k$ -means cluster.[8, p. 426-428] Moreover, Hasti et al. use it in [84, p. 272-276] for a two-component mixture model. Although Tipping never mentioned the use of the EM algorithm in the RVM, the procedure of it, explained in 2.3 is similar.[81, p. 233-234]. Furthermore, he introduced an Expectation-Maximization Update approach for determining the parameters.[81, p. 235]

As the name says, it consists of two steps. The algorithm has an  $E$  and  $M$  steps per iteration and will repeat as long as a convergence criterion, or the maximum iteration is reached, which can be specified in the PCVM algorithm (appendix 3). The first, the Expectation-Step finds expectations for the latent variables and results in an expectation for the posterior of the parameters, which is referred as  $\mathcal{Q}$  function. In the case of the PCVM it has the form:[19]

$$\begin{aligned} \mathcal{Q}(\mathbf{w}, b | \mathbf{w}^{old}, b^{old}) &= E_{\mathbf{H}_\theta, \alpha, \beta} [\log p(\mathbf{w}, b | \mathbf{y}, \mathbf{H}_\theta, \alpha, \beta) | y, \mathbf{w}^{old}, b^{old}] \\ &= 2\mathbf{w}^T \Phi_\theta^T \bar{\mathbf{H}}_\theta - \mathbf{w}^T \Phi_\theta^T \Phi_\theta \mathbf{w} + 2b \mathbf{I}^T \bar{\mathbf{H}}_\theta - b^2 N + 2b \mathbf{I}^T \Phi_\theta \mathbf{w} - \mathbf{w}^T \bar{\mathbf{A}} \mathbf{w} - \bar{\beta} b^2 \end{aligned} \quad (2.17)$$

With  $\bar{\mathbf{H}}_\theta = E[\mathbf{H}_\theta | y, \mathbf{w}^{old}, b^{old}]$ ,  $\bar{\mathbf{A}} = \text{diag}(E[\alpha_i | y_i, \mathbf{w}^{old}, b^{old}])$  and  $\bar{\beta} = E[\beta | y_i, \mathbf{w}^{old}, b^{old}]$ . For details to the determination of the expectations, see appendix B in [19].

This step relies on the 'old' latent variables and parameters from the previous iteration. The next part in the iteration is the Maximization-Step.[19]

The Maximization-Step is done by analyzing the derivative of equation 2.17, to find the most likely values concerning the parameters. However, a joint optimization is not possible. Therefore, every parameter is optimized separately:[19]

$$\mathbf{w}_{new} = (\Phi_\theta^T \Phi_\theta + \bar{\mathbf{A}})^{-1} (\Phi_\theta^T \bar{\mathbf{H}}_\theta - b \Phi_\theta^T \mathbf{I}) \quad (2.18)$$

$$b_{new} = \frac{\mathbf{I}^T \bar{\mathbf{H}}_\theta - \mathbf{I}^T \Phi_\theta \mathbf{w}}{\beta + N} \quad (2.19)$$

With that the maximum is found and therefore the 'new' parameters for  $\mathbf{w}^{old}$  and  $b^{old}$  for 2.17 are found for the next iteration. The optimization for  $\theta$  cannot be done analytically and is done by a conjugate gradient algorithm<sup>2</sup> which is done on basis of:[19]

$$\frac{\partial \mathcal{Q}}{\partial \theta_k} = 2 \sum_{i=1}^N \sum_{j=1}^N \left\{ (\Phi_\theta \mathbf{w} - \bar{\mathbf{H}}_\theta) \mathbf{w}^T \odot \left( \frac{\partial \Phi_\theta}{\partial \theta_k} \right) \right\}_{(i,j)} \quad (2.20)$$

With  $\odot$  as Hadamard matrix multiplication as defined in [15].

In practice, the implementation does differ from the original parameter estimation, seen in equation 2.18 and 2.19. The reason for that is that the small values of  $\bar{\mathbf{A}}$  are heading towards zero and are extended with a more regularized matrix.[19]

<sup>2</sup><http://learning.eng.cam.ac.uk/carl/code/minimize/>



### 2.4.4 Properties of Algorithm

The algorithm of the PCVM can simply be described. First initialize the parameters  $\mathbf{w}$  and  $b$  randomly and an indicator vector, which indexes the non-zero weights in  $\mathbf{w}$ . In every following iteration, the kernel with the current  $\theta$  is calculated. This is followed by the estimation of the weight  $\mathbf{w}$ , bias  $b$  and solving the optimization for a new  $\theta$ . After these steps, the indicator vector is updated. This is repeated until a convergence criterion is reached.[19] The complete pseudo code can be found in appendix C.

Chen et al. designed the PCVM in a way that the convergence of the algorithm is only measured with the convergence behavior of  $\mathbf{w}$  and is set to  $\|\mathbf{w} - \mathbf{w}^{old}\| = 1.0e^{-3}$  in practice.

In practice the Cholesky decomposition is used for the matrix inversion. This has the computational complexity of  $\mathcal{O}(B^3)$ , where  $B$  is the number of non-zero basis functions and is the overall complexity of the PCVM. Furthermore, it has a memory complexity of  $\mathcal{O}(B^2)$ .[19]

However, because of the sparsity of the prior, the PCVM reduces the basis function from the initial start with  $B = N$  to  $B < N$ .[19] In the best case it is even  $B \ll N$ . A practical example of this reduction can be found in table B.4 in appendix B. This table shows the number of vectors which are needed for the model. It can be seen that the PCVM has a sparser model in comparison to the SVM and has therefore less basis functions. Note that because of many zero-mean Gaussian assumptions, the training and test data must be z-scored.

## 2.5 Conclusion

Summarizing, Chen et al. have proposed the PCVM as an improvement to the SVM and RVM. They have evaluated this statement, in a performance study on three synthetic datasets and twelve benchmark datasets. They determined statistical significance with the 5x2 cv F test and the Friedman test. The result of this study is that the PCVM can achieve a greater performance on benchmark and synthetic datasets in comparison to the RVM and SVM.

In a nutshell, the SVM has a convex optimization problem which results in an optimum solution. The PCVM is randomly initialized and with the EM algorithm, it has only local optima and as a consequence, the model slightly differs from every training run. The complexity of the SVM is  $\mathcal{O}(N^3)$ , as the number of training basis functions. The PCVM has a complexity of  $\mathcal{O}(B^3)$ , with  $B$  as the number of non-zero basis functions and  $B \ll N$ , in the best case. The PCVM produces a reliable probabilistic output, but the probabilistic output of the SVM is not trustable. The resulting model of the PCVM gives a good interpretable model, which is sparse.

However, the original SVM does not achieve a sparse model and does not provide an interpretable model. Also, the size of the SVM model grows linearly with the size of dataset, however the PCVM can provide sparseness with large datasets.

Moreover, the PCVM in comparison with the RVM, it uses a more reasonable prior.[19]

Note that Chen et al. claimed that the PCVM has a better performance as the SVM. However this statement cannot be held when it comes to transfer learning, as we will see in the following chapters.

# Chapter 3

## Transfer Learning

In this chapter, the technique of transfer learning will be introduced. It can be considered as a broad interpretation of related work.

This chapter is organized as follows: First of all, we discuss some challenges of traditional machine learning algorithms. Furthermore, some real world and machine learning sample problems are discussed.

In the second section, the task transfer learning in the context of classification will be described.

There are various definitions of transfer learning in general. Many of the solutions obtained in this chapter are made for different definitions. Moreover, because of this the difficulty of the problem which is solved by the solutions may vary, see 3.2.

Additionally, we will see how the difference in transfer learning is measured.

Then, the settings of transfer learning will be introduced. The settings are implying the different setups to solve the transfer problem.

In the next section, the two main types of the transfer learning, which describe the conditions of feature space and probability distributions, will be introduced and explained. These types can be divided into approach-categories. Most of the discussed solutions are based on the formal ideas of a category.

The section, negative transfer deals with the problem that the provided transfer learning solutions are in fact worse than the baseline methods. For details to kernels see section 5.2.2.

### 3.1 Challenges of Supervised Learning Algorithms

The current state of 'traditional' supervised learning algorithms is the assumption that training and test data existing in the same feature space and that they have the same distribution. They are successfully learning patterns and predict events in the future. However, it is not always possible to obtain training and test data, which matches in distribution or features space. A reason for this is that training

data, which has to be labeled, is expensive or difficult to achieve.[88, p. 1]

When it comes to supervised learning, the algorithms are designed in a way to learn from the pre-labeled data and hence it is crucial for this types of algorithms to have some.[80, p. 6-7]

In the 'traditional' manner, which applies to the most statistical models, to solve the problem of differences, one has to collect new training data for the new feature space or distribution of the test data and rebuild the model.[66] Therefore, the good performance cannot be obtained, when the training and test data differs in feature space or distribution. This is the reason, one has to find a classifier, which can be applied to the target domain, although it is trained on the source domain This is the main motivation of Transfer Learning.[88, p. 1.]

A simple interpretation of Transfer Learning in the real world is given in the following: Imaging two people, who want to learn the piano. One has never played any musical instrument before, while the second one has previous knowledge on how to play the music through experience with other instruments. The latter will be able to learn the new instrument much faster than the first one with no previous experience because he can transfer his general music knowledge to boost his learning of the new instrument.[88, p. 1]

When going back to machine learning, there are more technical examples. Consider the task of web document classification. The goal is to classify web documents in predefined categories. The training data in this example are university web pages, which are associated with a category by manual labeling. If we want to classify our test data, which does not come from a university page, then the data features or distribution is different, based on the structure or topics of the new web pages. Therefore, we can not directly apply the classifier trained on the university web pages and assume a good category prediction rate on non-university web pages. Instead, we have to collect new websites from the same type as of the test web pages, assign the category manually and finally learn a new model. This process could be avoided if a classifier could transfer knowledge from one domain of pages into another web page domain.[66]

The last example is given from the topic of product reviews. Consider the problem that a model needs to be learned, in a way that it can automatically classify positive and negative reviews on a product. To create a source domain, one has to collect many reviews of the product and manually decide if it is positive or negative. Then the model can be trained and predict the sentiment of new reviews. However, if the product changes, then the distribution of the review may change, and the traditionally learned classifier cannot be applied anymore. Therefore, again one has to collect reviews manually and annotate them. This can be a very expensive process, and therefore the learner should be adaptable so that it can be learned on

the source domain (one product category) and can help to learn the classification model (another product category).[66]

## 3.2 Definition of Transfer Learning

Now that more practical examples have been given, the term transfer learning and associated keywords are defined in this section.

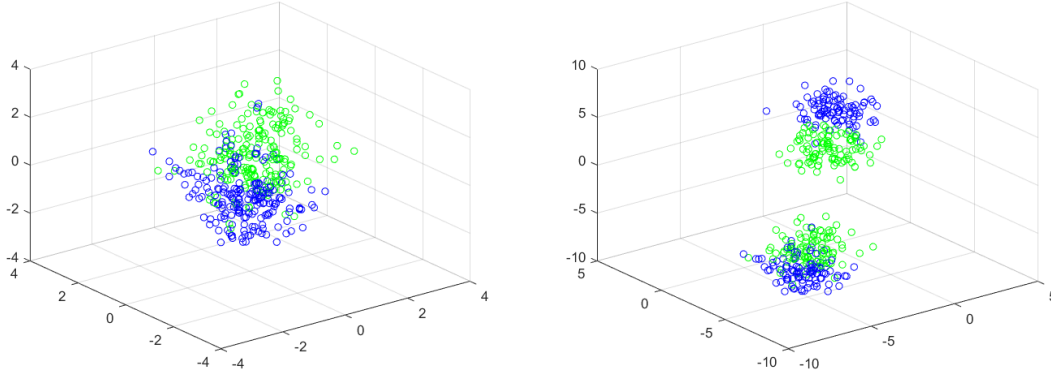
In this thesis, a domain  $\mathcal{D}$  is composed of a  $D$ -dimensional feature space  $\mathcal{F}$ , which can be feature extraction a of text documents. Moreover, a marginal probability distribution of a data matrix  $\mathbf{X}$  is denoted with  $P(\mathbf{X})$ . Formally, this means  $\mathcal{D} = \{\mathcal{F}, P(\mathbf{X})\}$  with  $\mathbf{X} \in \mathcal{F}$ . In general, if two domains  $\mathcal{X}$  and  $\mathcal{Z}$  are different, then they have either different feature spaces or marginal distributions.

This can be expressed as  $\mathcal{F}_{\mathcal{Z}} \neq \mathcal{F}_{\mathcal{X}} \vee P(x) \neq P(z)$ . [2, p. 542] The issue of different distributions is illustrated in figure 3.1. In 3.1a the traditional problem with similar source and target probability distributions is shown. The figure 3.1b shows the problem of transfer learning where conditional and marginal probability distributions are different (two domains). The samples are generated with the Gaussian-Random function in MatLab. The blue and green colors are showing two classes. Moreover, 3.1b shows the problem with two domains.

A task  $\mathcal{T}$  with a given domain  $\mathcal{D}$  is put together with a label set  $\mathcal{Y}$ , for example  $\{-1, 1\}$ , and a classifier  $f(\cdot)$  which is  $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$  with  $\mathbf{y} \in \mathcal{Y}$ . The vector  $\mathbf{y}$  represent the (predicted) labels for *all* data points regarding the label set  $\mathcal{Y}$ . The function  $f(\cdot)$  is a predictive function to make predictions for unseen, i. g. new data points  $\mathbf{x}$ . From a probabilistic viewpoint we can interpret  $f(\mathbf{x})$  as  $P(y | \mathbf{x})$ . In other words, given  $\mathbf{x}$  how probable it is that label  $y$  is correctly assigned. For example, based on the content of a document, how probable it is to be correctly assigned to certain category. [2, p. 542]

In classification, the number of varying labels will distinguish the classification problem. The Binary classification problem for two labels, e. g.  $\mathcal{Y} = -1, +1$  or discrete values, i. e. multiple classes, for example  $\mathcal{Y} = \{1, 2, \dots, C\}$  with  $C$  classes. In general, two tasks  $\mathcal{T}_{\mathcal{X}}$  and  $\mathcal{T}_{\mathcal{Z}}$  are different if they have varying conditional probability distributions or label spaces, which means  $\mathcal{Y}_{\mathcal{X}} \neq \mathcal{Y}_{\mathcal{Z}} \vee P(y | \mathbf{x}) \neq P(y | \mathbf{z})$ . [2, p. 542]

Note that in this thesis if we talk about transfer learning, then the training domain is denoted as  $\mathcal{Z}$  and  $\mathcal{X}$  represents the test domain. For the binary case within transfer learning the following definitions are made: Consider  $\mathcal{D}_{\mathcal{Z}} = \{(\mathbf{z}_i, y_{\mathcal{Z}i})\}_{i=1}^N$  as source domain data where  $\mathbf{z}_i \in \mathcal{F}_{\mathcal{Z}}$  as a single observation, for example, one document in the feature space. It is part of the training data matrix and therefore is part of  $P(\mathbf{Z})$ . Moreover,  $y_{\mathcal{Z}i} \in \mathcal{Y}_{\mathcal{Z}}$  is the corresponding class label, which forms the whole  $N$  sized label vector  $\mathbf{y}_{\mathcal{Z}}$ . Additionally, the target domain data  $\mathcal{D}_{\mathcal{X}} = \{(\mathbf{x}_i, y_{\mathcal{X}i})\}_{i=1}^M$ .



(a) Traditional Learning Problem

(b) Transfer Learning Problem

Figure 3.1: The figures show the comparison of distributions for two classes in the traditional and the transfer learning state. The left shows one domain with where the classes are similarly distributed. The right figure shows two different domains with different marginal and conditional probability distributions.

Analogue,  $\mathbf{x}_i$  is a single observation with  $\mathbf{x}_i \in \mathcal{F}_{\mathcal{X}}$  and the corresponding class label  $y_{\mathcal{X}i} \in \mathcal{Y}_{\mathcal{X}}$ , which is again summarized as  $M$  sized training vector  $\mathbf{y}_{\mathcal{X}}$ . [2, p. 2]

Note that we are doing the assumption that our problem has by default more than one dimension and therefore we write a single observation as a vector of features. Because of this, the whole data is represented as a matrix.

**Definition 1 (Transfer Learning)** *Given a source domain  $\mathcal{Z}$  and learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and learning task  $\mathcal{T}_{\mathcal{X}}$ , transfer learning aims to help improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , where  $\mathcal{Z} \neq \mathcal{X}$ , or  $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$ . [2, p. 542]*

Summarizing, our source domain consists of  $\mathcal{Z} = \{\mathcal{F}_{\mathcal{Z}}, P(\mathbf{Z})\}$  and task  $\mathcal{T}_{\mathcal{Z}} = \{\mathcal{Y}_{\mathcal{Z}}, f_{\mathcal{Z}}(\mathbf{Z})\}$  with  $f_{\mathcal{Z}}(\mathbf{Z}) = P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z})$ . Moreover, the target domain is  $\mathcal{X} = \{\mathcal{F}_{\mathcal{X}}, P(\mathbf{X})\}$  and  $\mathcal{T}_{\mathcal{X}} = \{\mathcal{Y}_{\mathcal{X}}, f_{\mathcal{X}}(\mathbf{X})\}$  with  $f_{\mathcal{X}}(\mathbf{X}) = P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$ . If we consider that the training and testing domain are equal and their learning tasks are the same, we step back to the traditional machine learning problem. Note that in this thesis the size of the whole dataset (source and target data points) is denoted as  $K = N + M$ . Furthermore, if any of the methods are transforming the original features into a subspace. The dimension of the subspace is denoted with  $L$ . The notation is summarized in appendix A table A.1.

According to our challenges from section 3.1, the definition can be explained with the help of the web document classification example. [88, p. 4] The example can be divided into two cases: In case one, if two domains are different, which means that  $\mathcal{Z} \neq \mathcal{X}$  as above, then they differ either the feature space  $\mathcal{F}_{\mathcal{Z}} \neq \mathcal{F}_{\mathcal{X}}$  or in their marginal probability distribution  $P(\mathbf{Z}) \neq P(\mathbf{X})$ . Going back to the web document

example, the first would imply that the language of the two documents is different. The second could be caused by the fact that the documents are focusing on various topics, which would lead to different term features.[66]

The second case is that the tasks are different, which implies  $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$  and can further be divided that they rather differ in the label space  $\mathcal{Y}_{\mathcal{Z}} \neq \mathcal{Y}_{\mathcal{X}}$  or the conditional probability of the two tasks are different with  $P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z}) \neq P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$ . In practice, the first would imply that the source task has only two categories, which results in a binary classification problem, although the target task has more than two labels and is, therefore, a multi-class problem. The second case could be based on the fact that source and target documents are divided into very unbalanced user-defined classes. Finally, if there exists any relation between the feature spaces of the two domains, then they are *related*.[66]

Note that in the context of transfer learning the term Domain Adaptation (DA) may appear. Based on the discussion in [65], the following definition can be extracted.

**Definition 2 (Domain Adaptation)** *Given a source domain  $\mathcal{Z} = \{\mathcal{F}_{\mathcal{Z}}, P(\mathbf{Z})\}$  and learning task  $\mathcal{T}_{\mathcal{Z}}$  and a target domain  $\mathcal{X} = \{\mathcal{F}_{\mathcal{X}}, P(\mathbf{X})\}$  and learning task  $\mathcal{T}_{\mathcal{X}}$ . Let  $\mathcal{T}_{\mathcal{Z}} = \mathcal{T}_{\mathcal{X}}$  and the domains  $\mathcal{Z} \neq \mathcal{X}$  in a way that  $\mathcal{F}_{\mathcal{Z}} = \mathcal{F}_{\mathcal{X}}$  and  $P(\mathbf{Z}) \neq P(\mathbf{X})$ , domain adaptation aims to help improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$ .*

This is related to the covariance shift problem.[65] It assumes that the domain data generation is made in a way that the sampling, which the data is based on, is  $P(\mathbf{y}|\mathbf{X})P(\mathbf{X})$ . Furthermore, the marginal distribution  $P(\mathbf{X})$  changes through the domain shift that the marginal distribution for data sampling of the source and target data are different. As a consequence, the distributions are different.[70, p. 8-9]

In practice, we can observe that the key challenge of Domain Adaptation methods is to focus on aligning the differences in the marginal distribution, which can be observed in [65], [56], [30], [5] and [65].

### 3.3 Measurement of Difference

From the previous examples in 3.1 and definitions in 3.2, it seems that the domains are different. This difference needs to be determined to classify the amount of transfer which is done. In this thesis, it is measured with the Maximum Mean Discrepancy (MMD) and Kullback-Leibler Divergence (KLD). Note that they only consider the differences in the marginal probability distributions. Another, point worth mentioning is that the methods, which we will introduce later on, are based on these measures. For example, the Transfer Component Analysis and the Joint Domain Adaption solutions are using the MMD in their optimization problems. On

the other hand, the authors of TrAdaBoost using the KLD to determine the difference in domains, to measure the quality of the transfer-algorithm.[56][65][22]

### 3.3.1 Maximum Mean Discrepancy

First, the Maximum Mean Discrepancy is a non-parametric function to measure the difference of two independent and identically distributed (IID) samples  $\mathbf{X}$  and  $\mathbf{Y}$  drawn from  $p$  and  $q$  respectively, which is introduced by [38, p. 724-728].

Two samples of a random variable are independent if a current sample is not influenced by any previous samples and will not influence following samples. Identically distributed, as the name says, requires that the random variables are must have the same distribution.[21, p. 7-8]

The MMD is calculated with samples drawn from two distributions. It measures the distance between the expectations of two samples,  $\mathbf{X} = \{x_1, \dots, x_N\}$  and  $\mathbf{Y} = \{y_1, \dots, y_M\}$  in a Reproducing Kernel Hilbert Space (RKHS)  $\mathcal{H}$ . Assuming that  $\mathbf{X} \sim p$  and  $\mathbf{Y} \sim q$ . Because  $\mathcal{H}$  is a Hilbert space, the mapping function is defined as  $f(x) = \langle f, \phi(x) \rangle_{\mathcal{H}}$  with  $\phi(x) = k(x, \cdot)$  as positive semi-definite kernel. Note that a Positive Semi-definite Kernel (PSD) kernel, has only non-negative eigenvalues.[74, p. 30]

With the use of the RKHS the authors claim, that the  $f(\cdot)$  is smooth and therefore the MMD it is. Moreover, the MMD is easily expendable, for example, with new points drawn from  $p$  and  $q$ . Finally, the MMD and his empirical estimation are defined as (3.1) and (3.2):[38, p. 726-727]

$$MMD[p, q]^2 := [\sup(E_p[f(x)] - E_q[f(y)])]^2 \quad (3.1)$$

$$MMD_e[\mathbf{X}, \mathbf{Y}] := \left\| \frac{1}{M} \sum_{i=1}^M f(\mathbf{x}_i) - \frac{1}{N} \sum_{i=1}^N f(\mathbf{y}_i) \right\| \quad (3.2)$$

Note that  $f(\cdot)$  is the kernel function. Furthermore, the empirical MMD is biased. If  $p$  and  $q$  both are equal probability distributions, respectively  $p = q$ , then  $MMD_e = 0$ . [38, p. 726-727]

The MMD is symmetric after it takes the absolute value, as shown in (3.2).

Another advantage of (3.2) is that it can be calculated based on samples. Unlike the following Kullback-Leibler Divergence where the density has to be estimated before, the KLD can be calculated.[57]

The code for the MMD estimation can be obtained from the website of the Gatsby Computational Neuroscience Unit<sup>1</sup>, which uses the standard Gaussian kernel for feature mapping.

<sup>1</sup><http://www.gatsby.ucl.ac.uk/~gretton/mmd/mmd.htm>



### 3.3.2 Kullback-Leibler Divergence

The Kullback-Leibler Divergence is also named Kullback-Leibler-Risk. It needs two probability distributions  $f$  and  $g$  of a random Variable  $X$ . For discrete variables  $x_i, i = 1, \dots, M$  with  $p_i = f(x_i)$  and  $q_i = g(x_i)$  as probabilities respectively, the KLD is calculated as:[20, p.5-7]

$$KL(g | f) = \sum_{i=1}^M p_i \log \frac{p_i}{q_i} \quad (3.3)$$

For continues variables the KLD is computed:

$$KL(g | f) = \int f(x) \log \frac{f(x)}{g(x)} dx \quad (3.4)$$

For all  $f$  and  $g$  it is  $KL(g | f) \geq 0$ . Next,  $KL(f | f) = 0$  and finally, if  $KL(g | f) = 0$ , then  $f = g$ , which implies that the two distribution are nearly the same. Note that the KLD is not symmetric and as a consequence the switch of variables will result in another KLD[20, p.5-7]

Because the density has to be estimated, the KLD in this thesis will be determined by a Bayesian method approach, which is described in the supplementary material of [6]. The source code for the KLD estimation can be found on GitHub<sup>2</sup>.

## 3.4 Settings of Transfer Learning

In this section, the different settings of transfer learning are proposed. There are various settings of the transfer learning definition, which specifies the composition of domains and tasks and how they are related to each other. Therefore, each setting has his requirements on how labeled data in the source and target domain is available. In general, the settings could be summarized with 'What to transfer?' and 'How to transfer?'. The various settings are related to some traditional machine learning task.[66]

As we will see in the following, the definitions of homogeneous (3.5) and heterogeneous (3.6) transfer learning and the definitions of the settings will overlap.[88, p. 5-6]

In general, we will stick to the definitions of homogeneous and heterogeneous, because these are more general in comparison with the settings. Therefore, the settings are used to express the categories of the transfer learning methods more precisely. Additionally, but more importantly, we want to focus on the fact, whether labeled or unlabeled target data are required for the knowledge transfer. The practical methods will be explained within the section of homogeneous and heterogeneous transfer learning. The relationship is shown in figure 3.2.

<sup>2</sup><https://github.com/pberkes/neuro-kl>

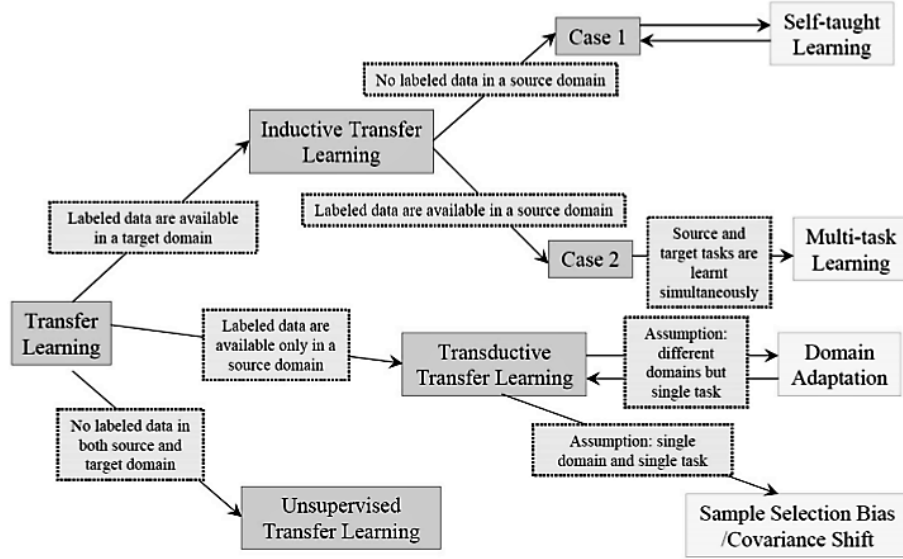


Figure 3.2: This figure shows the various settings for transfer learning and their relation to traditional machine learning or sampling problems. Moreover, the figure shows the required dataset compositions regarding unlabeled or labeled target data.[66]

### 3.4.1 Inductive-Transfer Learning

The setting Inductive Transfer Learning (ITL) makes the assumption that source task and target task are different, regardless of whether the domains of source and target are different or not. Methods of this setting requiring some labeled test data to *induce* a predictive model  $f_{\mathcal{X}}(\mathbf{X})$  for the training domain.[66] Therefore, Pan et al. formulated the following definitions.

**Definition 3 (Inductive Transfer learning [66])** *Given a source domain  $\mathcal{Z}$  and a learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and a learning task  $\mathcal{T}_{\mathcal{X}}$ , inductive transfer learning aims to help improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , where  $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$ , while some labeled target domain data is available.*

With this in mind, there are in fact two special cases to consider. The first one tries to achieve high performance in the target task by requiring not only some labeled test data but lots of labeled training data. This is related to the multi-task learning approach where a learner tries to learn both domains, the source and target domain, simultaneously.[66]

In the second case, there is no labeled source data at all available. The goal is similar to the self-thought learning setting. The challenge of self-thought learning is that information from the source domain can not be applied to the target because they are not the same and therefore can not be used directly. Summarizing, because the source data cannot be used directly, we have to find parts of the data which can be

combined with a few labeled target data to reach the goal.[66]

In the section 3.5.1, we will give an example of an instance based approach, based on the inductive transfer learning setting. Note that this solution tries to align the marginal distributions with some labeled data, which is little different from the original definition of inductive learning.

### 3.4.2 Transductive-Transfer Learning

The Transductive Transfer Learning (TTL) setting assumes that the tasks of two domains are the same, but the source and target domains are different. Furthermore, it requires a lot of labeled data in the source domain and assumes no labeled data in the target domain and has the requirement that some target data is available at training time.[66] The formal definition is made with:

**Definition 4 (Transductive Transfer Learning [66])** *Given a source domain  $\mathcal{Z}$  and a corresponding learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and a corresponding learning task  $\mathcal{T}_{\mathcal{X}}$ , transductive transfer learning aims to improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , where  $\mathcal{Z} \neq \mathcal{X}$  and  $\mathcal{T}_{\mathcal{Z}} = \mathcal{T}_{\mathcal{X}}$ . In addition, some unlabeled target domain data must be available at training time.*

The term *transductive* has several meanings. In traditional machine learning, it means that all target domain data has to be available at training time and as a consequence, the learned model cannot be reused with new, i.e. unseen data. However, this can be relaxed with the result that in the transfer learning topic, *transductive* means that the task must be the same and only *some* unlabeled target has to be available at training time.[66]

The transductive setting can be divided into two special cases: The feature spaces between the source and target data are different with  $\mathcal{Z} \neq \mathcal{X}$ . This is related to the definition of heterogeneous transfer learning from section 3.6. The second case is that the feature spaces are the same, but the marginal probability distributions differ, i.e.  $P(\mathbf{Z}) \neq P(\mathbf{X})$ .[66]

In this case, the transductive transfer learning is related to the domain adaptation task from definition 2. An example of transductive transfer learning method is given in [87]. In this, they cluster the unlabeled target data for pseudo label generation and use a custom dimensionality reduction with the clustered target data and the labeled source data to create a subspace. Summarizing, because the two domains are different, transductive transfer learning aims to improve the learning by using the source data and task in combination with a few unlabeled target data.

### 3.4.3 Unsupervised-Transfer Learning

The last setting, the unsupervised transfer learning, which obviously is not part of the category supervised learning, will be described for completeness. It assumes that in both domains no labeled data is available. Furthermore, the task of the source and target data are different. In comparison with the supervised transfer learning methods, the goal is not to find labels for the target data but solving clustering, dimensionality reduction and density estimation.[66] Finally, we can give the following definition.

**Definition 5 (Unsupervised Transfer Learning [66])** *Given a source domain  $\mathcal{Z}$  with a learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and a corresponding learning task  $\mathcal{T}_{\mathcal{X}}$ , unsupervised transfer learning aims to help improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , when  $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$  and  $\mathcal{Y}_{\mathcal{Z}}$  and  $\mathcal{Y}_{\mathcal{X}}$  are not observable.*

Note that the predictive function  $f_{\mathcal{X}}(\cdot)$  aims, for example, to find the cluster centers. An example for Unsupervised transfer learning methods can be given from section 3.5.2, when the corresponding learner will be replaced with an unsupervised learning algorithm. Examples are discussed in section 3.5.2. Summarizing, this setting aims to improve the respective tasks by the use of the source data or task. This is, in fact, similar to the inductive transfer learning setting from section 3.4.1.[66]

## 3.5 Homogeneous Transfer Learning

In this section, the homogeneous transfer learning approach will be introduced. In general, it has the assumption that the feature spaces are equal and the conditional or marginal distributions of the domains are different. There are three main goals of homogeneous transfer learning. The first one is to align the marginal distribution difference. The second one is to correct the difference in the conditional probability distributions, and the last one tries to align both, the marginal and conditional distribution difference.[88, p. 6] This is formalized in the same way as the previous definition, which is influenced by the formulation of Pan et al. from [66].

**Definition 6 (Homogeneous Transfer Learning)** *Given a source domain  $\mathcal{Z}$  and learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and learning task  $\mathcal{T}_{\mathcal{X}}$ , homogeneous transfer learning aims to help improve the learning of the target predictive function  $f_{\mathcal{T}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , where  $\mathcal{F}_{\mathcal{Z}} = \mathcal{F}_{\mathcal{X}}$  and  $\mathcal{Y}_{\mathcal{Z}} = \mathcal{Y}_{\mathcal{X}}$ , but  $P(\mathbf{Z}) \neq P(\mathbf{X})$  and/or  $P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z}) \neq P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$ .[88, p. 4]*

If the differences only exist in the marginal distribution, then we can build another relation to Domain Adaptation from definition 2. The homogeneous transfer learn-

ing category can be divided in five general approaches: Instance-transfer, symmetric-feature transfer, asymmetric-feature transfer, parameter-transfer and relational-knowledge transfer. These approaches are sometimes called categories of information-transfer. They give a general understanding of how the methods are trying to align the differences.[88, p. 6-7]

Unlike the previous definitions, settings and explanations, which describing the problem and only giving an abstract solution through the predictive function  $f_{\mathcal{X}}(\cdot)$ .

### 3.5.1 Instance-Transfer

A instance-transfer approach tries to fulfill the goal of aligning the marginal distribution by re-weighting some source data. This re-weighted data is then directly used with target data for training. It seems that these type of algorithm works best when the conditional probability is the same in source and target domain [88, p. 6] In the following, we will present such an instance-transfer algorithm. The so-called *TrAdaBoost* which is proposed by Dai et al.[66]

#### TrAdaBoost

The TrAdaBoost<sup>3</sup> algorithm extends the boosting-based learning algorithm adaBoost. The algorithm follows the instance transfer definition and assumes that the marginal probability distributions are different, but the conditional probability distributions are the same.[22] It is originally proposed as two class solution, but there are extensions like MsTrAdaBoost for multi-class problems.[42]

The main goal is to reuse some source data in the target domain. The source data is, in general, considered as out-dated. It uses a small amount of labeled target data, considered as new data, which is called same-distribution training data. Therefore, and because of definition 3, the method has an inductive setting. This new data is then used to help to vote on the 'usefulness' of the old training data.[22]

The filtering boosts the part of data which is not too far away from the same-distribution training data. The training data is called diff-distribution-training data. The voted part of the data is used to make the knowledge transfer. Furthermore, the same-distribution training data and the diff-distribution training data are forming the new source data for the training. In the corresponding article, the weighted-SVM is used as the underlying learner. It is rather crucial that the learner has some weighting function because the filtering is done via weighting.[22]

The TrAdaBoost algorithm has two free parameters, which is the number of iterations and the initial value of the weight vector. Suppose the number of fixed

---

<sup>3</sup><https://github.com/LinZhineng/transfer-learning/tree/master/tradaboost>

iterations is  $T$  and the unlabeled target dataset  $\mathbf{X}$ . Consider  $\mathbf{Z}_D$  as diff-distribution training dataset with labels of size  $N$  and the same-distribution data (from the target data) with labels  $\mathbf{Z}_S$  with size  $M_S$ . This forms the complete training set  $\mathbf{Z} = \mathbf{Z}_D \cup \mathbf{Z}_S$ , with  $N + M$  entries and  $\mathbf{Z} = \{(\mathbf{z}_i, y_i)\}$ , where  $y_i$  is the label for the  $i$ th-data point with  $y \in \{0, 1\}$  and  $\mathbf{z}_i$  with:[22]

$$\mathbf{z}_i = \begin{cases} \mathbf{z}_i^D, & i = 1, \dots, N \\ \mathbf{z}_i^S, & i = N + 1, \dots, N + M_S \end{cases}$$

In every iteration, the learner is trained with this training set and predicts labels for the training  $\mathbf{Z}$  and the unlabeled target dataset  $\mathbf{X}$ . Note that the label for  $\mathbf{Z}_S$  is predicted, too.[22]

At this point the filtering takes place: If a diff-distribution data point is mistakenly predicted, then it is caused by the difference of this instance to the same-distribution training data. When this is the case, the *TrAdaBoost* algorithm reduces the weight for this data point, and therefore this point will affect the learning process less than in the current iteration. This is done via updating the weight vector  $\mathbf{w}^t = \{w_1^t, \dots, w_{N+M}^t\}$  in the  $t$ -th iteration with:[22]

$$w_i^t + 1 = \begin{cases} w_i^t \beta^{|h_t(\mathbf{z}_i) - y_i|}, & 1 \leq i \leq N \\ w_i^t \beta^{-|h_t(\mathbf{z}_i) - y_i|}, & N + 1 \leq i \leq N + M \end{cases} \quad (3.5)$$

Where  $h_t(\mathbf{x}_i)$  is the predicted label for the  $i$ -th data point. Furthermore, the different betas are set to  $\beta_t = \epsilon_t / (1 - \epsilon_t)$ , where  $\epsilon_t$  is a weighted error function and  $\beta = 1 / (1 + \sqrt{2 \ln n / T})$ . Note that the  $\epsilon_t$  function value is must be less than 0.5.

The  $\beta$ , which is multiplied with the weight for the training data, is bound by  $\beta^{|h_t(\mathbf{x}_i) - y_i|} \in (0, 1]$ . After the  $T$  iteration, the algorithm creates the final hypothesis of the labels based on the previous iterations. Note that only steps which are in the range of  $\{[T/2], \dots, T\}$  are considered.[22]

The error bound of *TrAdaBoost* is similar to the underlying *AdaBoost*. Furthermore, the error rate of the same-distribution data, for the final hypothesis (final label assignment based on the  $T$  iterations), can be upper bounded with:

$$\epsilon \leq 2^{\lceil T/2 \rceil} \prod_{t=\lceil T/2 \rceil}^T \sqrt{\epsilon_t (1 - \epsilon_t)} \quad (3.6)$$

At this point, the required  $\epsilon_t < 0.5$  becomes important. If so, then the final error will be increasingly smaller after each iteration.[22]

### 3.5.2 Symmetric-Feature-Transfer

The solutions, which implement the symmetric-feature transfer are trying to find a common latent subspace for source and target domain, with the goal to reduce

the marginal distribution differences. In the subspace, they should preserve the underlying structure of the data, which can be any relation in between the data. An example of a symmetric feature space transfer method is the Transfer Component Analysis (TCA), which will be discussed in the following.[88, p. 6]

### Transfer Component Analysis

The TCA method is presented by Pan et al. in [65]<sup>4</sup>. The authors proposed TCA to support classifier for unsupervised and semi-supervised learning.[65] However, the resulting kernel and the transformation matrix can be used to train a kernel based learner like the SVM, which is in fact supervised. Note that because it uses unlabeled target data it has a transductive setting.

It has the assumption, which makes it a homogeneous transfer method that the feature space is the same, the conditional probabilities are similar, but the marginal probability distribution is different. Furthermore, they assume that there exists a feature transformation  $\phi$ , which can align the differences, i.e.  $P(\phi(\mathbf{Z})) \approx P(\phi(\mathbf{X}))$  and preserve  $P(\mathbf{y}_Z|\phi(\mathbf{Z})) \approx P(\mathbf{y}_X|\phi(\mathbf{X}))$ .[65]

In the following, we will see that TCA requires target data in the training process and is therefore transductive.

In general, they assume that there exists underlying data structures and relations between them. Some of them do cause the distribution difference and some of them are a reason for relations between the source and target domain. The part of the data which causes the relation is called *transfer component* and should be found to preserve structure but at the same time aligning the differences.[65]

Consider  $\mathbf{K}$  as cross-domain kernel, as a composition of the source and target domain, in the embedded subspace, which has the form:[65]

$$\mathbf{K} = \begin{bmatrix} K_Z & K_{ZX} \\ K_{XZ} & K_{XX} \end{bmatrix} \quad (3.7)$$

Where  $\mathbf{K} \in \mathbb{R}$  with size  $K \times K = (N + M) \times (N + M)$ . This is the kernel over all data points and should be learned in a way that the marginal distribution in the subspace is aligned and the data variance (preserve data structure) is maximized. For that, the MMD from section 3.3.1 is rewritten as  $MMD = \text{tr}(\mathbf{KM})$ . Where  $\mathbf{M}$  is the MMD matrix as:[65]

$$(M)_{ij} = \begin{cases} \frac{1}{N^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathbf{Z} \\ \frac{1}{M^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X} \\ \frac{-1}{NM}, & \text{otherwise} \end{cases} \quad (3.8)$$

---

<sup>4</sup><https://github.com/LinZhineng/transfer-learning/tree/master/tca>

This is, in fact, the same transformation matrix for integrating MMD as it is done by an asymmetric approach described in section 3.5.3. The objective function of TCA uses the Maximum Mean Discrepancy Embedding (MMDE), from Pans previous work from[64]. The following optimization problem is designed to align the differences of the distributions but maximizes the subspace variance:[65]

$$\begin{aligned} \min_{\mathbf{W}} \quad & tr(\mathbf{W}^T \mathbf{K} \mathbf{M} \mathbf{K} \mathbf{W}) + \lambda tr(\mathbf{W}^T \mathbf{W}) \\ \text{s.t.} \quad & \mathbf{W}^T \mathbf{K} \mathbf{H} \mathbf{K} \mathbf{W} = \mathbf{I}_{K \times K} \end{aligned} \quad (3.9)$$

With this, the MMDE is simply the extension of the Maximum Mean Discrepancy with a regularization term, which can be seen on the right side of equation (3.9).[65] The parameter  $\lambda \geq 0$  is a trade-off parameter and the matrix  $\mathbf{W} \in \mathbb{R}^{N \times L}$  does the subspace transformation.[65],

Note the centering matrix  $\mathbf{H} = \mathbf{I}_{K \times K} - (1/K)\mathbf{1}_{(K \times K)}$  in the constraints. Because of the centering transformation, the maximized variance matrix  $\mathbf{W}^T \mathbf{K} \mathbf{H} \mathbf{K} \mathbf{W}$  should be persevered as identity matrix.

However, this seems that the kernel from equation (3.7) is expensive to calculate via a Semi-Definite Program (SDP).[65]

An SDP is an extension of a linear optimization problem, where the non-negative constrained is replaced with  $\mathbf{K} \succeq 0$ , which is the PSD attribute of a matrix. Additionally, the vector space for linear problems  $\mathbb{R}^D$  is replaced with the vector space of symmetric  $K \times K$  matrices based on the linear transformation of matrices.[31]

For determining  $\mathbf{W}$ , they are ended up by formulating the optimization problem as an eigenvalue problem. By finding the largest  $L$  eigenvectors with  $(\mathbf{K} \mathbf{M} \mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{K} \mathbf{H} \mathbf{K}$ , the subspace kernel  $\mathbf{K}$  can be calculated, or the source and target data can be reduced with  $\mathbf{W} \in \mathbb{R}^{K \times L}$ . The computational complexity is given by  $\mathcal{O}(L(N + M)^2)$ , where  $L$  is the number of eigenvectors.[65] Note that the number of dimensions is not considered in the computational complexity.

### Geodesic Flow Kernel

Another symmetric feature transfer method is the Geodesic Flow Kernel (GFK). It is proposed by Gong et al. in [35]<sup>5</sup>. Following the definition of homogeneous methods and similar to TCA, the Geodesic Flow Kernel is trying to lower the marginal distribution differences.[88, p. 13]

Furthermore, it is claimed as unsupervised transfer learning method, however because the result of this method are, in fact, a kernel, it can be trained with a kernel machine.[35] Again because it needs unlabeled target data in the learning process, it will be considered as transductive transfer learning method.

The idea of GFK can be summarized in the following steps: The source and the

<sup>5</sup>[http://www-scf.usc.edu/~boqinggo/domain\\_adaptation/GFK\\_v1.zip](http://www-scf.usc.edu/~boqinggo/domain_adaptation/GFK_v1.zip)



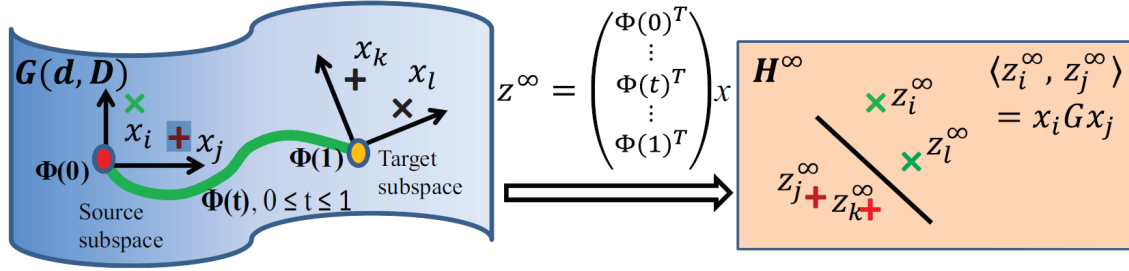


Figure 3.3: The geodesic flow with the start and endpoints from source and target. The green line shows the geodesic flow with the feature transformation. The infinite vectors and the induced kernel is shown on the right side.[35]

target data has to be embedded in a Grassmannian manifold. Consider a subspace  $\mathbf{P} \in \mathbb{R}^{K \times L}$  with  $K$  observations and  $L$  is the feature space dimension. Then the collection of the  $L$  dimensional subspaces are forming the Grassmannian  $\mathbb{G}(L, D)$  in the original  $D$  dimensional feature space.[35]

Next, construct a geodesic flow and merge source (start) and target (end) data as points on the flow. Proceed by integrating an infinite number of subspaces from start to end along the flow  $\Phi(t)$ .[35]

After that, the raw features  $\mathbf{x}$  are projected into the infinite subspaces, which makes them infinite dimensional feature vectors  $\mathbf{z}^\infty \in \mathcal{H}^\infty$ . The inner product will be applied over the vectors  $\mathbf{z}^\infty$  to define a kernel function, which can be computed in a finite subspace (kernel trick). The kernel in the space  $\mathcal{H}^\infty$  should then encapsulate the underlying difference and commonness of the source and target data.[35] The approach is summarized in figure 3.3

For more details please have a look in the thesis [34, p. 45;110-113] and the article [35].

The computational complexity is obtained by analyzing the source code and the article. The GFK algorithm computes the PCA for the whole data matrix  $K$ . The PCA has the time complexity of  $\mathcal{O}(DK^2L)$ , where  $D$  is the dimensions of the data and  $L$  as the number of the largest eigenvalues.[8, p. 563] We assume that the required SVD of the whole data matrix has a complexity of  $\mathcal{O}(DK^2)$ . The complexity of calculating the null space of a training set (start point) is given with the complexity of SVD, because the MatLab function<sup>6</sup>, for calculating the null space is based on SVD. Therefore, the overall complexity of the GFK algorithm is  $\mathcal{O}(DLK^2 + DK^2)$ .

### 3.5.3 Asymmetric-Feature-Transfer

An asymmetric feature transfer learning approach tries to transform the source domain data in the target (subspace) domain. This should be done in a way that

<sup>6</sup><https://de.mathworks.com/help/matlab/ref/null.html>

the transformed source data will match the target distribution. In comparison to the symmetric feature transfer approaches, there will be no shared subspace, but only the target space. In the following, the Joint Domain Adaption (JDA) algorithm is described, to give an idea of an asymmetric approach.[88, p. 6; 10]

### Joint Distribution Adaptation

The Joint Domain Adaption algorithm tries not only to solve the marginal probability distribution but also the conditional probability distribution. This solution is proposed by Long et al. in [56].<sup>7</sup>

Because it solves the probabilistic differences between the domains, it can be considered as transductive setting, with the need for unlabeled target data.

The primary goal is to alter the joint distribution by a feature transformation in a way that the joint expectations of the source and target features are matched between the domains.[56]

They approximate it by searching an adaptation matrix  $\mathbf{A} \in \mathbb{R}^{L \times N}$ , which can transform any new data in the (target) subspace and is, in fact, an orthogonal transformation matrix. Furthermore, the embedded data matrix  $\mathbf{Z} \in \mathbb{R}^{M \times L}$  is searched to transform the already existing data in the subspace.[56]

This approach consists of four steps, which are feature transformation, marginal distribution adaptation, conditional distribution adaptation and solving the resulting optimization problem.[56]

Revisiting our transfer learning definition, source and target data can be represented as data matrix  $\mathbf{T} = [\mathbf{Z}; \mathbf{X}] \in \mathbb{R}^{K \times D}$  where  $K = N + M$ , with the number of source and target data respectively. In fact, the optimization problem to solve the marginal differences is very similar to the TCA solution from equation (3.9).[65]

It seems that adopting the conditional probability distribution is kind of bigger problem. Since the goal is to be able to solve the transfer problem without labeled target data  $P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$  cannot be directly modeled or estimated with some statistics like it is done to align the marginal distributions.[56]

Because of this, they applied a pseudo label technique. This is done by using a base classifier, e. g., SVM to predict the not 'final' class labels for the target domain, which are the pseudo labels. Since  $P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$  is unknown, the class conditional probability functions in form of  $P(\mathbf{X}|\mathbf{y}_{\mathcal{X}} = c)$ (true class labels) and  $P(\mathbf{X}|\mathbf{y}_{\mathcal{X}} = c)$ (pseudo labels) are matched. Here  $c$  represents a class of the label space  $\mathcal{Y}$ , i. e.  $c \in \{1, \dots, C\}$  for  $C$  labels.[56]

By incorporating the class conditional distribution in the Maximum Mean Discrepancy, not the sample means, but the distance between the class conditional distri-

---

<sup>7</sup><http://ise.thss.tsinghua.edu.cn/~mlong/doc/joint-distribution-adaptation-iccv13.zip>

butions are measured:[56]

$$\left\| \frac{1}{N^{(c)}} \sum_{\mathbf{t}_i \in \mathbf{Z}^{(c)}} \mathbf{A} \mathbf{t}_i^T - \frac{1}{M^{(c)}} \sum_{\mathbf{t}_j \in \mathbf{X}^{(c)}} \mathbf{A} \mathbf{t}_j^T \right\|^2 = \min_{\mathbf{A}^T \mathbf{A}} \text{tr}(\mathbf{A} \mathbf{T}^T \mathbf{M}_c \mathbf{T} \mathbf{A}^T) \quad (3.10)$$

With  $\mathbf{Z}^{(c)}$  as the set of examples belonging to class  $c$  in the source data, where the points have the true class labels and  $N^{(c)} = |\mathbf{Z}^{(c)}|$  is the size of the set. The same applies to the target domain but with a pseudo label set  $\mathbf{X}^{(c)}$  and size  $M^{(c)} = |\mathbf{X}^{(c)}|$ . The elements of  $\mathbf{M}_c$  with the use of the class labels are computed by: [56]

$$(M_c)_{ij} = \begin{cases} \frac{1}{N^{(c)} N^{(c)}}, & \mathbf{t}_i, \mathbf{t}_j \in \mathbf{Z}^{(c)} \\ \frac{1}{M^{(c)} M^{(c)}}, & \mathbf{t}_i, \mathbf{t}_j \in \mathbf{X}^{(c)} \\ \frac{-1}{N^{(c)} M^{(c)}}, & \begin{cases} \mathbf{t}_i \in \mathbf{Z}^{(c)}, \mathbf{t}_j \in \mathbf{X}^{(c)} \\ \mathbf{t}_j \in \mathbf{Z}^{(c)}, \mathbf{t}_i \in \mathbf{X}^{(c)} \end{cases} \\ 0, & \text{otherwise} \end{cases} \quad (3.11)$$

Note that because of the differences in the domains the, most pseudo labels are incorrect, but it seems that in combination with the modified MMD, it helps to draw the distribution closer together in the subspace.[56]

By incorporating, the class conditional MMD, the final JDA optimization problem is given with:[56]

$$\min_{\mathbf{A}^T \mathbf{M}_c \mathbf{T} \mathbf{A}^T = \mathbf{I}} \sum_{c=0}^C \text{tr}(\mathbf{A} \mathbf{T}^T \mathbf{M}_c \mathbf{T} \mathbf{A}^T) + \lambda \|\mathbf{A}\|_F^2 \quad (3.12)$$

Where  $\lambda$  is again the regulation parameter to guarantee a distinct function for the optimization problem.

By analyzing this problem, they showed that solving equation (3.12) is equal to solve the eigenvalue problem of  $(\mathbf{T}^T \sum_{c=0}^C \mathbf{M}_c \mathbf{T} + \lambda \mathbf{I}) = \mathbf{T}^T \mathbf{H} \mathbf{T} \mathbf{A}^T \mathbf{\Phi}$  where  $\mathbf{\Phi} = \text{diag}(\phi_1, \dots, \phi_N)$  are Lagrange multiplier. By finding the  $L$  smallest eigenvectors for the eigenproblem, the transformation matrix  $\mathbf{A}$  can be obtained. The embedded matrix  $\mathbf{Z}$  can be obtained with  $\mathbf{Z} = \mathbf{A} \mathbf{T}^T$ . [56]

For details of the algorithm, please refer to [56].

The computational complexity of JDA for  $T \leq 50$  as the number of iterations until convergence and the subspace bases with  $L \leq 500$  is given by  $\mathcal{O}(TLM^2 + TCN^2 + TMN)$ . [56] Note that the number of dimensions is not considered in the computational complexity.

### 3.5.4 Parameter-Transfer

The third approach, the parameter-transfer aims to transfer knowledge through shared parameters from source and target domains. This is achieved by creating

multiple source models, combining them and reweights them to create an improved target learner. One approach in this category is the Multi-Model Knowledge Transfer (MMKT).[88, p. 7-8]

### Multi Model Knowledge Transfer

The MMKT approach is proposed by Tommasi et al. in [83]. They are using the Least-Square Support Vector Machine (LS-SVM) in combination with a discriminative method to transfer knowledge from previously learned models based on the source data and transferring it to the target data. Note that the term models may confusing, which we will see in the following. Furthermore, they want a classifier, that can be trained only on a few training examples and is able to learn more from less data than the standard LS-SVM does. The transfer knowledge method is adopted from his previous work in [82].

Note that this method does not try to explicit minimize the distribution differences, as the definition of the homogeneous transfer learning from definition 6 states. Consider a labeled source dataset with  $\{\mathbf{z}, y_{zi}\}_{i=1}^N$ , where  $y_{zi} \in \mathcal{Y}_Z = \{1, -1\}$ . Furthermore, the feature mapping function  $\phi(\cdot)$ , which maps the features in the infinite space and the inducing kernel function  $\mathbf{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$ . Suppose that we have prior knowledge from  $E$  classes (not models, but in following we will call it models), which are already trained. This knowledge is represented with the hyperplane parameter, i.e.,  $\mathbf{w}'_j, j = \{1, \dots, E\}$ , for each class.[83]

With that Tommasi et al. assuming that the transfer happens when the constraints of the hyperplane of the new class  $E + 1$  are set in a manner that it is close to the already trained classes of  $E$ . Keeping this in mind, the optimization problem for the LS-SVM can be extended to a model for multi knowledge transfer:[83]

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w} - \sum_{j=1}^E \beta_j \mathbf{w}'_j\|^2 + \frac{C}{2} \sum_{i=1}^N \zeta_i [y_i - \mathbf{w} \phi(\mathbf{x}_i) - b]^2 \quad (3.13)$$

Where  $\mathbf{w}$  is the searched model parameter and  $C$  as free cost parameter. With  $\mathbf{w}'_j$  as a parameter for the already trained model out of  $E$  models and  $\zeta_i$  as a parameter for weighting, which helps to balance the contribution of positive and negative examples in the training data in equation (3.13). It is determined with the following case:[83]

$$\zeta_i = \begin{cases} \frac{N}{2N^+}, & y_i = +1, \\ \frac{N}{2N^-}, & y_i = -1 \end{cases} \quad (3.14)$$

Where  $N^+$  and  $N^-$  representing the number of positive and negative examples of the source data, respectively. Furthermore, the optimal solutions for  $\mathbf{w}$  is now found with:[83]

$$\mathbf{w} = \sum_{j=1}^E \beta_j \mathbf{w}'_j + \sum_{i=1}^N \alpha_i \phi(\mathbf{x}_i) \quad (3.15)$$

In this,  $\mathbf{w}_j'$  is weighted through  $\beta_j$ . Because of the reformulated optimization problem the optimal determination of  $\boldsymbol{\alpha}$  and  $b$  can be expressed as:[83]

$$\begin{bmatrix} \mathbf{K} + \frac{1}{C}\mathbf{W} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} \quad (3.16)$$

Where  $\mathbf{W} = \text{diag}\{\zeta_1^{-1}, \dots, \zeta_N^{-1}\}$  with  $\zeta_i$  as defined in (3.14).

The  $\beta_j \in (0, 1)$  parameter forms  $\boldsymbol{\beta} = \{\beta_1, \dots, \beta_E\}$  for controlling the degree of closeness of the new model to the  $j$ -th model. Note that  $\boldsymbol{\beta}$  has to be chosen in the unitary ball and therefore has to be  $\|\boldsymbol{\beta}\|_2 \leq 1$  (in the  $L_2$ -space). This term is necessary to avoid overfitting problems, which happens when the number of models in comparison with the training data is large. Furthermore, it is possible with the new model to find the optimal classes for knowledge transfer automatically. Hence, an optimal value for  $\boldsymbol{\beta}$  must be found.[83]

In this method the error of the LS-SVM is quantified with the Leave One Out (LOO)-Error.[83]

The 'classical' approach of LOO-Error, as the name implies, leaves one data point out and creates a new 'dataset' on which a model is trained. After that the error is calculated and is repeated as long as not all data points are already leaved out. From the resulting errors the mean is calculated, this forms the Leave One Out-Error.[28, p.74-76]

They have incorporated this error with the LS-SVM optimization problem and obtained the loss function for the new approach:[83]

$$\text{loss}(y_i, \tilde{y}_i) = \zeta_i \max[1 - y_i \tilde{y}_i, 0] = \max \left[ y_i \zeta_i \left( \mathbf{G}_{ii}^{-1} - \sum_{j=1}^k \beta_j \frac{\alpha'_{i(j)}}{\mathbf{G}_{ii}^{-1}} \right), 0 \right] \quad (3.17)$$

Note that  $\mathbf{G}_{ii}$  in this context means, the dataset without the  $i$ -th data point is applied. Furthermore,  $\tilde{y}_i$  are the LOO predictions and  $\alpha_i = \mathbf{G}_{ii}^{-1} = [\hat{y}_1^j, \dots, \hat{y}_{i-1}^j, \hat{y}_{i+1}^j, \dots, \hat{y}_N^j, 0]^T$ , with  $\hat{y}_i = \mathbf{w}_j' + \phi(x_i)$ .

With that the convex 'global' loss function over all models can be formulated with respect to the constrain  $\|\boldsymbol{\beta}\|_2 \leq 1$ . This will find optimal  $\beta$ 's. In practice, the optimization process is implemented with a project sub-gradient descent algorithm.[83] The computational complexity of the algorithm is  $\mathcal{O}(N^3 + EN^2)$ , with  $N$  training data points and  $E$  classes, i. e., prior models.

### 3.5.5 Relational-Knowledge-Transfer

The relational knowledge transfer aim to find some relationship between the source and target data.[88, p. 7] Because the Transfer-Kernel learning algorithm, from section 4.2, finds this relationship in the eigenvalues of source and training data, we assign this category to the method. The details of the algorithm are discussed in

the corresponding chapter, because the Probabilistic Classification Transfer Kernel Vector Machine is based on it.

## 3.6 Heterogeneous Transfer Learning

In this section, we discuss the heterogeneous transfer learning category. In contrast to the homogeneous transfer learning, in general, it has the assumption that the marginal distributions of the feature spaces are equal, but the feature spaces or the label space are not equal.[88, p. 4]

Keeping this in mind, the goal of the heterogeneous transfer learning methods is to align the differences in the source and target feature space. If the marginal distribution of the domains is not the same, then further aligning steps via domain adaptation or homogeneous transfer must be done. [88, p. 6]

We can again give a formal definition with:

**Definition 7 (Heterogeneous Transfer Learning)** *Given a source domain  $\mathcal{Z}$  and learning task  $\mathcal{T}_{\mathcal{Z}}$ , a target domain  $\mathcal{X}$  and learning task  $\mathcal{T}_{\mathcal{X}}$ , heterogeneous transfer learning aims to help improve the learning of the target predictive function  $f_{\mathcal{X}}(\cdot)$  in  $\mathcal{X}$  using the knowledge in  $\mathcal{Z}$  and  $\mathcal{T}_{\mathcal{Z}}$ , where  $P(\mathbf{Z}) = P(\mathbf{X})$ , but  $\mathcal{F}_{\mathcal{Z}} \neq \mathcal{F}_{\mathcal{X}}$  and/or  $\mathcal{Y}_{\mathcal{Z}} \neq \mathcal{Y}_{\mathcal{X}}$ . [88, p. 4]*

Because the examples in section 3.1, do not cover the heterogeneous definition completely, we will give an example in the following. Imagine a machine learning application of software module defect classification, where a learner is trained to predict whether a software module works well or is defect prone. Furthermore, there exists a metric for every software module, which measures, e. g., some performance criteria. This forms the feature space. If the model is trained on one software with the metric  $\mathcal{F}_{\mathcal{Z}}$ , which has e. g., ten metrics and the prediction should be done for a different software which is reviewed on metric  $\mathcal{F}_{\mathcal{X}}$  and has, for example, six metrics, then it is a heterogeneous transfer problem.[88, p 3-4]

However, because this work focuses on supervised homogeneous transfer, we will not go into more detail about the heterogeneous transfer categories, but present an example in the following to get an idea of how it works.

### Text to Image Transfer

The TTI method is presented by Qi et al. in [68]. It is considered as a symmetric transfer, and it can be regarded as domain depended for image classification with a text extension and is therefore no general solution.[88, p. 22] This is caused by the fact that Text to Image (TTI) transforms both the text data (source domain) and

image data (target domain) into a latent feature space called *topic space*, as we will see in the following.

Note that in the previous transfer learning methods the source and target domains, were separated by learning the source domain and evaluating at the target domain. This is not the case with TTI, because source and target are just symbolic for two domains, which should be merged (transformed), to improve the accuracy in the target domain. However, the method requires target domain data.[68]

In general, image classification suffers from a few problems. For example, the semantic gap between image datasets and labels are often higher than by text sets. This is caused by the fact that text features to labels are often better to interpret in comparison with image features to labels.[68]

There are various websites and social networks, which have images linked to text. This text can be used to increase the amount of target data (images), if this text can be transformed and used for image classification. They are using these co-occurrences (images linked to a text; text linked to category) to build a semantic bridge from text to images. This bridge can further be used to transfer semantic of text into images.[68]

Revisiting the previous notation from 3.2, we can formulate the source (text) domain as  $\mathcal{Z}$ , where the samples are  $\mathbf{Z} \in \mathbb{R}^{N \times A}$  or a single document as  $\mathbf{z}_i \in \mathbb{R}^A$ , with  $A$  as a number of feature space dimensions. Furthermore, the target (image) domain as  $\mathcal{X}$ , with sample matrix  $\mathbf{X} \in \mathbb{R}^{M \times B}$  or a single image with  $\mathbf{x}_i \in \mathbb{R}^B$ . The authors assume that the size of images set is much smaller than the size of the text set.[68] The linkage through the co-occurrence of text and image is then formalized with  $\mathcal{C} = \{(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l, c(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l))\}$ . This means that  $\bar{\mathbf{z}}_k$  is the corresponding text feature vector to an image feature vector  $\bar{\mathbf{x}}_l$ . Note that in the next,  $c(\cdot)$  is abbreviated with  $c_{k,l}$ . Through the linkage function, they build a semantic bridge from text to image.[68] The so-called *translator* function with  $T : \mathbb{R}^A \times \mathbb{R}^B \rightarrow \mathbb{R}$  weights the linkage strength between a document and image. Now the label for an image is found with:[68]

$$f_T(\mathbf{x}) = \sum_{i=1}^N y_i T(\mathbf{z}_i, \mathbf{x}) \quad (3.18)$$

Where  $y_i$  is the text document label, and  $f_T(\mathbf{x})$  provides the label depending on the sign.

The objective proposed by Qi et al. is to learn the function  $T(\cdot)$ , which maximizes the classification accuracy or rather minimizes a loss function. However, they determine  $T(\cdot)$  by finding a transformation matrix to transfer text and images in the  $L$  dimensional topic space. This means that they aggregate the data into the least  $L$  topics, where the linkage strength of image and text co-occurrence is weighted. The extended  $T(\cdot)$  function has the form of:[68]

$$T(\mathbf{z}_i \mathbf{x}_j) = \langle \mathbf{W}_{\mathcal{Z}} \mathbf{z}_i, \mathbf{W}_{\mathcal{X}} \mathbf{x}_j \rangle = \mathbf{z}_i \mathbf{W}_{\mathcal{Z}} \mathbf{W}_{\mathcal{X}}^T \mathbf{x}_j^T = \mathbf{z}_i \mathbf{S} \mathbf{x}_j^T \quad (3.19)$$

With  $\mathbf{S} = \mathbf{W}_Z \mathbf{W}_X^T$  and size  $A \times B$  and  $\mathbf{W}_Z$  and  $\mathbf{W}_X$  are the transformation matrices respectively. In the following, it is sufficient just to learn  $\mathbf{S}$  rather than the transformation matrices and has to be determined in order to minimize the loss function for the prediction accuracy. This is done by the following minimization problem:[68]

$$\min_{\mathbf{S}} \gamma \sum_{j=1}^{N_T} l(y_j \cdot f_T(\mathbf{x}_j)) + \lambda \sum_C \chi(c_{k,l} \cdot T(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l)) + \|\mathbf{S}\|_{\Sigma} \quad (3.20)$$

Note that  $y_j$ , in this case, is the image label. The parameter  $\gamma$  and  $\lambda$  controlling the relative importance of the auxiliary data and the co-occurrence data respectively. Furthermore,  $C$  is the number of weighted co-occurrences.

With equation (3.20) the general performance of the translation process is measured and can be divided into three parts.[68]

The first is the empirical loss of the prediction, based on the function  $f_T$ . For the loss, they use the logistic loss function with  $l = \log\{1 + \exp(-z)\}$ .

The second is the sum over the  $C$  weighted co-occurrences  $c_{k,l}$  with a monotonic exponential decreasing function  $\chi = \exp(-z)$ . They pretend with this choice, the optimization problem is closed-form and easy to optimize.

The last term is for regulating the learning of the translator, for an improved generalization ability. They are incorporating the Frobenius norm, which can be found here [59], and the trace norm, for example, defined in [71], to formulate the regularization term:[68]

$$\|\mathbf{S}\|_{\Sigma} = \inf_{\mathbf{S}=\mathbf{W}_Z \mathbf{W}_X^T} \frac{1}{2} \left( \|\mathbf{W}_Z\|_F^2 + \|\mathbf{W}_X\|_F^2 \right) = \Omega(T) \quad (3.21)$$

Where  $\|\cdot\|_F^2$  is the Frobenius norm and  $\|\mathbf{S}\|_{\Sigma}$  the trace norm.

With that they found that the trace norm of  $\mathbf{S}$  is a convex function and can be computed by a Semi-Definite Program. They argue that by minimizing equation (3.20), the resulting  $\mathbf{S}$  will give a minimum of topics, used to explain the correspondence between text and image. For finding the optimum, they used a proximal gradient method to approximate the optimal solution. For further information about the algorithm, the reader may consider [68].

### 3.7 Negative Transfer

A major key assumption of transfer learning is that the source and target domains are related to each other. This is pointed out in the examples from section 3.1.

If this assumption, however, does not hold and the domains may not related, then the transfer learning solution can impact the performance of the underlying classifier in a negative way.[88, p. 29]

The problem is that all methods take the assumption that a relation exists, but there is no current way to determine whether a relation exists or not.[77, p. 1672] This is



also shown in the dataset description in chapter 5, where the datasets are designed in a way that the domains are related to each other. However, as pointed out in the chapter, this selection of datasets is common in the transfer learning community and may not consider the fact of negative transfer. However, the negative transfer is a phenomenon that exists, which is seen in the results of sections 5.5 and 5.6, where the transfer learning methods are worse than the baseline classifier.

## 3.8 Conclusion

It is again worth mentioning that there are not only many relations among the definitions, but in the solutions too, refer to section 3.5. Weiss et al. pointed out that there are many inconsistencies in the definition of transfer learning.[88, p. 5-6] However, fair enough, many of the definitions are used to distinguish how the transfer is done. For example, is labeled target data or any target data at all needed, or how much source data has to be available, for the transfer. In our opinion, this is crucial information to know, because it changes on how datasets have to be prepared, to make a comparison among different solutions. Furthermore, one has to additional manual label target data before an inductive solution can be used. Therefore, the mentioned definitions of inductive transfer (3) and transductive transfer (4) are solid extensions to the homogeneous (3.5) and heterogeneous (3.6) transfer learning definitions.

In contrast to the mentioned survey papers of [88] and [66], we choose to dive into the transfer learning methods in detail, to get an idea of how the general problem is formulated. Furthermore, the process of how they get a solution for the problem is described, instead of giving a high-level description. This seems reasonable, because of the already pointed out relations between the solutions, which do not omit the need for further research, but give a generalized understanding on how the transfer problem can be solved.

Another point worth mentioning is that the provided solutions are proposed to support only certain learning approaches and varying between supervised, semi-supervised and unsupervised. However, as seen in this chapter, the output of a solution is often a kernel or transformation matrix, which can be used with any kernel-based algorithm. Nevertheless, to be careful, for example from [56], [57] [35] appears that their solutions can be easily applied to any standard classifier.

In practice, a classifier may has dependencies in his algorithm, and the integration is a more challenging problem. This might be the reason why in [35], they are using a custom implementation of  $k$ -nearest neighbor or that only the LibSVM, which supports pre-calculated kernels, is utilized in the corresponding articles. Therefore, the integration into a kernel machine may need further investigation.

# Chapter 4

## Integration of Transfer Learning

In this chapter, two versions of the Probabilistic Classification Transfer Kernel Vector Machine (PCTKVM) are proposed. The PCTKVM which handles the  $\theta$  as a fixed parameter and the PCTKVM $_{\theta_{\text{Est}}}$ , which does a theta estimation. The extensions 'TK' to original term PCVM should indicate the new kernel approach to address the transfer learning problem. As already mentioned in chapter 3 there are various solutions to solve the transfer learning problem. Revisiting section 3.5, the homogeneous transfer learning problem can be summarized in four approaches: Instance-Transfer, Symmetric-Feature-Transfer, Asymmetric-Feature-Transfer, and Relational-Knowledge-Transfer.

However, Long et al. proposed in [57], transfer kernel learning as another approach-category. Nevertheless, that the relational knowledge approach, transfers information through relationships between the source and target data, the TKL approach suits well in this category. Furthermore, because it needs target data for the learning process and according to the definition of transductive transfer learning, it can be categorized as transductive transfer setting.

The main idea of it is to approximate the kernel of the training set with the kernel of the test set via the Nyström kernel approximation.[57] On the other hand [73], proposed a PCVM with linear costs which is also achieved via the Nyström approximation. Therefore, it seems reasonable, to integrate another interpretation of the Nyström approximation into the PCVM.

Another advantage is that TKL is very simple when it comes to dataset usage. According to 3.2 there are transfer learning solutions which need pre-labeled test data. As we will see in the following, the TKL using only the training and an unlabeled test dataset to compute the transfer kernel and therefore has an 'easy' handling. However, before going into the approach itself, the underlying technique should be discussed. See section 5.2.2 for details to kernels and the Gram matrix.

## 4.1 Nyström Approximation

The main Problem by computing eigenvalues and eigenvectors is that optimal eigenvalue decomposition has a time complexity of  $\mathcal{O}(N^3)$  and alternative approaches must be found. The Nyström Method was originally made for solving the eigenvalue equation, which can be seen in equation (4.2).[94]

It is done by randomly selecting  $N$  observations (rows/columns) without replacement out of  $\mathbf{K}$  for calculating the approximation.[90] Considering a symmetric positive semi definite kernel  $k(\mathbf{z}, \mathbf{x})$  which satisfies Mercers Theorem, then  $\mathbf{K}$  can be represented as:[90]

$$k(\mathbf{z}, \mathbf{x}) = \sum_{i=1}^D \lambda_i \phi_i(\mathbf{z}) \phi_i(\mathbf{x}) \quad (4.1)$$

With  $D < \infty$  as the number of feature space dimensions and  $\lambda_1 \geq \lambda_2 \geq \dots \geq 0$  as the sorted eigenvalues and  $\phi_i$  as eigenfunctions. As consequence of Mercer follows that  $\lambda_i$  and  $\phi_i$  have to satisfy the eigenvalue equations:

$$\int_{\mathbf{z}} k(\mathbf{x}, \mathbf{z}) \phi_i(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \lambda_i \phi_i(\mathbf{x}) \quad (4.2)$$

With  $p(\mathbf{z})$  as the probability density of the input vector  $\mathbf{z}$ . Furthermore, the orthogonality conditions are satisfied with:[74, p. 59]

$$\int \phi_i(\mathbf{z}) \phi_j(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \delta \quad (4.3)$$

If a kernel as defined above is given, then an empirical estimate can be done by, drawing an identical and independently sample  $\mathbf{z}_1, \dots, \mathbf{z}_N$  from  $p(\mathbf{z})$ . The integral can be rewritten as an empirical estimate:[90]

$$\sum_{k=1}^N \frac{k(\mathbf{x}, \mathbf{z}_k) \phi_i(\mathbf{z}_k)}{N} \approx \lambda_i \phi_i(\mathbf{x}) \quad (4.4)$$

Where  $N$  is the number of sampled observation from the kernel and the  $N$  orthogonal eigenfunctions. If we consider the  $N$  columns from  $\mathbf{K} \in \mathbb{R}^{K \times K}$  as separate kernel matrix  $\mathbf{K}^{(N)}$  with size  $N \times N$  and elements  $K_{ij}^{(N)} = K(\mathbf{z}_i, \mathbf{z}_j)$  for  $i, j = \{1, \dots, N\}$ , then we can use this for the eigenvalue problem of the sample kernel:[90]

$$\mathbf{K}^{(N)} \mathbf{U}^{(N)} = \mathbf{U}^{(N)} \mathbf{\Lambda}^{(N)} \quad (4.5)$$

With  $\mathbf{U}^{(N)} \in \mathbb{R}^{N \times N}$  as eigenvectors and  $\mathbf{\Lambda}^{(N)}$  as a diagonal matrix with the eigenvalues for the small kernel with  $\lambda_1^{(N)} \geq \lambda_2^{(N)} \geq \dots \lambda_N^{(N)} \geq 0$ . If we consider points within the kernel  $K^{(N)}$ , then  $\mathbf{x}$  can be replaced with  $\mathbf{z}_j$ . Continuing with matching (4.4) against (4.5) then the following approximation can be obtained:[90]

$$\phi_i(\mathbf{z}_j) \approx \sqrt{N} U_{ij}^{(N)} \quad (4.6)$$

Finally, by using  $\phi_i(x_k)$  in (4.4) the approximation for any eigenvector or eigenvalue can be given by:

$$\phi_i(\mathbf{x}) \approx \frac{\sqrt{N}}{\lambda_i^{(N)}} \sum_{k=1}^N k(\mathbf{x}, \mathbf{z}_k) U_{k,i}^{(N)}, \quad \lambda_i \approx \frac{\lambda_i^{(N)}}{N} \quad (4.7)$$

With this, the  $i$ -th eigenfunctions and eigenvalues can be approximated by calculating the sample kernel.[90] As a consequence, the eigenfunctions and eigenvalues of the whole kernel can be approximated.

### 4.1.1 Nyström Kernel Learning

Williams and Seeger extended the work of Nyström to speed up kernel machines. The problem with kernel-based predictors, e.g., SVM, is similar to the eigenvalue decomposition problem from section 4.1 It requires the computational complexity of  $\mathcal{O}(N^3)$  in the worst case, to find a solution.[90]

With the Nyström approximation, the complexity for predictors can be bound by  $\mathcal{O}(M^3N)$  with using a smaller system with size  $M$ , where  $M < N$ .[90]

Now the goal is to create a low-rank approximation Matrix  $\tilde{\mathbf{K}}$  of the original  $N \times N$  kernel matrix  $\mathbf{K}$ . Revisiting section 4.1, by using the technique of Nyström, a kernel  $\mathbf{K}$  which satisfies Mercer's theorem is needed.[90] Furthermore, we draw  $M$  IID samples of  $\mathbf{K}$  and therefore we can rewrite the symmetric kernel according to:[62]

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{M,M} & \mathbf{K}_{M,N-M} \\ \mathbf{K}_{N-M,M} & \mathbf{K}_{N-M,N-M} \end{bmatrix} \quad (4.8)$$

Where  $\mathbf{K}_{M,M}$  is the sample matrix and  $\mathbf{K}_{N-M,M}$  is the crossdata-matrix of  $\mathbf{K}$  and the rest of the data  $\mathbf{K}_{N-M,N-M}$ . Because  $\mathbf{K}$  is symmetric  $\mathbf{K}_{N-M,M} = (\mathbf{K}_{M,N-M})^T$ . Furthermore,  $\mathbf{K}_{N-M,N-M}$  is the part of the kernel which is not sampled or 'used' for the approximation, as we will see.

Motivated by the general eigenvalue problem, which can be obtained from [39, p. 221] and shown in (4.5), a kernel can be approximated through another kernel  $\tilde{\mathbf{K}} = \tilde{\mathbf{U}} \tilde{\Lambda} \tilde{\mathbf{U}}^T = \sum_{i=1}^p \tilde{\lambda}_i^{(N)} \tilde{\mathbf{u}}_i^{(N)} (\tilde{\mathbf{u}}_i^{(N)})^T$ . [90] With the approximated eigenvalues and eigenvectors as  $\tilde{\lambda}_i^{(N)}$  and  $\tilde{\mathbf{u}}_i^{(N)}$  respectively.

Therefore, we proceed with the equations to approximate the eigenvalues and eigenvectors. By applying the equations of (4.6) and (4.7), the approximation is based on the eigenvalues and eigenvectors of the sample matrix and can be made with:[94]

$$\begin{aligned} \tilde{\lambda}_i^{(N)} &= \frac{N}{M} \lambda_i^{(M)} \quad \sim \quad \Lambda^{(N)} = \frac{N}{M} \Lambda^{(M)} \\ \tilde{\mathbf{u}}_i^{(N)} &= \sqrt{\frac{M}{N}} \frac{1}{\lambda_i^{(M)}} \mathbf{K}_{N,M} \mathbf{u}_i^{(M)} \quad \sim \quad \mathbf{U}^{(N)} = \sqrt{\frac{M}{N}} \mathbf{K}_{N,M} \mathbf{U}^{(M)} (\Lambda^{(N)})^{-1} \end{aligned} \quad (4.9)$$

With  $i = 1, \dots, N$  and  $u_i^{(M)}$  and  $\lambda_i^{(M)}$  from the eigenproblem (Eq. (4.5)). After obtaining the approximated eigenvalues and vectors, the full approximated kernel

can be restored:[90][94]

$$\tilde{\mathbf{K}} = \left( \sqrt{\frac{M}{N}} \mathbf{K}_{N,M} \mathbf{U}^{(M)} (\mathbf{\Lambda}^{(N)})^{-1} \right) \left( \frac{N}{M} \mathbf{\Lambda}^{(M)} \right) \left( \sqrt{\frac{M}{N}} \mathbf{K}_{N,M} \mathbf{U}^{(M)} (\mathbf{\Lambda}^{(N)})^{-1} \right)^T \quad (4.10)$$

With the use of  $\tilde{\mathbf{K}} = \tilde{\mathbf{U}} \tilde{\mathbf{\Lambda}} \tilde{\mathbf{U}}^T$  in equation (4.9) the approximation can be rewritten as:

$$\tilde{\mathbf{K}} = \mathbf{K}_{N,M} (\mathbf{K}_{M,M})^{-1} \mathbf{K}_{M,N} \quad (4.11)$$

From (4.11) one can see that the kernel approximation can fully be made by the sample matrix and the submatrix  $\mathbf{K}_{N-M,N}$  of the 'original' kernel.[90]

The error bound of the Nyström Kernel approximation is  $E_{Nys} = \|\mathbf{K} - \tilde{\mathbf{K}}\|$ . [40] This error can be minimized by choosing the samples carefully and a reasonable amount of it. If the original Kernel  $\mathbf{K}$  has rank  $M$  and the samples are linearly independent columns from the kernel, then the approximation is exact. As a consequence, from above choosing only  $P$  samples with  $P < M$  the error rises the fewer columns are chosen.[90]

The problem with choosing (sampling) columns/rows is that we usually do not know which columns are linearly independent. Therefore, some sampling methods or approximations were made in the past to make this sampling more deterministic. For example [52], which tries to find good samples through various sampling techniques, or [54], where the authors are drawing a large sample but approximate the submatrices with Single Value Decomposition (SVD).

To summarize the Nyström Kernel approximation, first, we select  $P$  of  $N$  'good' samples from the data and reorganizing the kernel according to (4.8). At this point, we have two options: First calculating the kernel just for  $\mathbf{K}_{M,M}$  and the submatrix  $\mathbf{K}_{N-M,M}$ , continuing with approximate the eigenvectors and eigenvalues and finally calculating the approximated kernel  $\tilde{\mathbf{K}}$  (4.10). Second calculate the approximation directly from the kernel as shown in equation (4.11). Note that the second approach needs a matrix inversion of the kernel, which should be considered for the use with large datasets.

## 4.2 Domain Invariant Kernel Learning

The Transfer Kernel Learning (TKL) approach is made by Long et al. in [57]<sup>1</sup>. They are interpreting the kernel approximation from a different point of view. The main idea is to see the train dataset-matrix, which has to be approximated by the test dataset. Proceed by finding eigenvalues in a way that the difference between the approximation and ground truth training kernel will be minimal, which results in aligned marginal probability distributions. This forms a domain invariant kernel for

<sup>1</sup><http://ise.thss.tsinghua.edu.cn/~mlong/doc/transfer-kernel-learning-tkde15.zip>

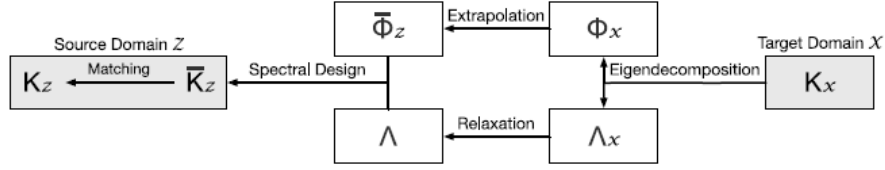


Figure 4.1: The process of the Domain Invariant Kernel learning. It illustrates the relaxation of eigenvalues and extrapolation of eigenvectors. The tasks relaxation and extrapolation are forming the new domain invariant kernel.[57]

transfer learning.[57] The principal process is shown in figure 4.1.

From the above consider  $\mathbf{z}$  as  $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$  as training dataset which is sampled from  $p(\mathbf{Z})$  within training domain  $\mathcal{Z}$  and  $\mathbf{x}$  as  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$  as test dataset sampled from  $p(\mathbf{X})$  within the test domain  $\mathcal{X}$ . The definitions are based on 3.2. The associated kernels are  $\mathbf{K}_Z$  for training and  $\mathbf{K}_X$  for testing.

A fundamental insight of the RKHS is that any PSD kernel can be reconstructed based on the eigensystem of it. Furthermore, any new datasets can be incorporated into a kernel.[57]

This means given a kernel  $\mathbf{K}_X$  of the dataset, using only the eigensystem  $\{\lambda_i, \phi_i(\mathbf{x})\}$  the kernel can be evaluated by any new data point  $\mathbf{z}$  and finally generating the kernel  $\mathbf{K}_Z$ . Again we could approximate the needed training kernel through:[57]

$$\mathbf{K}_Z \simeq \mathbf{U}_Z \mathbf{\Lambda}_X \mathbf{U}_Z^T = \mathbf{K}_{Z\mathcal{X}} \mathbf{K}_X^{-1} \mathbf{K}_{X\mathcal{Z}} \quad (4.12)$$

Where  $\mathbf{U}_Z$  as eigenvectors of the training kernel,  $\mathbf{\Lambda}_X$  as eigenvalues of the target kernel and  $\mathbf{K}_{Z\mathcal{X}} = \mathbf{K}_{Z\mathcal{X}}^T$ . This is, in fact, a rewritten version of (4.11).

However, there might be a major problem. Regarding two feature spaces, especially when it comes to transfer learning, then we have made the assumption that test and training feature spaces following different probability distributions. This means that  $P(\mathbf{Z}) \neq P(\mathbf{X})$  and as a consequence the error of the approximation from (4.12) can be arbitrarily large.[57]

However, to align the distribution difference is one goal of transfer learning. The main improvement which is done by Long et al. is to tackle this problem and to learn the domain invariant approximated kernel  $\bar{\mathbf{K}}_{\mathcal{A}}$ . [57]

### 4.2.1 Learning Approach

From a 'kernel' point of view aligning the distribution of data, where the kernel is based on, can be formulated as  $P(\phi(\mathbf{z})) \simeq P(\phi(\mathbf{x}))$ . [57] However, the kernel-induced feature map cannot be explicitly represented and therefore the handling of distributions in the corresponding Hilbert space is difficult.[47]

Therefore, Long et al. take the assumption that it is sufficient that  $\mathbf{K}_Z \simeq \mathbf{K}_X$ .

However, there is another problem coming up with it, because it can not be assumed that the training and test sets are equal, which means that the sizes of the kernels are not equal, i. g.,  $\mathbf{K}_Z \in \mathbb{R}^{N \times N}$  and  $\mathbf{K}_X \in \mathbb{R}^{M \times M}$  with  $N \neq M$ . As a consequence, the extrapolated source kernel  $\bar{\mathbf{K}}_Z$  will be  $\mathbb{R}^{N \times N}$ , by using the eigensystem of the target kernel  $\mathbf{K}_X$ . Therefore, the kernel  $\bar{\mathbf{K}}_Z$  will be compared to the actual training kernel  $\mathbf{K}_Z$ . [57]

For clarifying the terms, Long et al. proposed the term extrapolated instead of approximation regarding the Nyström approximation. [57] In this thesis, we will stick to the proposed notation to coincide.

The task is now to learn this extrapolated kernel, which is done step by step by first *extrapolate* the eigenvectors with: [57]

$$\bar{\mathbf{U}}_Z \simeq \mathbf{K}_{ZX} \mathbf{U}_X \mathbf{\Lambda}_X^{-1} \quad (4.13)$$

According to Nyström, the eigenvalues  $\mathbf{\Lambda}_X$  of the test kernel are indispensable. However, in this approach, a *relaxation* of the eigenvalues is done by considering the eigenvalues as learnable parameters  $\mathbf{\Lambda} = \text{diag}[\lambda_1, \dots, \lambda_M]$ . Therefore, the kernel extrapolation can be expressed as: [57]

$$\bar{\mathbf{K}}_Z = \bar{\mathbf{U}}_Z \mathbf{\Lambda} \bar{\mathbf{U}}_Z^T \quad (4.14)$$

With that  $\mathbf{\Lambda}$  does not necessarily reduce the distribution difference and has to be well chosen. This approach should preserve the original structure of the test domain while remaining flexible to solve the difference in distribution. Another point of view is that the kernel is extrapolated from the target data but evaluated by the training data. [57]

To construct the new kernel, Long et al. using the key knowledge from spectral kernel design: It states that a kernel which is generated from the eigensystem of another positive semi-definite kernel, then the produced kernel will positive semi-definite itself. [47]. Therefore, it seems reasonable to set  $\lambda \geq 0$  to guarantee a PSD kernel. The error of the approximation  $\bar{\mathbf{K}}_Z$  to the original ground truth kernel  $\mathbf{K}_Z$  is determined using the squared loss, which is presented in the following: [57]

$$\begin{aligned} \min_{\mathbf{\Lambda}} \|\bar{\mathbf{K}}_Z - \mathbf{K}_Z\|_F^2 &= \|\bar{\mathbf{U}}_Z \mathbf{\Lambda} \bar{\mathbf{U}}_Z^T - \mathbf{K}_Z\|_F^2 \\ \lambda_i &\geq \zeta \lambda_{i+1}, i = 1, \dots, M-1 \\ \lambda_i &\geq 0, i = 0, \dots, M \end{aligned} \quad (4.15)$$

With  $\zeta \geq 1$  is the eigenspectrum dumping factor since the eigenspectrum is power-law distributed and therefore  $\zeta$  should produce larger eigenvectors. With this, they should contribute more to the knowledge process. [57]

### 4.2.2 Learning Algorithm

In the section above, we defined the squared error loss function for the TKL approach. In this subsection, we will introduce the resulting optimization problem and the TKL algorithm in general.

The learning problem is convex and therefore will always find the global minimum. Furthermore, it is a Quadratic Program (QP) with linear constraints which can be solved for example by the build in MatLab package *quadprog*. The squared loss function from (4.15) is reformulated in a general matrix format and is expressed as:[56]

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \boldsymbol{\lambda}^T \mathbf{Q} \boldsymbol{\lambda} - 2\mathbf{r}^T \boldsymbol{\lambda} \\ & \mathbf{C} \boldsymbol{\lambda} \geq 0 \\ & \boldsymbol{\lambda} \geq 0 \end{aligned} \quad (4.16)$$

The inequalities representing the linear constraints. The coefficient matrix  $\mathbf{Q}$ ,  $r$  and the constraint matrix  $\mathbf{C}$  are defined as:

$$\begin{aligned} \mathbf{Q} &= (\bar{\mathbf{U}}_{\mathcal{Z}} \bar{\mathbf{U}}_{\mathcal{Z}}^T) \odot (\bar{\mathbf{U}}_{\mathcal{Z}}^T \bar{\mathbf{U}}_{\mathcal{Z}}) \\ r &= \text{diag}[\bar{\mathbf{U}}_{\mathcal{Z}}^T \mathbf{K}_{\mathcal{Z}} \bar{\mathbf{U}}_{\mathcal{Z}}] \\ \mathbf{C} &= \mathbf{I} - \zeta \mathbf{I} \end{aligned} \quad (4.17)$$

Where  $\mathbf{I} \in \mathbb{R}^{M \times M}$  as the identity matrix and  $\odot$  as Hadamard multiplication. Note that the eigenspectrum dumping factor  $\zeta$  is the only free parameter which needs to be tuned because the  $\boldsymbol{\Lambda}$  is learned within the optimization problem.

Finally, similar to equation 4.8, we can reconstruct the 'joint' Kernel:[56]

$$\bar{\mathbf{K}}_{\mathcal{A}} = \begin{bmatrix} \bar{\mathbf{U}}_{\mathcal{Z}} \boldsymbol{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T & \bar{\mathbf{U}}_{\mathcal{Z}} \boldsymbol{\Lambda} \mathbf{U}_{\mathcal{X}}^T \\ \mathbf{U}_{\mathcal{X}} \boldsymbol{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T & \mathbf{U}_{\mathcal{X}} \boldsymbol{\Lambda} \mathbf{U}_{\mathcal{X}}^T \end{bmatrix} = \bar{\mathbf{U}}_{\mathcal{A}} \boldsymbol{\Lambda} \bar{\mathbf{U}}_{\mathcal{A}}^T \quad (4.18)$$

Where  $\mathcal{A} = \mathcal{Z} \cup \mathcal{X}$  as one dataset for two domains. Therefore, it forms the eigenvector matrix  $\bar{\mathbf{U}}_{\mathcal{A}} = [\bar{\mathbf{U}}_{\mathcal{A}}; \mathbf{U}_{\mathcal{X}}]$ , which are the extrapolated eigenvectors of the training dataset and the eigenvectors of the test dataset.

The complete TKL algorithm is summarized in algorithm 1.[57] The underlying ker-

---

#### Algorithm 1 Transfer Kernel Learning

---

**Input:** Input Data  $\mathbf{I} = [\mathbf{Z}; \mathbf{X}]$ ; kernel(-type)  $k$ ; eigenspectrum dumping factor  $\zeta$ .

**Output:** Domain invariant kernel  $\bar{\mathbf{K}}_{\mathcal{A}}$ .

- 1: Compute the kernel parts  $\mathbf{K}_{\mathcal{Z}}$ ,  $\mathbf{K}_{\mathcal{X}}$  and  $\mathbf{K}_{\mathcal{Z}\mathcal{X}}$  with  $k$ .
  - 2: Eigendecompose of  $\mathbf{K}_{\mathcal{X}}$  for  $\{\boldsymbol{\Lambda}_{\mathcal{X}}, \mathbf{U}_{\mathcal{X}}\}$  like (4.5).
  - 3: Extrapolate for source eigensystem  $\bar{\mathbf{U}}_{\mathcal{Z}}$  with (4.13).
  - 4: Solve the QP for eigenspectrum  $\boldsymbol{\Lambda}$  with (4.16).
  - 5: Merge results and return it as (4.18).
-



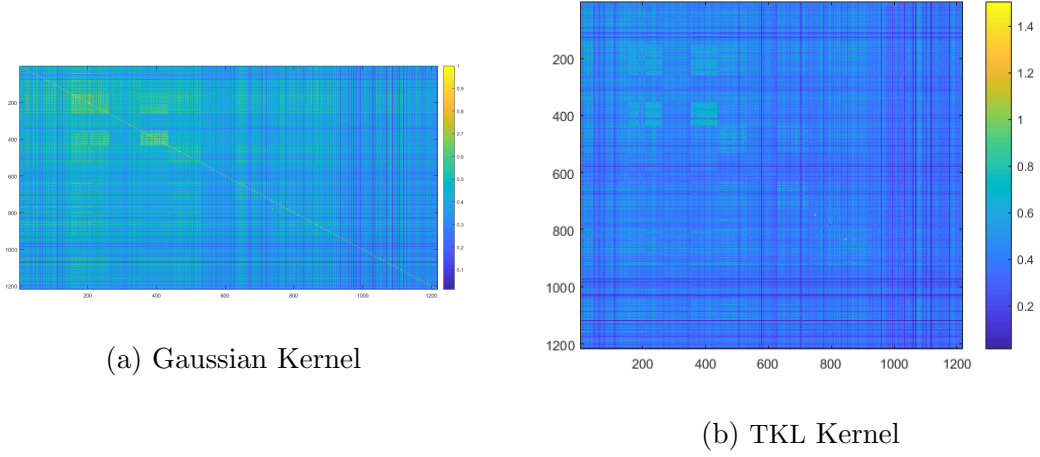


Figure 4.2: A scatter plot of the Gaussian and the TKL kernel. On the left side the Gaussian kernel is shown. On the right, the TKL kernel is shown. It is observable that the differences of source and target sets are slightly aligned. The kernel is based on the image set. This figure is best viewed with colors.

nel which is calculated in the first row of algorithm 1 is the Gaussian kernel from (5.3).

The process of getting the TKL kernel is shown in figure 4.2. In 4.2a, in the upper left, the source set is visible. This means that the source and target dataset is separable from each other. After creating the TKL kernel in 4.2b, the differences are slightly aligned.

The error of the approximation of the Transfer Kernel Learning algorithm is  $E_{TKL} = \|\bar{\mathbf{U}}_{\mathcal{Z}} \mathbf{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T - \mathbf{K}_{\mathcal{Z}}\|$ . Furthermore, the approximation error of the Nyström kernel is bound by  $E_{NystKer} = \|\mathbf{K}_{\mathcal{Z}\mathcal{X}} \mathbf{K}_{\mathcal{X}}^{-1} \mathbf{K}_{\mathcal{X}\mathcal{Z}}\|$ . Based on these errors, if  $\mathbf{\Lambda} = \mathbf{\Lambda}_{\mathcal{X}}$ , then  $E_{NystKer}$  and  $E_{TKL}$  are equal. However, because the  $\mathbf{\Lambda}$  is a free parameter and the approximation error is directly minimized, the error of the TKL approximation is bound by  $E_{TKL} \leq E_{NystKer}$ . [56]

In many real-world problems, the eigenvalues are following the power law distribution. This means that there are a few large eigenvalues and many small, in comparison with the large, eigenvalues. [60] Therefore, it is considered as unnecessary to use the whole eigenspectrum and just compute the  $R$  largest eigenvalues. This is used to speed up the computation because the problem is greatly reduced. The number of eigenvectors is therefore fixed to  $R = \min(500, M)$  and the eigenvectors can be reduced to  $\bar{\mathbf{U}}_{\mathcal{Z}} \in \mathbb{R}^{R \times R}$  or  $\lambda \in \mathbb{R}^{R \times M}$ . [56]

The complexity of the TKL algorithm can be given with  $\mathcal{O}((D+R)(N+M)^2)$ , where  $R$  denotes the number of used eigenvectors,  $D$  refers to the dimensions of data. [56]

### 4.3 PCTKVM Algorithm

In this section, the Probabilistic Classification Transfer Kernel Vector Machine algorithm will be discussed in detail. As shown above, we have obtained our transfer learning kernel with the TKL algorithm (1). According to [57] the new kernel can be feed to any kernel machine. This means that we can train the regular Probabilistic Classification Vector Machine (PCVM) with our new kernel. To train it we are using the extrapolated training kernel which is.

$$\bar{\mathbf{K}}_Z = \bar{\mathbf{U}}_Z \mathbf{\Lambda} \bar{\mathbf{U}}_Z^T \quad (4.19)$$

However, simply using it in the PCVM algorithm (3) has some disadvantages. Revisiting the PCVM algorithm, the kernel is recalculated in every iteration based on the optimized theta from the previous iteration. As a consequence, we have to recalculate the entire transfer kernel too. The complexity of the PCVM is  $\mathcal{O}(B^3)$  with  $B$  as a number of basis functions. We consider the worst case and substitute the number of basis function with the number of training samples  $N$ , which results in  $\mathcal{O}(N^3)$ . The complexity of the TKL is  $\mathcal{O}((D+R)(N+M)^2)$ , where  $R$  is the number of eigenvectors and  $D$  is the dimension of the data. Combining these two, we would end up with a computational complexity of  $\mathcal{O}(N^3(D+R)(N+M)^2)$ . This might be resulting in a long computational time. In fact, the question is, whether we explicitly need the theta optimization in every iteration or not. However, one advantage of the PCVM was, to optimize the parameters within and therefore omit the need for a parameter optimization via grid search. Furthermore, the performance of the PCTKVM depends a lot from the quality of the theta, which can be seen in figure 4.3. To keep this advantage and the performance up, a new approach must be found. One approach could be to optimize the theta in not every iteration. However, this seems to be no good solution, because we have observed that the theta is just dangling around the initial value, where we optimized the theta in every 5 or 10 iterations and nevertheless was leading in our tests to a worse performance, rather than just use a fixed value for theta. The latter option would be to set the theta to a fixed value and omit the theta optimization completely. Fair enough we can observe a good performance, which can be seen in section 5. Furthermore, we could obtain the complexity of  $\mathcal{O}(N^3 + (D+R)(N+M)^2)$ . If we consider  $D$  as negligible because  $D \ll N$ ,  $R$  as constant and by multiplying out, we obtain  $\mathcal{O}(N^3 + M^2)$  for the PCTKVM with a fixed theta. For large  $D$  we obtain  $\mathcal{O}(N^3 + DM^2)$ .

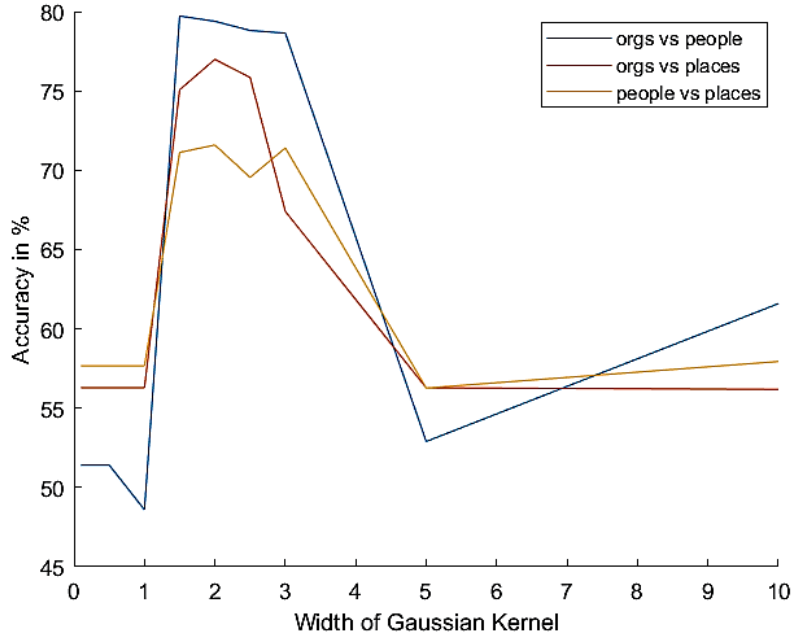


Figure 4.3: The performance of the PCTKVM algorithm for different thetas shown as accuracy in % which is evaluated on the Reuters dataset. It shows that the performance of the algorithm depends a lot on the quality of the theta.

### 4.3.1 Theta Estimation

Another, more deterministic but heuristic, approach to obtain a theta is presented by Kitayama in [48] and is discussed in the following.

It is important to notice that is just a simple estimation and therefore the optimal theta may not be found with this approach.

The width of the Gaussian kernel can play an important role not only for the PCVM but many other algorithms. Defining a 'good' value for the width leads to a more useful function which is observable in figure 4.4.[48] In this the black dots representing the data, the dashed lines are the Gaussian kernel, and the solid lines represent the regression. In 4.4a the width is set to 0.5 and in 4.4b it is set to 1.

They are claiming in general that solving the theta via an optimization problem is very time-consuming. The reason for this is because the optimization problem is depending on the data function, which can give a very large amount of variables and therefore proposed a simpler algorithm to find a good  $\theta$ . [48]

First, they use a scaling technique to scale every dimension to an equal range. Consider  $K = N + M$  as the number of all samples with  $D$  dimensions. In the approach, they using a *Min-Max* normalization to do it:[48]

$$x_j = \frac{x_j - x_j^L}{x_j^U - x_j^L} \times s, \quad i = 1, \dots, D \quad (4.20)$$

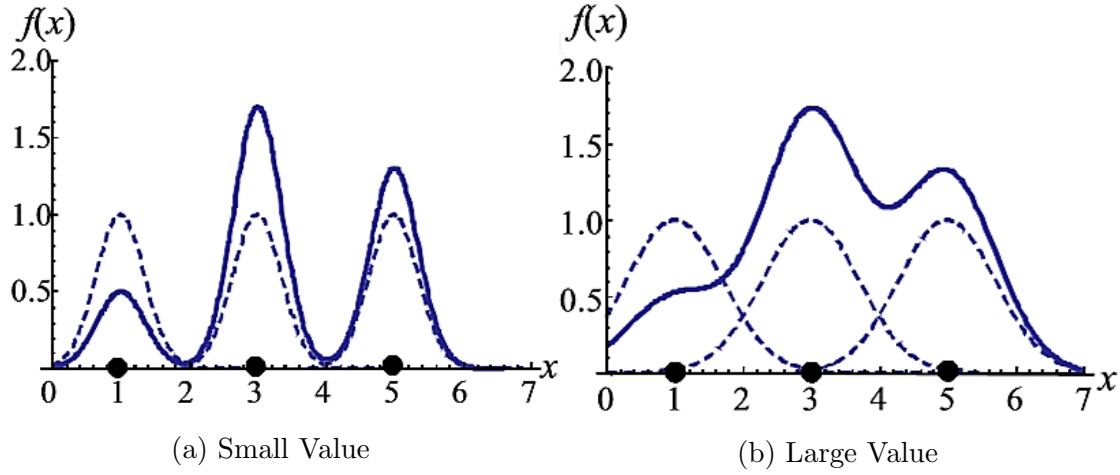


Figure 4.4: The effect of regression concerning the width of the Gaussian kernel. The left figure shows the effect of a width of 0.5. The right plot has used one as width. The black dots representing the data, the dashed lines are the Gaussian kernel, and the solid lines represent the regression. It can be seen that the larger width provides a smoother regression.[48]

Where  $x_j^L$  is the lower bound and  $x_j^U$  is the upper bound of value and  $s = \alpha \times s\alpha \geq 0$ . The  $s$  is the abort criterion, but not needed in our approach. The author's recommendation is to set alpha to 1.1. Moreover, in every iteration of the algorithm, the dimension are rescaled.

However, in general, the training and test data for the PCVM must be z-scored, as discussed in section 2. Therefore, it seems reasonable to replace the scaling technique from *Min-Max* to the z-score, obtained for example from [61]:

$$x_{ij} = Z(x_{ij}) = \frac{x_{ij} - \bar{x}_j}{\sigma_j}, \quad i = 1, \dots, K \quad j = 1, \dots, D \quad (4.21)$$

Where  $\bar{x}_j$  is the mean and  $\sigma_j$  is the standard deviation in every dimension.

Furthermore, because after one run of z-score the mean is 0 and the standard deviation is 1.[61] Revisiting (4.21) with this, we obtain  $x_{ij} = Z(x_{ij}) = \frac{x_{ij}}{1}$  after the first run, which is the same for every following iteration. With this change, we omitted the need for further iteration and can calculate the width in one iteration.

After the z-score is applied, and the dimensions are equally scaled they originally continue with determining the distance:[48]

$$\theta_i = \frac{d_{i,max}}{\sqrt{D} \sqrt[p]{K-1}}, \quad i = 1, \dots, K \quad (4.22)$$

Where  $d_{j,max}$  is the maximum distance between the  $i$ -th data point and another data point. Moreover,  $\theta_i$  is the width for the  $i$ -th Gaussian kernel in every row.

This is the main improvement in comparison with other approaches to determine the width because for example  $\theta = \frac{d_{max}}{\sqrt[p]{K} \sqrt{D}}$  can only be applied to uniform distributed

data. However, with (4.22) the distance can also be applied to non-uniform distributed data, which is mostly the case in real-world problems.[48]

Finally, we set our width to the smallest maximal distance found in (4.22). The estimate can be summarized in the following steps based on [48]:

**Step 1:** Rescale data using z-score from (4.21)

**Step 2:** Calculate the distance matrix

**Step 3:** Find maximum distance for  $K$  data points using (4.22)

**Step 4:** Find the minimum distance for every row with  $\theta_{min} = \min_{1 \leq i \leq K} \theta_i$  and return  $\theta_{min}$

To calculate the distance in practice, we are using a dissimilarity matrix  $\mathbf{D}_{K,K}$ . This matrix employs the Euclidean distance with  $d_{i,j} = \|x_i - x_j\|^2, \forall i, j \in \mathbf{D}$  as dissimilarity measure. This matrix is symmetric and is zero on the diagonal.[32, p. 22;299] Because the Gaussian kernel is used in this approach, the dissimilarity matrix is already involved in the computational complexity. Therefore, this matrix has just to be calculated once and can be reused in the theta estimation and the Gaussian kernel. Another important thing to notice is that the data is already z-scored in the preparation and therefore there is no need to do it twice. With this, we just search the maximum distance with the built-in MatLab function *min()* and can proceed with (4.22) in Step 3.

Although we can reduce the complexity with a clever implementation, the complexity discussion is based on the worst case involving all steps to compute, except z-score. The computational complexity of calculating the dissimilarly matrix is  $\mathcal{O}(K^2)$ .[49] Furthermore, finding a maximum in  $K$  rows with  $D$  values can be solved in  $\mathcal{O}(KD)$ . Because the dissimilarity matrix is symmetric the complexity changes to  $\mathcal{O}(K^2)$  and multiplying a scalar to  $K$  elements costs  $\mathcal{O}(K)$ , which can be summarized in  $\mathcal{O}(K^2 + K) = \mathcal{O}(K^2)$ . Again finding a minimum within the theta array from step four takes  $\mathcal{O}(K)$ . Summarizing, we can obtain the complexity of the theta estimation with  $\mathcal{O}(K^2)$ . Note that we observed for small datasets, that the theta would be large, which will eventually lead to a worse performance, see section 5.1 and table 5.3.

### 4.3.2 Training Algorithm

With the results from the previous section, the PCTKVM is presented in algorithm 2. Note that the complete algorithm which involves every detail can be found in appendix C. As already discussed, we are reusing the dissimilarity matrix. Therefore, the function parameters of *transfer\_kernel\_Learning*( $\mathbf{D}, \cdot$ ) from the TKL algorithm

**Algorithm 2** Probabilistic Classification Transfer Kernel Vector Machine (Short)

**Input:** Input Data  $\mathbf{K} = [\mathbf{Z}; \mathbf{X}]$  as  $N$  sized training and  $M$  sized text set;  $\mathbf{Y}$  as  $N$  sized training label vector; kernel(-type)  $ker$ ; eigenspectrum dumping factor  $\zeta$ ;  $\theta$  as kernel parameter;  $niter$  as maximal number of iterations; *threshold*  $\tau$  as convergence criteria; **InitVector** as  $N$ -sized initialization vector.

**Output:** Weight Vector  $\mathbf{w}$ ; bias  $b$ , kernel parameter  $\theta$ ; transfer kernel  $\bar{\mathbf{K}}_{\mathcal{A}}$ .

- 1:  $\mathbf{D} = \text{calculate\_Dissimilarity\_Matrix}(\mathbf{K})$ ;
- 2:  $\theta = \text{theta\_Estimation}(\mathbf{D})$ ; ▷ According to section (4.3.1) (optional)
- 3:  $\bar{\mathbf{K}}_{\mathcal{A}} = \text{transfer\_Kernel\_Learning}(\mathbf{D}, ker, \theta, \zeta)$ ; ▷ According to (4.2)
- 4:  $[\mathbf{w}, b] = \text{pcvm\_Training}(\bar{\mathbf{K}}_{\mathcal{Z}}, \mathbf{InitVector}, niter, \tau)$ ; ▷ Algorithm (3)

(1) changing from using the raw data to using the dissimilarity matrix. We observed in practice that the kernel is for some  $\theta$ , not PSD anymore, although he is in theory.[57] We assume that there is some issue in the floating point multiplication for small numbers in MatLab and therefore added the regularization term  $eps\mathbf{I}_{K \times K}$  to the kernel, which solved the problem.

It can be seen that the number of free parameters remains the same because the  $\theta$  is estimated and only the eigenspectrum dumping factor  $\zeta$  is left. Furthermore, we obtain by, combining the complexity of the theta estimation with the complexities of PCVM and TKL, the same complexity as already discussed, which is  $\mathcal{O}(N^3 + M^2)$ . Again, for a large  $D$  we obtain  $\mathcal{O}(N^3 + DM^2)$ . This is achieved by reusing the dissimilarity matrix for the parameter estimation and the kernel function. This statement can be validated in practice with table B.5, which shows that the PCTKVM version with fixed theta and the algorithm with estimation need a similar time.

### 4.3.3 Prediction

Additionally, we can make predictions based on the provided prediction function of the PCVM with the use of  $\bar{\mathbf{K}}_{\mathcal{XZ}} = \mathbf{U}_{\mathcal{X}}\mathbf{\Lambda}\bar{\mathbf{U}}_{\mathcal{Z}}^T$  as the kernel for prediction.[57]

Because of the sparsity of the PCVM, the kernel size is greatly reduced as discussed in section 2. If we consider that our model has  $B$  non-zero weight vectors with  $B < N$  and because the PCVM uses only kernel rows/columns corresponding to the non-zero weight vector index, then our final kernel  $\bar{\mathbf{K}}_{\mathcal{BX}}$  for prediction has size  $B \times M$ . Therefore, the prediction function of the PCTKVM has the form:

$$\mathbf{y} = \bar{\mathbf{K}}_{\mathcal{BX}}\mathbf{w} + b \quad (4.23)$$

Finally, the class label is obtained from the sign of the elements  $y_i$  the label vector  $\mathbf{y}_{\mathcal{X}}$ , which has size  $M$ .

Note that the TKL transforms the data into an RKHS. Bases on the reproducing property of an RKHS, any additional target points could be integrated into the

kernel.[57] Moreover, they can be evaluated with the PCTKVM.

However, consider that new target points, which are evaluated after the calculation of the transfer kernel, should be drawn from the same marginal distribution as the previous target data. If this prerequisite cannot be held, then the whole kernel has to be recalculated with the old and the composition of target data to obtain proper performance.

The probabilistic output is calculated with the probit link function as the PCVM does it in equation 2.15.

### 4.3.4 Extensions

In the course of this work, there was the need to implement a multi-class option for the PCVM, PCTKVM and PCTKVM <sub>$\theta_{\text{Est}}$</sub> , because the Image dataset has more than two classes.

In general, there are many approaches to solve a multi-class problem for vector machines.[1, p. 113] According to the LibSVM documentation<sup>2</sup>, for the LibSVM Library, two approaches are considered: The one vs. all and the one vs. one approach. In the following, we will discuss the advantages and disadvantages for these two ideas and explain how it is integrated into the PCVM.

The first is the one vs. rest approach. Note that in the literature, the term one vs. all appears instead of one vs. rest. If we consider  $C$  classes, then the classifier would be trained with one class against the  $C - 1$  remaining classes. This approach will have  $C$  iteration, and in every iteration, another class is trained against every remaining class. The label assignment for one class is done in one iteration only. However, this will lead to large unclassifiable regions in the feature space.[1, p. 114-116]

Therefore, the second approach, one vs. one is implemented. With this, we split up the labels in the different classes and train the classifier with every class against the others. Again, if we take  $C$ , different classes, then our number of iteration is  $I = C(C - 1)/2$ . Furthermore, the unclassifiable region in comparison with the one vs. rest approach is smaller. For the prediction, we do  $I$  iterations again and predict the whole dataset with two classes.[1, p. 127-128]

In the implementation, consider a test set with size  $M$ . Then our label matrix  $\mathbf{Y}$  and the probabilistic output matrix  $\mathbf{P}$  has size  $M \times K$ .

We choose a final label for the final label vector  $\mathbf{y}$  by counting the number of occurrences of a label for a data point. The data point is then assigned to the label which the classifier has assigned the most. In case of a tie the smaller class label, in terms of numbers, is assigned, according to the MatLab *mode*<sup>3</sup> function. This

<sup>2</sup><http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#f419>

<sup>3</sup><https://de.mathworks.com/help/matlab/ref/mode.html>

is rather unconventional because it is likely to be a random selection, but in our tests, the results are the better as to assign the class label according to the biggest probability given from  $\mathbf{P}$ .

To determine a final probability output for a final label decision concerning a data point, the algorithm calculates the mean of the probabilistic output for the runs, where the label is equal to the final label, which is assigned.

The number of relevance vectors is determined by counting all unique vectors, which are used during the  $C$  iterations.

## 4.4 Conclusion

In this chapter, we successfully integrated transfer learning in the PCVM. We used a proposed transfer learning method within the PCVM, which forms the PCTKVM. Furthermore, we solved the problem that the algorithm will be slow through the theta optimization by adopting a simple theta estimation and adjust it to the needs of the PCVM. Because it uses unlabeled target data, the algorithm is transductive and solves the transfer problem with a relational-knowledge approach regarding homogeneous transfer learning.

Additionally, we extend the PCVM and PCTKVM to being able to solve multi-class problems via the one vs. one approach. The MatLab source code of the PCTKVM and PCTKVM <sub>$\theta_{Est}$</sub>  can be obtained from Github<sup>4</sup>. Furthermore, the reader can find the complete code written for this thesis, including all tests and transfer learning approaches used in the study, in the repository.

---

<sup>4</sup><https://github.com/ChristophRaab/pctkvm.git>



# Chapter 5

## Benchmark Study

In this chapter, the preparations and the result of the study are described. With this, used datasets to determine the performance are defined, and the set of transfer learning classifiers with corresponding parameters are appointed.

Furthermore, used statistical methods in the study are introduced, and the statistics are shown and discussed. This study is divided into three parts of statistics. The first part is a basic descriptive statistics to give an overview of the general performance of the classifiers. This is followed by the comparison of classifiers over one dataset and finally over multiple datasets. With that all datasets described in this thesis are tested.

The procedure and selection of statistics are mainly adopted from [19] and [57]. The main reason for this is to guarantee a fair comparison of the PCVM, PCTKVM and the TKL method as wrapper algorithm for the SVM.

On first sight, it may be strange to compare first the difference between single datasets and proceed with an overall test. This is made because of two analytic reasons. First, the study has two different kinds of datasets (image and text) and many subcategories and subsets. With the single test, it can be determined if there is a significant difference between the performance of classifiers and datasets. When it comes to a real-world scenario, the choice of a classifier may depend on his strengths and weaknesses to solve a task on a specific kind of dataset. With this listing of performance may one can choose more precisely which classifier he chooses for the job. At least for text documents and images.

On the other hand from the scientific point of view, the aggregated comparison over all used datasets and the results in performance giving the most representative insights.

Note that in this chapter for clarity only the aggregated results are shown. The complete results are shown in appendix B.

## 5.1 Dataset Description

The study consists of 18 benchmark datasets. Three of them are text-based from Reuters-21578<sup>1</sup> and the remaining twelve are images from Caltech-256<sup>2</sup> and Office. Note that the datasets are not extracted from the raw data, but rather used from others with in a pre-processing state. The corresponding articles, which are use this version of datasets are [35] and [57].

This 18 sets are well known in the area of domain adaptation and transfer learning and for example used in [57],[35],[30],[56],[22],[69] and discussed in [66].

A crucial characteristic of the datasets is that the domains for training and testing are different but related. This relation exists because the train and test classes have the same top category or source. The classes themselves are subcategories or subsets.[46]

The relation will be clear in the separate sections for the datasets respectively.

### 5.1.1 Text Dataset

The text for the Reuters-21578 collection appeared on the Reuters news-wire in 1987. The documents were assembled with categories by personnel from Reuters Ltd. and Carnegie Group, Inc (CGI). Reuters and CGI published the documents for research purposes.[23]

It is a frequently used text collection to evaluate the performance of text categorization techniques. The collection is segmented in five top categories and many subcategories. For testing classifiers, the task is to evaluate on three big categories *orgs*, *people* and *places*. In the last subcategory, all documents with the topic USA are removed. This makes the categories nearly even because more than a half of the documents are in the USA subcategory. Furthermore, some pre-processing is done at the collection:[89]

First, all letters are converted to lower case, and the words are stemmed using the Porter stemmer. This stemmer removes the suffixes form the terms.[67] Furthermore, stopwords are removed. With the Document Frequency (DF)-Threshold, the numbers of features are cut down. To apply the DF-Threshold the number of documents in which a certain term occurs has to be counted. Each term which is less than a previously set threshold is removed from the feature space.[92] The Threshold parameter is set to three with the goal to speed up the classification process. Finally, Term-Frequency Inverse-Document-Frequency (TFIDF) is applied for feature generation. The TFIDF value is used to train the classifiers. The TFIDF is a measure of how well a term characterizes a document. It takes the frequency of a term in a

<sup>1</sup><http://www.daviddlewis.com/resources/testcollections/reuters21578>

<sup>2</sup>[http://www.vision.caltech.edu/Image\\_Datasets/Caltech256/](http://www.vision.caltech.edu/Image_Datasets/Caltech256/)

Top Category	#Examples 1	#Examples 2	#Features	# Labels	KLD	MMD
Orgs vs. People	1237	1208			0.4570	0.0410
Orgs vs. Places	1208	1016	4771	2	0.7420	0.0455
People vs. Places	1016	1208			0.6690	0.0421

Table 5.1: Overview of the key figures of the Reuters-21578 dataset. It shows the number of sample points, features, labels and the KLD and MMD divergences.

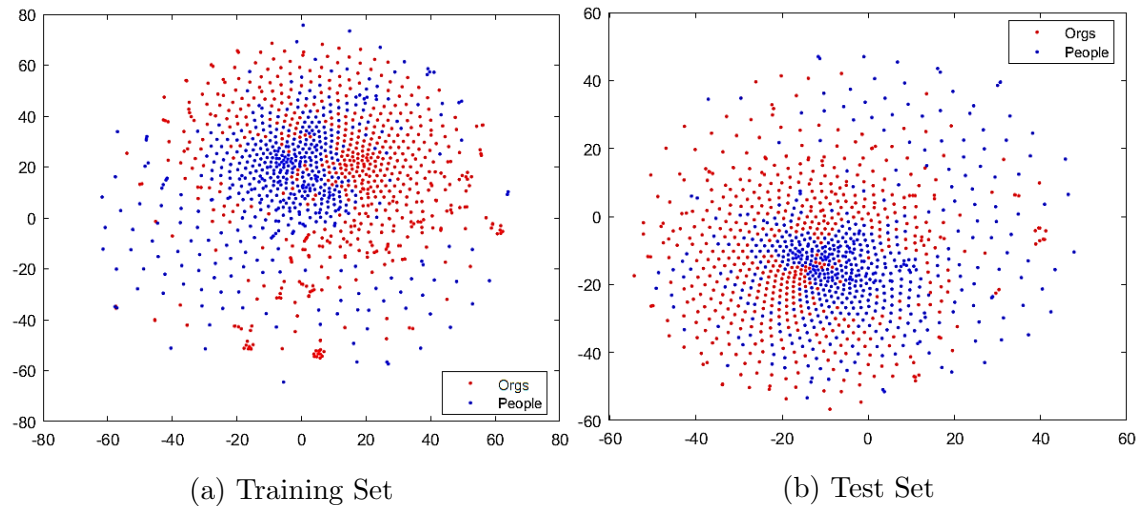


Figure 5.1: The plot of the *Orgs vs. People* dataset. On the left hand is the training set, and on the right, the test set is shown. The dimension is reduced with t-SNE to two dimensions. The KLD to the original Dataset is 2.233 and 2.464. This is best viewed with colors.

document times the inverse document frequency which is the number of occurrences of this term in all documents. [53, p. 26]

For the domain adaptation purpose, six datasets were generated from Reuters-21578: *orgs vs. places*, *orgs vs. people*, *people vs. places*, *places vs. orgs*, *people vs. places* and *places vs. people*. The labels are created by assigning orgs for example as positive class and people as the negative class. Furthermore, the two labels are split by its subcategories to create the training and test sets.[89]

In table 5.1 the dataset with its features, observations, labels and the corresponding divergences, KLD and MMD, are shown. The divergences are defined in Section 3.3.2. To get an overview of the distribution and its transfer learning purpose the *Orgs vs. People* dataset is shown in Figure 5.1.

### 5.1.2 Image Dataset

The image dataset has two main Parts. The first one which is named *office* is collected and created from [72]. This contains various images from three different sources which are forming four datasets. One are images taken from the web,

Name	#Examples 1	#Examples 2	#Features	# Labels	KLD	MMD
Caltech vs. Amazon		958			1.2062	0.0449
Caltech vs. DSLR	1123	295	800	10	1.1418	0.0654
Caltech vs. Webcam		157			1.1567	0.0852

Table 5.2: Overview of the key figures of the Image dataset. It shows the number of sample points, features, labels and the KLD and MMD divergences.

downloaded from the online merchant's *amazon* (Name in Dataset). Originally 31 categories, which are different types of products (objects), with an average of 90 images are captured. The objects are typically shown from a canonical viewpoint. A canonical viewpoint is a certain view of an object in a certain rotation, which is considered as 'good' to recognize the object correctly. [27]

The second consists of images that are captured with a digital SLR camera (*dslr*). The conditions should be realistic environments (*office*) and natural lighting. The resolution of the images is high with low noise. With that 31 object categories and five different objects within an object category are taken. An object contains on average three images, taken from various viewpoints. This makes a total of 423 images. [72]

The last set of images are from a *webcam*. Images are taken from this, with a low resolution and high noise. The purpose of this is to simulate sensors that are similar to robotic sensors. It has the same 31 categories with five objects per category and a total of 795 images.

The second main part of this dataset is from [37] and is called *Caltech-256*. In comparison with the other image collections, the *Caltech-256* dataset is much larger. It has an average of 119 images per category and 257 categories with a total of 30607 images. They are crawled from Google and PicSearch, but duplicates are deleted. [37]

The Scale Invariant Feature Transform (SIFT) approach transforms images into scale-invariant coordinates relative to local features which are 'summarized' in key point descriptors. [58] With this, images with over 15 similar SIFT descriptors are removed. The images are sorted in a category itself by the amount of how good one represents the category. The Range goes from "Good: A clear example of the visual category" to "Not Applicable: Not an example of the object category." [37, p.3]

To get an overall collection of the four image sets, which are considered as domains, categories with the same description are taken. From the 31 and 256 categories ten similar categories are extracted: backpack, touring-bike, calculator, head-phones, computer-keyboard, laptop-101, computer-monitor, computer-mouse, coffee-mug and video-projector. They are the class labels from one to ten. With this, a classifier should be trained on the training domain and should be able to



Figure 5.2: Examples Images from the four Image Datasets Caltech10, Amazon, DSLR and Webcam. Transfer learning is used, for example, to learn a classifier with Caltech-256 and test the model on Amazon.[35]

classify the test image to the corresponding image category. The final feature extraction is done with Speeded Up Robust Features (SURF) and encoded with 800-bin histograms. Finally, the twelve sets are designed to be trained and tested against each other by the ten labels.[35]

The SURF Algorithm uses integral images for images convolutions and uses existing detectors and descriptors and reduced the used methods to the essentials.[85] In figure 5.2 there are some example images taken from the four datasets of the category computer monitor. As in 5.1.1, some key figures from the dataset are pointed out in table 5.2. Note that for clarity, not the twelve datasets are shown, but rather only the 'Caltech vs. others'. The remaining sets should be similar.

## 5.2 Study Procedure and Settings

In this part, the data sampling, parameters and the used kernels are introduced. Every transfer learning method, which is used in the study is already introduced in chapter 3. Note that for a fair comparison the heterogeneous methods are not included in the study because they solve a different problem, which is described above in section 3.6. The basic task in this study is classification with data from the same feature space.

The SVM and the Probabilistic Classification Vector Machine are the baseline classifier in this study. This means that the transfer solutions JDA, TCA and TKL are

using the SVM, as underlying classifier. Therefore, in this chapter the transfer learning abbreviations are referring to the transfer learning method *and* the SVM. If the abbreviation SVM is used, it is the SVM without any transfer learning 'ability'. The PCVM is described in section 2 and is the baseline algorithm of the PCTKVM.

### 5.2.1 Data Sampling

The samples for testing the classifiers are drawn with a five times two-fold cross-validation manner. However, to obtain the differences from training- to test-set and the resulting transfer learning attributes, a sample is not drawn from the dataset as one, but rather obtaining the different domains and are drawn from one, only for training or testing. This means for example, the two image sets *amazon* and *webcam* are not merged. The training set is sampled from *amazon* and the test is from *webcam*. If the sampling is not done in this way, then the comparability of this study with another study would be lost. This is a suggested standard method for cross-validation in transfer learning.[35] Furthermore, the datasets are normalized via z-scored. This can be obtained from equation 4.21 in section 4.3.1.

### 5.2.2 Kernel

A kernel in this work is a symmetric matrix, in the following  $\mathbf{K}$ , which has to be positive semi-definite and therefore satisfies the Mercer conditions and integrates a similarity (dissimilarity) measure. It is an inner product of  $N$  training patterns, represented as vectors, which is a  $N \times N$  matrix:

$$\mathbf{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \quad (5.1)$$

With  $\phi(\cdot)$  as explicit feature mapping function. Because of its symmetry  $\mathbf{K}(\mathbf{x}, \mathbf{x}') = \mathbf{K}(\mathbf{x}', \mathbf{x})$ . In its simplest form, the linear kernel, the function becomes  $\phi(\mathbf{x}) = \mathbf{x}$ , which results in the simple vector dot product  $\mathbf{x}\mathbf{x}^T$  as similarity measure. There are various kernels and techniques to construct it. For example, if one replaces the inner product with some metric, e. g., Euclidean distance, then the kernel functions are referred as basis- instead of feature mapping functions.[8, p. 291-296, 329]

In this study almost all kernel machines are using a modified version of the Radial Basis function (RBF)- or Gaussian-Kernel. The 'well known' kernel can be obtained from [86, p. 17] and has the form:

$$k(\mathbf{x}, \mathbf{x}') = \exp \frac{d(\mathbf{x}, \mathbf{x}')^2}{2\theta^2} \quad (5.2)$$

Where  $d$  is the Euclidean distance and  $\theta$  is the width of the Gaussian kernel. The used one is suggested from [26] and 'replaces' the  $\theta$  parameter with  $\frac{1}{A}$  and  $A$  as

median over the samples:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{A} \|\mathbf{x} - \mathbf{x}'\|^2) \quad (5.3)$$

Note  $\gamma = \frac{1}{A}$  is a used abbreviation in [57]. There are two classifiers with different kernels. First, the GFK uses the implementation of an own supervised  $k$  nearest neighbor approach which can be obtained from the website of the author, refereed in section 3.5.2. Secondly, the PCVM, because in the cross-validation state for determining the parameters, the performance drops dramatically with the use of (5.3). Therefore, the PCVM uses the proposed RBF-Kernel from [19].

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\theta^2}) \quad (5.4)$$

The rest is trained with the suggested one from (5.2), although they are used to train with (5.3), which is obtained by the source code from the methods. However, the TKL approaches are using it initially. This is done to compare the pure transfer performance, as independent as possible.

### 5.2.3 Parameters

The cost Parameter  $C$  of the SVM, which controls the trade-off between margin and the ability to adapt the decision boundary to points near to it[84, p. 421-422], is set to 10. Finally, the LibSVM implementation, which can work with pre-calculated kernels, is used. Kernels are described above.

The PCVM as baseline Classifiers has only one parameter  $\theta$ , which is set to 1 for both sets. Furthermore, *niter* is set to 600 because we observed that the PCVM often terminates somewhere between 500-600 iterations. The threshold is set to 0.001.

The parameters for the transfer learning methods are partly obtained from the corresponding papers or with cross-validation. The PCTKVM and TKL algorithms are mainly using the eigenvalue dumping factor  $\xi$  which is set to two for the text datasets and 1.1 for the image datasets.[57] Furthermore, the PCTKVM uses the new kernel with  $\gamma = \frac{\theta}{A}$  in (5.3). Because of that, there is another parameter  $\theta$  which is set to two for the text sets and one for the image datasets. When it comes to the theta estimation for PCTKVM <sub>$\theta_{Est}$</sub>  described in section 4.3.1 the  $\theta$  value is determined for every dataset respectively. The estimated thetas are summarized in table 5.3.

JDA has two model parameters. First the number of subspace bases  $k$ , which is set to 100 and the regularization parameter  $\lambda$  which is set to one for both.[56]

For the GFK solution the parameter, number of subspace dimensions, is evaluated with cross-validation from  $k = \{1, 5, 10, 20, \dots, 100, 200\}$  and finally set for the text sets to 40 and to 50 for the image sets.

The TCA has also one parameter which gives the subspace dimensions and are determined from  $\mu = \{1, 5, 10, 20, \dots, 100, 200\}$  and is set to  $\mu = 50$  for both datasets.

Dataset	Theta	Dataset	Theta
Org vs. People	1.415	C vs. D	2.187
Org vs. Place	2.224	A vs. W	2.152
People vs. Place	2.187	A vs. D	2.166
C vs. A	2.224	D vs. W	2.163
C vs. W	2.187		

Table 5.3: List of the estimated thetas based on text and image datasets. The theta estimation is symmetric in terms of training and test dataset, i.e. C vs. A produces the same theta as A vs. C.

The goal of the parameter determination procedure was to identify the best performance concerning the parameters.

### 5.3 Performance Metrics

The performance of the transfer learning classifier has to be determined and should be compared with others. There are several metrics which can be used to measure the performance of a classifier.

However, before these are defined, there are some basic definitions to make, which are applied to the metrics in the following sections. These definitions are helping to get an idea of the location- and distribution characteristics of the sample.[78, p. 216-217] First, the mean which gives the average value of samples.

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad (5.5)$$

This samples can be for example one of the key figures of one classifier for the 18 datasets. The standard deviation is formulated to measure how the samples are distributed around the mean.

$$s^2 = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (5.6)$$

In both (5.5) and (5.6),  $N$  is the number of samples.

The null hypothesis in this context has the assumption that two classifiers are equal. If not this hypothesis is rejected.[4] To determine, whether the null hypothesis can be rejected or not, is the goal of the statistics from 5.5.1 and 5.6.3. However, before this decision can be made, the performance of an algorithm has to be captured. In this thesis four key figures are used. For clarity, the key figures are represented in the result table in percent. This differs from the calculation presented here. The selection of the key figures is influenced by [19].



### 5.3.1 Area under a ROC Curve

The first metric is the Area under a ROC Curve (AUC) which measures exactly what the name says.[29, p. 13]

A Receiver Operating Characteristics (ROC) Curve is a technique for visualizing and organizing classifiers based on their performance. This graph is a two-dimensional plot of the true positive rate on the Y-axis and the false positive rate on the X-axis. It illustrates the trade-off between the benefits and costs. The more the graph heads north-west, the better is the classifier. Note that if a classifier has a low ROC curve near the X axis, then it may be conservative in making an assignment to a positive class because it makes it only with strong evidence. That means it makes few false positive errors, but they often have a low true positive rate.[29, p. 4]

In this thesis, when it comes to two classes, a low ROC curve is interpreted as a lower performance, because the true positive points have a higher probability to be wrongly classified concerning the negative label.

Because the AUC is an area in a unit square, it has a value between zero and one. The statistical interpretation is as follows: "The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance." [29, p. 13-16]

Concerning a two-class problem, the AUC tells us the probability for a member of the positive class and the negative class that the positive one is correctly classified to the positive class and the other respectively.

In this thesis, the build-in MatLab function  $[X, Y, AUC] = \text{perfcurve}(LABELS, SCORES, POSCLASS)$  is used. The function needs the ground truth 'LABELS' and the probabilistic estimate that this point actual belongs to the positive class ('SCORES') determined by a classifier. With 'POSCLASS' the positive class is specified. In this thesis, '1' is always the positive class. It gives the arrays **X** and **Y** back to create the ROC curve as MatLab plot. The function output 'AUC' is the needed value for comparison. Note that with this definition of AUC, it makes only sense for two class problems. For the multi-class problems, the accuracy is more insightful, but the AUC is calculated for completeness with '1' as positive label and the rest of the labels as negative.

### 5.3.2 Accuracy and Error

The accuracy is another used metric with:[57]

$$Accuracy = \frac{|x : x \in \mathcal{X} \wedge f(x) = y(x)|}{|x : x \in \mathcal{X}|} \quad (5.7)$$

With  $f(x)$  as the given label from the classifier and  $y(x)$  as the ground truth label. In other words, it describes how many of the test points are getting the correct labels

from a classifier assigned, concerning all points.

Another way to interpret the accuracy is that it is the relation of the true positive rate and true negative rate to all samples. The accuracy goes from zero to one because it is normalized.[29, p. 3] In this thesis, the metric Error (ERR) is simply calculated with  $Error = 1 - Accuracy$ .

### 5.3.3 Root Mean Squared Error

The last metric is the Root Mean Square Error (RMSE). It takes the root of the squared mean errors.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^n e_i^2} \quad (5.8)$$

With  $N$  as the number of errors which relies on the number of datasets. The RMSE assumes that the errors are normal distributed and unbiased. It is per definition always higher than the simply Mean Absolute Error (MAE). With that the RMSE tends to become increasingly larger in comparison with the MAE, as the distribution of the error magnitude becomes larger. The RMSE penalizes the variance of errors by weighting larger absolute errors more than smaller ones. Note that the RMSE is sensitive to outliers. This key figure is another idea trying, to describe the error distribution. [16]

## 5.4 Descriptive Statistics

In this section, the general performance of the classifiers is shown. As part of the descriptive statistics, the mean and the standard deviation are used.[43]

With that the different key figures over the datasets are summarized. The performance, in general, is divided into datasets. The full result as well as the remaining plots of performance can be considered in appendix B.

### 5.4.1 Metric Results

In this subsection, the performance over the metrics, which are described earlier, is shown. In general, according to the MMD's and KLD's for the 18 datasets, classification of an image datasets is a more challenging problem in comparison with the Reuters dataset. The results and the number of support vectors are summarized in table 5.4. The comparison of the number of support vectors is discussed in section 5.4.3. Note that this data is produced on the fly in the 5x2 cv F test. Therefore, the shown results are the mean over ten repetitions of cross-validation. The standard deviation is shown in brackets. The results are split up into two dataset categories

ERR	SVM	PCVM	PCTKVM <sub>offEst</sub>	PCTKVM	TCA	JDA	GFK	TKL
Reuters	31.82(2.02)	34.19(3.65)	<i>30.03</i> (6.63)	<i>26.12</i> (2.97)	30.87(3.24)	32.95(2.54)	34.53(2.74)	<b>22.70</b> (2.17)
Image	61.51 (2.75)	68.13(3.11)	<i>63.53</i> (3.48)	<i>62.64</i> (3.35)	59.56(3.33)	59.39(3.32)	64.24(2.96)	<b>58.25</b> (7.63)
RMSE	SVM	PCVM	PCTKVM <sub>offEst</sub>	PCTKVM	TCA	JDA	GFK	TKL
Reuters	31.88 (8.43)	34.48(5.93)	<i>30.77</i> (10.35)	<i>26.28</i> (9.32)	31.03(9.00)	33.05(8.81)	34.63(6.28)	<b>22.81</b> (7.63)
Image	61.59(14.71)	68.23(6.88)	<i>63.64</i> (9.15)	<i>62.76</i> (9.35)	59.67(14.76)	59.50(14.81)	64.33(15.39)	<b>58.36</b> (13.63)
AUC	SVM	PCVM	PCTKVM <sub>offEst</sub>	PCTKVM	TCA	JDA	GFK	TKL
Reuters	69.00 (2.33)	70.96(3.99)	<i>76.47</i> (8.63)	<i>79.36</i> (4.82)	72.32(3.78)	69.11 (3.13)	-	<b>83.46</b> (2.26)
Image	74.04 (5.25)	65.93 (8.13)	73.69 (8.63)	74.61 (6.20)	69.77 (8.27)	<b>76.92</b> (6.41)	-	74.61(6.2)
N. SV.	SVM	PCVM	PCTKVM <sub>offEst</sub>	PCTKVM	TCA	JDA	-	TKL
Reuters(1153.66)	482.35 (20.96)	46.93 (20.96)	<b>3.02</b> (1.14)	<b>3.02</b> (0.22)	182.70 (23.76)	220.28 (26.25)	-	190.73 (46.63)
Image(633.25)	309.90(210.89)	68.40 (44.76)	<b>50.02</b> ( <b>27.73</b> )	<i>54.88</i> (31.73)	253.30 (170.57)	287.433(193.76)	-	382.53 (198.04)

Table 5.4: The results of the transfer learning classifiers with cross-validation under the four metrics on the 18 datasets. The values are showing the mean over 10 runs and the standard deviation in brackets. Our approaches are in italic and the winning classifier is bold. The second part of the table shows the mean number of support vectors.

Image and Reuters, respectively. Furthermore, the four metrics error, RMSE and AUC are applied. The Accuracy (ACC) can simply be determined with the error. The classifiers, which are created in the course of this work are in italics. The performance of the best classifier is bold. The GFK implementation supports no probabilistic output and therefore has no AUC value in general. Although that the AUC is only defined for two classes, we also show the results of AUC for images. The positive class is set two one and the negative class are the remaining classes (2-10). Note that through a cross-validation on the 18 datasets, the means metrics are combined from 180 performance results. Furthermore, the mean error over Reuters and image datasets for the SVM, TKL and the PCVM's are shown in figure 5.3. Note that the TKL is excluded in the second image (5.3b) for clarity. The error graph over the whole datasets and the remaining classifier are shown in B.

It can be observed that TKL has the overall best performance. In general, and according to section 3.7, it can be seen that negative transfer happens between the baseline SVM and the transfer solution GFK. However, we can see that the PCTKVM achieves an *overall* better performance result than the PCVM in this test. Furthermore, the PCTKVM can generally achieve a better performance at text sets than the remaining transfer learning solutions and the SVM.

The performance of the new PCVM's at image datasets is comparable with the result of the SVM, which is also observable in 5.3b. Nevertheless, this is also observable for the other transfer learning solutions, which are not very better as the SVM. The performance of the SVM is always better as from PCVM at images.

We assume that this happens because the datasets are very small and the classes are not balanced because we can rather obtain a good performance on the larger more balanced text datasets in comparison with the SVM. This assumption may be more

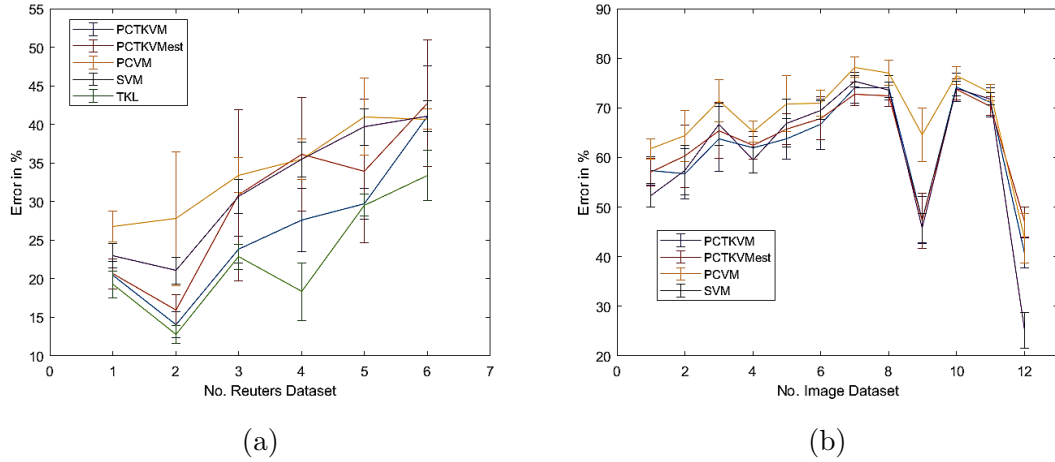


Figure 5.3: The plot of mean errors with standard deviations of the cross-validation test. The left shows the result on Reuters and the right shows the result on images. A graph shows the error and a vertical bar shows the standard deviation. The no. of datasets are the order of datasets. In the right figure the TKL is omitted for clarity.

valid when one takes a look at the comparison, which involves the whole dataset, which is shown in table B.7. It seems that the SVM can handle very small datasets better than the PCVM in general. However, again referring to the detailed test results, it states that the PCTKVM is often the second best classifier. The PCTKVM does a better job as the other transfer learning classifiers, except TKL. Additionally, we can see that the SVM baseliner has an overall better performance than the PCVM baseliner. Apart from the size of datasets, it might be a better idea for transfer learning to find a hyperplane, which separates the classes best, rather than modeling it via probabilistic, to handle the difference in marginal probability distributions.

We can observe that the  $PCTKVM_{\theta_{Est}}$  is in general a little bit worse than the PCTKVM. Because these two approaches only differ in the  $\theta$  value, we can say that the  $\theta$  estimation approach described in section 4.3.1, does not work better rather than a fixed theta, obtained with a parameter grid search. However, we can avoid this grid search and the difference in performance is not really large. As we will see in the statistical tests, the difference isn't significant. In fact, the estimation was originally designed for regression and classification [48]. However, it seems that the value of the estimation is too 'smooth' and the probabilistic estimation for one class are 'overlapping' in the other classes space. This space, would one intuitively assign to the other label. This might be similar to the problem of the RVM from [19], where it assigns a negative weight in the heart of the positive class.

The essence of the standard deviation in table 5.4 states the overall performance range in the datasets, which fluctuates concerning the composition of the datasets. Therefore, to make a valid assumption on the standard deviation concerning the

classifiers, we again have to look in the detailed result table B.2. As pointed out in the results of Chen et al. in [19] and with our results again, the range of standard deviation of the PCVM is not that kind of a problem. However, when it comes to the PCTKVM $_{\theta_{\text{Est}}}$ , we see that the range of standard deviation, especially for the text datasets, is higher in comparison with the deviation of the remaining classifiers. Since PCTKVM and PCTKVM $_{\theta_{\text{Est}}}$  only differ in the theta value, we can say that the fluctuation of performance in this test also depends on the theta. The fact that all PCVM's are randomly initialized is not critical because we can observe that the standard deviation is not greatly higher in comparison with the other classifiers from table B.2.

## 5.4.2 Computational Complexities

According to [19], one might compare the actual computational time in seconds. However, these results are highly subjective because of differences in the implementation techniques and programming languages. For example, the LibSVM is completely based on C++ only, and the PCTKVM is based on MatLab. Besides the differences in theoretical complexity, the TKL with the LibSVM will therefore always be faster than the PCTKVM. Therefore, we will compare the complexity of the classifiers, rather than the actual computation time. However, some of the results discussed here, are also valid for the actual time measurement. The discussion of the computational time can be found in appendix B, section B.1.1.

Note that the number of dimension  $D$  is omitted in the comparison because every classifier has to deal with it. This means not that the dimensions can be neglected. The complexities of the algorithms are summarized in table 5.5. Note that in this table the methods are combined with the SVM. The complexity of the SVM is  $\mathcal{O}(N^3)$ , with  $N$  as the number of training basis functions.

We assumed in chapter 2 that the complexity of the PCVM is  $\mathcal{O}(B^3)$ , with  $B < N$ . However, this assumption cannot be held for transfer learning because the convergence rate to lower the number of non-zero weight vectors is very slow. Furthermore, because of the slow convergence rate and the slow prune of zero weight vectors, the complexity of the optimization problem in 2.20 should be mentioned. It uses a conjugate gradient descent algorithm with the complexity of  $\mathcal{O}(N\sqrt{V})$ , with  $N$  as training data and  $V$  as the number of variables.[75, p. 37-38] This (slow) process will be repeated  $I$  times until convergence. Because in practice this takes a lot of time, we estimate the computation complexity of the PCVM as  $\mathcal{O}(I(N^3 + N\sqrt{V}))$ .

When it comes to the PCTKVM's, we can stick to the previously discussed complexity of  $\mathcal{O}(N^3 + DM^2)$  because we removed the optimization problem and the convergence of the weight vector is steady. The TKL has the complexity of  $\mathcal{O}(R(N + M)^2)$ , with  $R$  as the number of kept eigenvalues. Because  $R$  is always upper bounded with 500

Algorithm	Complexity	Algorithm	Complexity
SVM	$\mathcal{O}(N^3)$	TCA	$\mathcal{O}(N^3 + LM^2)$
PCVM	$\mathcal{O}(I(N^3 + N\sqrt{K}))$	JDA	$\mathcal{O}(N^3 + LM^2)$
PCTKVM	$\mathcal{O}(N^3 + M^2)$	GFK	$\mathcal{O}(\mathbf{K}^2 + \mathbf{NC})$
PCTKVM <sub><math>\theta_{\text{Est}}</math></sub>	$\mathcal{O}(N^3 + M^2)$	TKL	$\mathcal{O}(N^3 + M^2)$

Table 5.5: This table shows the comparison of the computational complexity of the algorithms used for this study. Note that the complexity of the baseline method is always included. The number of dimensions  $D$  is omitted for clarity because every algorithm has to deal with it. This means not that  $D$  can be neglected. The fastest complexity is bold and given from GFK.

and it is the same for the PCTKVM,  $R$  is omitted.

By incorporating the complexity of a SVM, we obtain the complexity of the TKL-SVM with  $\mathcal{O}(N^3 + M^2)$ .

The computational complexity of TCA is given by  $\mathcal{O}(L(N + M)^2)$ , where  $L$  is the number of subspace dimensions. Again with the SVM and making the assumption that  $L \ll N$  we obtain  $\mathcal{O}(N^3 + LM^2)$ .

When it comes to JDA the bounded parameters are  $T \leq 50$  as the number of iterations until convergence, the subspace bases with  $L \leq 500$ . The computational complexity, with SVM, is given by  $\mathcal{O}(N^3 + LM^2)$ .

The GFK approach has  $\mathcal{O}(K^2L + K^2)$ . Moreover, it uses a kernalized  $k$ -nearest-neighbor approach for predictions which has  $\mathcal{O}(NC)^3$ , with  $C$  as number of classes and therefore we can obtain  $\mathcal{O}(K^2 + NC)$  as complexity for GFK.

At the beginning of this chapter, we outvoted the time comparison results because they are very subjective. However, keeping in mind the different programming languages and the assumptions throughout this section, one could see that the order of the fastest algorithms in table B.5 is around the order of the complexity estimation.

### 5.4.3 Number of Support Vectors

The number of support vectors is an indicator for the complexity of a model. Note that although we defined for the RVM and PCVM, that the model vectors are relevance vectors, we will stick to support vectors just for simplicity.

Many support vectors are leading to a low training error rate, but with the growing complexity, there may be more test errors. This happens because of the bad generalization abilities of the model.[43, p. 81]

Another explanation for a large number of support vectors is that the dataset is com-

<sup>3</sup><https://www.ismll.uni-hildesheim.de/lehre/ml-07w/skript/ml-2up-03-nearest-neighbor.pdf>

plicated. For example, if a dataset tends to be inseparable.[1, p. 78;86] However, in testing the transfer learning algorithms on the specific dataset may a different view arises. Thus the model complexity is large for the transfer learning methods with the SVM classifier, the classification performance is not that bad in comparison with the sparse PCTKVM or PCVM, referring to table 5.4, 5.7 and 5.9. However, we want to point out that in general, lesser support vectors in a model leads to more compact models.[8, p. 349]

This is the strength of the PCTKVM <sub>$\theta_{Est}$</sub>  and PCTKVM, although the sparse model of the PCVM has a worse performance, our solutions are obtaining relative sparse models by maintaining a competitive performance on difficult datasets. The average number of support vectors needed by a classifier is split up in dataset categories and is summarized in table 5.6. The mean number of the datasets are shown on the right-hand side of the name in brackets.

N. SV.	SVM	PCVM	PCTKVM <sub><math>\theta_{Est}</math></sub>	PCTKVM	TCA	JDA	TKL
Reuters(1153.66)	482.35	46.93	<b>3.02</b>	<b>3.02</b>	182.70	220.28	190.73
Image(633.25)	309.90	68.40	<b>50.02</b>	<i>54.88</i>	253.30	287.433	382.53

Table 5.6: The mean of support vectors of a classifier for Reuters and Image datasets. The numbers on the right side of the datasets are the mean number of points over the six text sets and the twelve image sets. This is the result of 60 runs on text sets and 120 runs on image sets.

## 5.5 Comparison over one Dataset

The 5x2 cv F test is a statistical tool to compare the performance of supervised classification learning algorithms on one dataset.[19] The first approach, the 5x2 t-test, was designed by Dietterich in 1998. [25] The improved 5x2 cv F test should have less Type I error (wrongly reject the null hypothesis) and a higher power (probability of detecting a difference if it exists) than the proper.[4] It is in general designed to compare two classifiers. As mentioned in 5.2.1 the sample is drawn with the use of the cross-validation approach and randomly splits up the data in 2 parts, fold one and fold two. This is repeated five times. To determine whether a classifier is indeed better than another, the test statistic has to be calculated.

Suppose that  $p_i^{(j)}$  is the difference between the error rates of two classifiers on fold  $j = 1, 2$  and replication  $i = \{1, \dots, 5\}$ . The average of a replication is  $\bar{p}_i = (p_i^{(1)} + p_i^{(2)})/2$ . And the estimated variance is  $s_i^2 = (p_i^{(1)} - \bar{p}_i)^2 + (p_i^{(2)} - \bar{p}_i)^2$ . Then the value to

determine statistical significance in the differences of the errors is calculated as:

$$F = \frac{\sum_{i=1}^5 \sum_{j=1}^2 (p_i^{(j)})^2}{2 \sum_{i=1}^5 s_i^2} \quad (5.9)$$

This value is approximately F distributed. It has ten and five degrees of freedom. The null hypothesis that the classifiers have the same error rate is rejected with a confidence of 0.95 if the f is greater than 4.74.

An Idea of the F distribution can be obtained from [78, p. 338-340] Critical values from this distribution can be obtained from a statistic table, e.g., [11, p. 591]. Note that this explanation focuses only on the error metric. However, it should be possible to do this test with other metrics as well.

This is possible because the idea of this statistic is to determine if the metrics, e.g. error, of two classifiers, are drawn from the same distribution (null hypothesis).[4]

### 5.5.1 Result and Discussion

The result of the 5x2 cv F test is shown in table 5.7. With the descriptive statistic table from 5.4, we can not make a valid statistical evidence if the difference in performance is significant, i.g., we can reject the null hypothesis.[45, p. 9] We can only observe trends and assume significance. Therefore we will proceed with this test.

In this table, the cells are showing the percentage of wins for PCTKVM against a listed classifier. A win is determined by comparing the performance of the two folds (10 runs). If there are more wins out of the ten runs for one classifier, then it is a win for it. If the wins and losses are equal, then it is a tie. That means a tie does not represent a real equal performance, but rather that the wins and losses are even. The amount of wins in percent is shown in the rows Reuters and Image. The percentage of ties is shown in the same row in brackets. The row Significant is showing the percentage of the significant wins out of all wins. The amount of significant wins of the competitor are shown in brackets.

The results are showing, that the PCTKVM does a decent job at the Reuters dataset. In fact, it wins 100% of all test runs against the baseliners and the transfer learning solutions, except against the TKL, partly even half of the wins are significant.

The PCTKVM can achieve a good performance in comparison to the baseline PCVM, with 67 % significant wins out of all wins, which are 100% of all test runs. The performance gap is not that large at images, but sill our approach wins all runs on the twelve datasets with 8% significant wins.

Compared to the PCTKVM <sub>$\theta_{Est}$</sub> , it wins 67 % of the test runs and 17 % ties. When it comes to image dataset the performance, between the two, is more balanced. We assume that this could have two reasons: One might argue that the  $\theta$  for the image



Wins in %	SVM	PCVM	PCTKVM <sub><math>\theta_{\text{Est}}</math></sub>	TCA	JDA	GFK	TKL
Reuters	<b>100</b>	<b>100</b>	67(17)	100	<b>100</b>	<b>100</b>	0
Significant	<b>50</b>	<b>67</b>	0	33	<b>50</b>	<b>67</b>	0
Image	33(17)	100	25(42)	25	25 (25)	50	8(8)
Significant	0(8)	8	0(8)	8(17)	0(25)	0 (17)	0(33)

Table 5.7: The result of the 5 x 2 cv F test to compare classifiers on one dataset under the Error metrics. The first row per dataset comparison shows the percentage of wins of the PCTKVM against the listed classifier. The percentage of ties is shown in brackets. In the row Significant the percentage of significant wins out of all wins is shown. The percentage of significant wins out of all wins from the listed classifier is shown in brackets. Test-runs with more than 50% significant wins for the PCTKVM are bold.

is not well selected, but it is selected with a parameter grid search to determine the best performance. Another reason could be that the  $\theta$  estimation does a better job when it comes to images or takes values that are not considered in the parameter optimization.

Considering the performance with the image dataset with respect to listed classifiers, the baseline SVM wins 66% of all test runs against the PCTKVM, 8% of them are significant. Moreover, half of the test runs are ties. With that we can say that the SVM and the PCTKVM are comparable at images. The same is valid for the SVM and the PCTKVM <sub>$\theta_{\text{Est}}$</sub> . A similar result is observed with the remaining transfer learning classifiers. Note that the performance between all classifiers, except the TKL and PCVM, are more balanced. As already discussed in section 5.4.1, a reason for this might be small datasets.

However, going back to the text results, we can observe a good performance of the PCTKVM because it wins 100% of the runs against the GFK, TCA and JDA, with many significant wins. Note that this statistic has ten and five degrees of freedom.

## 5.6 Comparison over multiple Datasets

Testing algorithms over only one dataset has a certain risk. The problem with a high amount of tests like it is done in section 5.5 is that a certain proportion of the null hypotheses are rejected due to random chance. This issue is a well known statistical problem. The goal is to control the family-wise error: The probability of making in any of the comparisons at least one Type 1 error. The Friedman test can be used as a statistic to compare supervised classification learning algorithms

over multiple Datasets. It is first introduced, as the name implies, by Friedman in 1937. It is a non-parametric equivalent of the repeated-measures of Analysis of Variance (ANOVA).[45, p. 9-10]

### 5.6.1 Analysis of Variance

The ANOVA, briefly, compares the variability between the classifiers, variability between datasets and the remaining error variability. Is the between classifier variability significantly larger than the error variability, the null hypothesis can be rejected with the conclusion that there are differences between the classifier. However, ANOVA makes some assumption that cannot be held for the comparison of classifiers and therefore the Friedman test is chosen. For example, ANOVA has the assumption of sphericity, and this is not guaranteed for comparison of multiple classifier and datasets.[45, p. 10]

Sphericity is the condition that the difference between samples is homogenate. If not the ANOVA F value will be too large.[41, p. 152]

### 5.6.2 Friedman Test

The Friedman test ranks the classifiers for each data separately. The best one has rank 1, the second has rank 2 and so on. The average rank is assigned in case of a tie. With  $r_i^j$  as the  $j$ -th rank for one algorithm out of  $k$  on the  $i$ -th of  $N$  datasets. Moreover, with  $R_j = \frac{1}{N} \sum_{i=1}^N r_i^j$  as the average rank of an algorithm, the Friedman tests compares the average ranks. The null hypothesis has the assumption that all classifiers are equal and hence the average ranks  $R_j$  should be similar. The F value of Friedman is calculated with:[45, p. 11]

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[ \sum_{j=1}^k R_j^2 - \frac{k(k+1)^2}{4} \right] \quad (5.10)$$

It is distributed according to  $\chi_F^2$  (Chi-Square-Distribution). This distribution has  $k-1$  degrees of freedom. However, this takes only place if  $N$  and  $k$  are high enough. A solid numbers would be  $N > 10$  and  $k > 5$ . [45, p. 11]

We are about the 'solid' numbers with eight classifiers and 18 datasets. However, Iman and Davenport showed in [44] that the Friedman  $\chi_F^2$  value is needless conservative and proposed a better statistic which is determined with:[45, p. 11]

$$F_f = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} \quad (5.11)$$

This statistic is F distributed with  $k-1$  and  $(k-1)(N-1)$  degrees of freedom. The Friedman test only determines if there is any significant difference at all. If the

null hypothesis can be rejected (significant difference), then we can proceed with the post-hoc test to determine which classifiers are significantly different. Significance exists, if the F value from (5.11) is equal or above the critical values, for a confidence level of 0.95 from the F distribution.[45, p.11]

In this article, the Bonferroni-Dunn test is used for the post-hoc test as it is used in [19]. Finally, we can reject the null hypothesis and proceed with the post-hoc test. The performance of two algorithms is significantly different if their average ranks differ by at least the critical difference. This difference needs the parameter  $q_\alpha$  and is set to 2.690, because of eight classifiers and the confidence level of  $\alpha = 0.05$ . So the confidence levels of Friedman and 5x2 cv F test are the same. The critical difference is determined with:[45, p. 11-12]

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \quad (5.12)$$

Note that the Friedman test and the post-hoc test is implemented on our own because it seems that MatLab<sup>4</sup> provides only the first part of the test (original Friedman), which is the Friedman test without the described improvements from Iman and Davenport.

### 5.6.3 Result and Discussion

The Friedman test results will be covering the 18 datasets and are discussed in the following. The result is divided into two tables. The first 5.8 is showing the mean ranks under the performance metrics. This is done by ranking the mean of performance according to one dataset and results in 18 ranks per classifier. A tie is assigned if the means of two classifiers are equal.

The second table 5.9 shows the Friedman test result and the corresponding post hoc test. It lists the results of Friedman test with the corresponding post hoc test. The critical difference is calculated according to equation (5.12). These key values are showing the difference of mean ranks against the mean rank of the PCTKVM.

To see if a classifier has a significant difference in performance a key value in the table has to be at least the critical difference. For clarity, it shows not the absolute rank difference, but rather with the corresponding sign. That means for error and RMSE a classifier is better if the rank difference is negative and worse if it is positive. For AUC and ACC, it is vice verse. The bold cells are indicating any significance. Note that one could draw further results out of the tables by just taking the mean ranks of two classifiers from table 5.8, subtract them and compare the difference against the critical difference.

The statistic has  $7 \times 119$  degrees of freedom for error, RMSE and accuracy and  $6 \times 102$

---

<sup>4</sup><https://de.mathworks.com/help/stats/friedman.html>

Mean Ranks	SVM	PCVM	PCTKVM <sub><math>\theta_{Est}</math></sub>	PCTKVM	TCA	JDA	GFK	TKL
ERR	4.5	6.94	4.94	3.78	3.39	4.11	6.56	1.78
RMSE	4.33	6.94	4.94	3.89	3.39	4.22	6.5	1.78
AUC	3.39	2.94	4.33	4.94	3.22	4	-	5.17
ACC	4.5	2.06	4.06	5.22	5.61	4.89	2.44	7.22

Table 5.8: The mean rank of the tested classifiers under the Metrics ERR, AUC and RMSE. The means are based on 18 mean ranks concerning the Reuters and image datasets. Every mean rank consists of 10 test runs sampled with two fold cross validation.

degrees of freedom for AUC.

With the results of table 5.9, we can say that the performance of PCTKVM is significantly better in three out of four metric categories than the PCVM. The same statement is valid when it comes to GFK.

In comparison with the PCTKVM <sub>$\theta_{Est}$</sub> , we can say that the PCTKVM is in general better but not significantly. In fact, the SVM is a little bit better (not significantly) than the PCTKVM <sub>$\theta_{Est}$</sub>  and is always better as the PCVM in the overall comparison. It is worth mentioning, that in this study the number of the image dataset is twice the number of text sets. It is obvious that the SVM can handle the tested images well. Nevertheless, one would say that the PCTKVM <sub>$\theta_{Est}$</sub>  and PCTKVM could be better in studies with more text datasets.

In section 5.4.1, we discussed the fact that the SVM is comparable with the PCTKVM at images. However, the Friedman test shows that this is not that critical overall, as in general, the PCTKVM is better.

By comparing TKL with the PCTKVM, we can observe that the first is indeed better, almost but not significant. The performance of the PCTKVM against JDA and TCA

Metrics	Friedman test	CD <sub>0.05</sub>	SVM	PCVM	PCTKVM <sub><math>\theta_{Est}</math></sub>	TCA	JDA	GFK	TKL
ERR	0.00	2.20	0.72	<b>3.16</b>	<i>1.16</i>	-0.39	0.33	<b>2.78</b>	-2
RMSE	0.00	2.20	0.44	<b>3.05</b>	<i>1.05</i>	-0.5	0.33	<b>2.61</b>	-2.11
AUC	0.01	2.20	-1.55	-2	<i>-0.61</i>	-1.72	-0.94	-	0.23
ACC	0.00	2.20	-0.72	<b>-3.16</b>	<i>-1.16</i>	0.39	-0.33	<b>-2.78</b>	2

Table 5.9: The result of the Friedman test with Bonferroni-Dunn as post-hoc Test to compare classifiers over multiple Datasets. The first column shows the Friedman value, the second the critical difference which is subtracted from the mean ranks. The classifier columns are showing the result of the post-hoc test. Bold differences are significant.

is comparable. Furthermore, it appears that aligning the conditional probabilistic distributions are no guarantee for a good transfer learning performance concerning JDA. However, this is the main extra effort or difference, which is done by JDA in comparison with the TCA and the PCTKVM: Aligning the conditional and marginal probability distribution instead of just aligning the latter.[56][65][57]

Another important fact which arises from the result is that the SVM, by incorporating the kernel from section 5.2.2, is very competitive to the transfer learning solutions. Note that we do not say, that the kernel causes the good performance, but we want to point out the difference to the suggested kernel setting of the LibSVM<sup>5</sup>. Only the TKL can achieve a significantly better performance in comparison with the SVM. Furthermore, the differences of PCTKVM <sub>$\theta_{Est}$</sub>  and the TKL are significant.

## 5.7 Conclusion

Summarizing, we showed that by integration of transfer learning the performance improves significantly. Furthermore, our approach can compete with current transfer learning solution. In fact, the PCTKVM is clearly better for text set as JDA, TCA and GFK. It is only worse in comparison with the TKL. Moreover, it is significantly better than the PCVM and in general better than the SVM

We tested it with an extensive study over 18 dataset compositions of two data categories, which are common in the testing of transfer learning, and by finally incorporating two statistical procedures to determine significance.

Based on the test result we can summarize the advantages of the PCTKVM with: The fact that the PCTKVM is probabilistic. We can estimate the uncertainty of predictions through the algorithm. This is very rare for transfer learning solutions, according to [88], because none out of 18 reviewed methods produce a reliable probabilistic output. Because, the probabilistic output of an underlying SVM is considered as unreliable[81] The output of a PCTKVM is.[19] Furthermore, a PCTKVM needs no labeled test data, no huge amount of training data and only one source dataset for the transfer, which makes it well applicable to new datasets. On the other hand, a PCTKVM can handle the probabilistic estimation in a reasonable time and can create a sparse model by maintaining a good performance. Furthermore, by the use of the theta estimation, we can lower the effort of cross-validation, because a PCTKVM <sub>$\theta_{Est}$</sub>  has only the eigenspectrum dumping factor as a free parameter.

---

<sup>5</sup><http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.htm#f506>

# Chapter 6

## Evaluation and Future Work

This chapter is devoted to critically review the characteristics of the study and consider future work.

The study classifiers used in this thesis are primarily based on the kernel from (5.3). According to the corresponding articles, the TCA JDA approaches are using the standard Gaussian kernel from (5.2). [56][65] In there, we can naturally observe different performances, may better or worse. For more insights, whether the transfer learning performance is rather critical to the suggested kernel type, it would be useful to repeat the study with the suggested kernels from the corresponding articles. Furthermore, it should be compared to this study to determine the differences. Additionally, the ability to transfer knowledge of the SVM is comparable to the transfer learning solutions with the use of the new kernel, according to Friedman test from table 5.9. In a new study, we can verify, whether the SVM works comparably good with the standard Gaussian kernel or not. With that it can be seen if the SVM is, in fact, better for the transfer learning settings as the PCVM.

Another topic which is not included in this work, is the use of other datasets, to give a more general evidence, whether a transfer learning solution is better or not. For example the 20-Newsgroup<sup>1</sup> dataset, as a text-based dataset, as suggested in [66]. The reason why we did not include this dataset is that the GFK algorithm needs to calculate the null space of the dataset. This task is main-memory consuming and can not be done by the for the study available computers, which results finally in an 'out of memory' error from MatLab. We decided for the comparison with the PCTKVM that it would be better to include more transfer learning solutions rather than datasets.

Other examples for image transfer learning datasets are *USPS*<sup>2</sup>, *MNIST*<sup>3</sup>, *COIL20*<sup>4</sup>. [56][55] These are not included in the study. Again for a more general assumption

---

<sup>1</sup><http://qwone.com/~jason/20Newsgroups/>

<sup>2</sup><http://www-i6.informatik.rwth-aachen.de/~keyzers/usps.html>

<sup>3</sup><http://yann.lecun.com/exdb/mnist/>

<sup>4</sup><http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php>

on the performance, it would be useful. As well as including real-world scenarios like WiFi-localization or human activity classification as suggested in [88, p. 32]. Furthermore, as described in 5.4.1, a more analytic way to determine the width of the Gaussian kernel should be found. Moreover, the current estimation should be modified in a future work to improve the performance and reduce the 'smoothness' a little bit. A possible approach would be to incorporate another estimation for finding the theta for separated classes in the training set only. In this case, we could decide whether the theta over all sample is too large.

In the course of this study, we observed an overall good performance. However, we have been observed a mixed performance for small multi-class image problems. Furthermore, we can also observe a drop of performance by the *people vs. places* dataset, which is larger than the image sets. In future work, the reason for the mixed performance and the performance dips could be determined.

### EM-Based Transfer Learning

Another point worth mentioning is that the PCTKVM is based on a non-probabilistic transfer learning method. However, the PCVM are based on probabilistic assumptions, which is one of the achievements of Chen et al. In future work, the issue could be addressed to find a probabilistic transfer learning method. For example, as mentioned in section 3.4.1, the Inductive Transfer Learning is related to self-thought learning. With this, an idea would be to extend the current EM algorithm of the PCVM to learn the source and target domain simultaneously and transferring or modify the parameters from one into the other. With that we could extend the PCVM with transfer learning in a probabilistic manner. A recent approach for 'self-paced' learning which uses a latent variable model and the EM algorithm is a step in this direction and proposed in [51].

### Rotational Transfer Learning

Inspired by the ideas of GFK and TKL, we could formulate another transfer learning approach.

We make the assumption that the marginal distributions are aligned with the use of the z-score. With that the transfer learning problem would be reduced to aligning the conditional probabilistic distributions. This idea is based on the Single Value Decomposition, which can be used to decompose the rectangular source and target matrices with:[33]

$$\mathbf{Z} = \mathbf{U}_Z \mathbf{\Lambda} \mathbf{V}_Z \wedge \mathbf{X} = \mathbf{U}_X \mathbf{\Gamma} \mathbf{V}_X \quad (6.1)$$

Where  $\mathbf{U}$  and  $\mathbf{V}$  are rotation matrices and  $\mathbf{\Lambda}$  and  $\mathbf{\Gamma}$  are the singular values.

To do the transfer, one idea is to 'rotate'(approximate) the source data with the use

of the rotations matrices of the target data with:

$$\mathbf{Z} = \mathbf{U}_{\mathcal{X}} \mathbf{\Lambda} \mathbf{V}_{\mathcal{X}} \quad (6.2)$$

Note that with this solution, the size of samples has to be aligned, which is currently solved with a Latent-Semantic-Analysis. With that the training data could be extended with the structure information obtained from the target data. The error of this approximation can be bounded by  $E_{RT} = \|\text{tr}(\mathbf{\Lambda}) - \text{tr}(\mathbf{\Gamma})\|$ . A first test has been showed that the performance is outstanding for text sets, but has a worse performance on image datasets. Note that because this idea is not finished yet, the approach is not in the study. One can find the first version of the Rotational Transfer-Probabilistic Classification Vector Machine (RT-PCVM) in the source code of this thesis. In a further work, the design of this approach could be finished.

## 6.1 Extensions

Although, future works have already been mentioned in the course of this chapter, we want to point out some further extensions, which can help to make the PCTKVM applicable. These extensions are coming mainly from [57].

The current algorithm can be extended to multiple source learning. With that we can match the distribution difference of multiple source domains with the target domain. This can be implemented by learning a  $\mathbf{\Lambda}$  for every source/target combination and then using an existing multiple source learning, approach to predict the target based on multiple source domains. For example, one could use the Conditional Probability based Multisource Domain Adaptation (CP-MDA) algorithm from [17] for it.

Another suggested extension is to adopt the algorithm to 'big data'. Because, a kernel machine can have at least  $\mathcal{O}(N^2)$  or even worse, they may not be applied well to huge datasets. Here we can make use of the original Nyström method from 4.1 in combination with the Nyström kernel approximation from 4.1.1.[57]

Consider a large test kernel matrix  $\mathbf{K}_{\mathcal{X}} \in \mathbb{R}^{M \times M}$ . Then draw a sample from this kernel in an IID manner to form a subset kernel  $\mathbf{K}_{\mathcal{X}\hat{\mathcal{X}}} \in \mathbb{R}^{M \times \hat{M}}$  with  $\hat{M} \ll M$ , which forms the subset  $\hat{\mathcal{X}}$ . In the next step the eigensystem can be calculated through (4.5) and then the eigensystem of the target kernel can be extrapolated with:[55]

$$\mathbf{U}_{\mathcal{X}} = \mathbf{K}_{\mathcal{X}\hat{\mathcal{X}}} \mathbf{U}_{\hat{\mathcal{X}}} \mathbf{\Lambda}_{\hat{\mathcal{X}}}^{-1} \quad (6.3)$$

Furthermore, we can approximate the source kernel  $\mathbf{K}_{\mathcal{Z}}$  with the cross-domain kernel  $\mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}} \in \mathbb{R}^{N \times \hat{M}}$  and the subset kernel  $\mathbf{K}_{\hat{\mathcal{Z}}}$ . The procedure is similar to (6.3) or with the use of the kernels only, shown in (4.12) or (4.11), by replacing the target kernel with  $\mathbf{K}_{\hat{\mathcal{Z}}}$ .

$$\mathbf{K}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}} \mathbf{K}_{\hat{\mathcal{Z}}}^{-1} \mathbf{K}_{\hat{\mathcal{Z}}\mathcal{Z}} \quad (6.4)$$



For being able to solve the quadratic programming problem from (4.16), we need to approximate the extrapolated eigensystem. First, we extrapolate the eigensystem of the target kernel to the source subset via (4.13):

$$\bar{\mathbf{U}}_{\hat{\mathcal{Z}}} \simeq \mathbf{K}_{\hat{\mathcal{Z}}\mathcal{X}} \mathbf{U}_{\mathcal{X}} \mathbf{\Lambda}_{\hat{\mathcal{X}}}^{-1} \quad (6.5)$$

Second, the approximated eigensystem has to be extrapolated to the full source data:

$$\bar{\mathbf{U}}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}} \bar{\mathbf{U}}_{\hat{\mathcal{Z}}} \mathbf{\Lambda}_{\hat{\mathcal{X}}}^{-1} \quad (6.6)$$

With this, we have the required information to solve the Quadratic Program and are making the TKL algorithm and applicable for big data.[57]

Because it is a Nyström approximation, the quality of the approximation depends a lot on the quality of the taken sample, as mentioned in 4.1.1.

# Chapter 7

## Conclusion

In the course of this work, we have given a general understanding of transfer learning with many partial sub-problems and applications. Moreover, we gained knowledge on how probabilistic classification works and discussed advantages and disadvantages concerning the algorithms. One of many key features of a PCVM, is to deal with the uncertainty or latent state of the underlying structure.

Furthermore, we described how one could use transfer learning to boost the performance of the probabilistic classification vector machines. Additionally, we discussed standard methods to create a reliable study. This statistics vary in their methodology and addressing different constitutions of datasets. We have successfully created a new transfer learning algorithm and obtained many insights.

Especially the behavior, potential and limits of transfer learning methods. We gained knowledge of the efficiency of different Gaussian kernels concerning the parameters. It can be seen that the probabilistic classification vector machine can be greatly improved with the help of transfer learning and is finally prepared for difficult, especially transfer learning, tasks. The result is the Probabilistic Classification Transfer Kernel Vector Machine which is comparable to many currently available transfer learning solutions.

Summarizing, in general transfer learning aims to increase the performance of traditional supervised classification significantly. It can be shown that by integrating it, the performance can be greatly improved. This can be verified with several cross-domain datasets and statistical tests.

The state of current research shows a high amount of transfer learning methods.[88, p. 33] All of them are solving the problem in their way with an individual set of requirements, which results naturally in various performances, concerning tasks or domains in general. Based on different subproblems in transfer learning, the current state is that domain adaptations will lower the differences in marginal distributions. Furthermore, to address the issue of different conditional probability distributions, the pseudo labeling technique could be applied. When it comes to different tasks,

an inductive approach with a small amount of labeled target data is sufficient to align the differences. If one is faced with the problem that the feature spaces of two domains are not equal, e. g., different metrics, the use of heterogeneous transfer learning solutions may be helpful. Moreover, heterogeneous transfer learning can be used to create a sentiment bridge between two domains to add information to a certain domain to improve the classification. This results in a wide range of approaches which can be considered if one wants to integrate transfer learning in his application to meet his unique requirements and situations.

Moreover, traditional supervised machine learning algorithms are greatly challenged in real-world applications or difficult datasets.[66] This challenge can be addressed with transfer learning to make machine learning in general more applicable for real-world problems. The range of from real-world problems is vast. Some examples are sentiment classification from very different kinds of text document, difficulty image classification or the detection of defect-prone software.[88] Also, but more importantly the task of human activity recognition, cancer prediction or other health-care related tasks.[14][50]

The incorporating of (transfer) machine learning in this research sections has the potential to greatly improve the health or life of patients. Although that transfer learning does a good job, there are still problems and potential left for the supervising learning task. This issues should be addressed to improve the performance of transfer learning not only under lab conditions but also in the real-world. This can help to reach a new level of supervised classification performance in the future.

# Appendix A

## Mathematical Notation

In the following appendix, the mathematical notation is explained. This notation is used throughout the whole thesis. In particular, the matrix notations are not consistently defined, which is observable for example in the following literature: [19], [8] and [57]. First there are some basic definitions of linear algebra:

A scalar or real number is represented with a normal upper letter, e.g.  $N$ . A vector is denoted as lower bold letter such as  $\mathbf{v}$  and is assumed as a column vector. The superscript  $T$  denotes the transpose of a matrix or vector. So a row vector is indicated as  $\mathbf{v}^T$ . Furthermore,  $(v_1 \dots v_N)$  is a row vector with  $N$  elements. The corresponding column vector is simply  $\mathbf{w}$  and  $(w_1 \dots w_N)^T$  is also a column vector. A bold upper letter such as  $\mathbf{M}$  represents a Matrix. The rows ( $N$ ) and columns ( $M$ ) of a matrix are denoted as upper letters i.e.,  $N \times M$ . In the context of pattern recognition, the rows and columns are the numbers of observations or data points and dimensions (features). The  $M \times M$  identity matrix is denoted as  $\mathbf{I}_{M \times M}$ . The elements of the identity matrix  $\mathbf{I}_{i,j}$  are:

$$I_{i,j} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

The expectation of a function  $f(x, y)$  with respect to a random variable  $X$  is denoted as  $E_x[f(x, y)]$ . If the distribution of  $X$  is conditional on another variable  $Y$ , then conditional expectation will be written as  $E_x[f(x) | Y]$ . In other words, the conditional expectations of  $X$  given  $Y$ .

The variance is denoted as  $var[f(x)]$ . The covariance of two observations  $\mathbf{x}$  and  $\mathbf{y}$  is written as  $cov[\mathbf{x}, \mathbf{y}]$ . The covariance of a vector with itself is denoted as  $cov(\mathbf{x}, \mathbf{x})$ . The shorthand is  $cov(\mathbf{x})$ .

This part of notation is inspired by Bishop, which is found in [8, p. xi - xii]

The marginal probability distribution of a random variable  $X$  is denoted as  $P(X)$ . The conditional probability variable  $Y$  given  $X$  is denoted as  $P(Y | X)$ . If  $g$  is a distribution of a random variable  $X$ , then  $p_i = g(x_i)$  is the probability that  $X = x_i$ . If a probability distribution  $f$  follows the normal distribution  $N$ , then this is denoted

General			
Number of training samples	$N$	Frobenius Norm	$\  \cdot \ _F$
Number of test samples	$M$	Identity matrix / Input matrix	$I$
Number of all samples	$K$	Dissimilarity matrix	$D$
Dimensions of original feature space	$D$	Kernel	$K$
Dimensions of subspace	$L$		
Transfer Learning Notation			
Source Domain	$\mathcal{Z}$	Target Domain	$\mathcal{X}$
Source Feature Space	$\mathcal{F}_{\mathcal{Z}}$	Source Feature Space	$\mathcal{F}_{\mathcal{X}}$
Source Domain Data	$\mathbf{Z}$	Target Domain Data	$\mathbf{X}$
Marginal Probability of Source	$P(\mathbf{Z})$	Marginal Probability of Target	$P(\mathbf{X})$
Source Task	$\mathcal{T}_{\mathcal{Z}}$	Target Task	$\mathcal{T}_{\mathcal{X}}$
Source Label Space	$\mathcal{Y}_{\mathcal{Z}}$	Target Label Space	$\mathcal{Y}_{\mathcal{X}}$
Source Labels	$\mathbf{y}_{\mathcal{Z}}$	Target Labels	$\mathbf{y}_{\mathcal{X}}$
Conditional Probability of Source	$P(\mathbf{y}_{\mathcal{Z}} \mathbf{z})$	Conditional Probability of Target	$P(\mathbf{y}_{\mathcal{X}} \mathbf{x})$
Predictive Function of Source	$f_{\mathbf{Z}}(\cdot)$	Predictive Function of Target	$f_{\mathbf{X}}(\cdot)$
Source Label/Data Composition	$\mathcal{D}_{\mathcal{Z}}$	Target Label/Data Composition	$\mathcal{D}_{\mathcal{X}}$
Transfer Learning Solution Notation			
Knowledge classes (MMKT)	$E$	Dimensions of image feature space (TiT)	$B$
Dimensions of text feature space (TiT)	$A$	Co-Occurrence (TiT)	$C$
Transformation Matrix (TCA)	$\mathbf{W}$	Transformation Matrix (JDA)	$\mathbf{A}$
Embedding Matrix (JDA)	$\mathbf{Z}$	Data-matrix of source and target	$\mathbf{T}$
Subspace (GFK)	$\mathbf{P}$		

Table A.1: Notation for Transfer Learning and Pattern Recognition. It is separated in three parts: First, the general notation. Secondly, transfer learning notation. The last shows the notation for a specific transfer learning method.

as  $f \sim N$ .

A dataset is represented as matrix shown above. The number of dimensions indicates the number of features and the number of elements indicate the number of observations. The composition of the features represents the feature space of the dataset. Matrix indices are denoted as  $(\mathbf{M})_{i,j}$  with  $i, j \geq 0$ . With this, the value of the  $i$ -th row and  $j$ -th column is addressed.

The explained above is the basis used in this thesis. However, for Transfer Learning and Pattern Recognition arises another need to capture their notations. This appendix shows only the aggregated notation in table A.1. The details can be found in chapter 3. In this table, transfer-solution specific definitions are marked.

### A.0.1 Terminology

In this thesis various definitions of the Gaussian distribution are appearing. This happens because we tried to be consistent with the cited sources. The Normal

distribution is equal to the Gaussian. The standard Normal or Gaussian distribution is defined as  $N(0, 1)$ , which is centered at the origin. Furthermore, the zero-mean Gaussian distribution has its mean also in origin with  $N(0, \sigma^2)$ , with  $\sigma^2$  as variance. If  $\sigma^2 = 1$ , then it is the standard Gaussian distribution.

The terms source and training or target and testing are synonyms.

A class refers to a portion of labels, which represents them. When it comes to multi-class problems, if it is evident, then we may also refer to it as a multi-label problem. Source and target classes are sometimes represented or called by source labels and target labels, respectively.

# Appendix B

## Complete Test Results

This appendix shows the complete result tables. They are not in chapter 5 for clarity. The detailed metrics for the single datasets can be looked up here. Furthermore, in this chapter there are test results which have not been showed yet. The reason for this is that the test in his construction has no cross-validation. However, this test is sometimes used to test transfer learning classification [56] and can be found in section B.2. The complete five two cross-validation test results are shown in section B.1.

### B.1 Five-Two Comparison

This section shows the detailed metrics of the cross-validation test. The tables are showing the mean over the 5x2 cv F test, which is a total of ten repetitions. Every repetition has it's unique sampling from the dataset. In table B.1, the complete test-results of the  $5 \times 2$ cv F test divided in wins, loses and ties is shown. The table B.2 shows not only the mean but the standard deviation in brackets. This result is also plotted in figure B.1 for images and plotted in figure B.2 for Reuters. The number of a dataset is the order of the sets, shown, for example, in B.2. Furthermore, in table B.3, the AUC and ACC values are shown. The used mean number of support vectors can be viewed in table B.4. The table B.6 shows the needed time in seconds. Finally, the mean of needed time for training and testing a cross-validation dataset is shown in table B.5.

Win-Lose-Tie	SVM	PCVM	PCTKVM $_{\theta_{Est}}$	TCA	JDA	GFK	TKL
Reuters	0-6-0	0-6-0	<i>4-1-1</i>	0-6-0	0-6-0	0-6-0	6-0-0
Significant	0-3	0-4	<i>0-0</i>	0-2	0-3	0-4	0-0
Image	6-4-2	0-12-0	<i>4-3-5</i>	9-3-0	6-3-3	6-6-0	10-1-1
Significant	1-0	0-1	<i>1-0</i>	2-1	3-0	2-0	4-0

Table B.1: The result of the 5 x 2 cv F test to compare classifiers on one dataset under the Error metrics. The competitor is PCTKVM and the result is shown as the number of wins, losses and ties of the listed classifier against the PCTKVM. The row significant shows the number of significant wins and loses.

Error Image	SVM	PCVM	PCTKVM $_{\theta_{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
C vs A	<b>52.38</b> (2.30)	61.77 (2.04)	57.12 (2.78)	57.35 (2.88)	53.11 (3.28)	53.32 (3.00)	63.68 (3.49)	53.70 (1.72)
C vs D	57.45 (5.03)	64.33 (5.19)	60.24 (6.23)	<b>56.69</b> (5.05)	63.44 (4.45)	58.72 (4.31)	65.21 (4.94)	57.71 (4.21)
C vs W	66.70 (4.25)	71.46 (4.21)	65.44 (5.6)	63.72 (6.57)	70.30 (5.11)	64.76 (6.34)	73.22 (2.27)	<b>63.52</b> (2.35)
A vs C	59.54 (2.61)	65.24 (2.14)	62.42 (2.74)	61.91 (2.27)	58.66 (2.12)	58.90 (1.33)	66.20 (1.44)	<b>58.70</b> (2.21)
A vs D	66.90 (4.86)	70.86 (5.71)	65.72 (3.12)	63.82 (4.09)	64.33 (4.34)	62.33 (6.85)	72.25 (4.08)	<b>60.90</b> (5.10)
A vs W	69.51 (1.91)	70.91 (2.59)	67.86 (4.34)	66.67 (5.05)	69.44 (4.25)	67.34 (2.42)	76.01 (2.07)	<b>64.29</b> (2.94)
D vs C	75.41 (1.13)	78.15 (2.05)	72.08 (2.39)	74.07 (3.09)	71.83 (1.78)	72.38 (2.04)	72.47 (2.38)	<b>70.94</b> (2.52)
D vs A	73.63 (1.61)	77.06 (2.52)	72.44 (2.07)	74.03 (2.47)	<b>68.93</b> (2.97)	66.99 (2.68)	71.25 (2.34)	70.39 (2.73)
D vs W	45.77 (2.97)	64.54 (5.35)	47.27 (5.57)	47.46 (4.74)	32.81 (3.87)	36.07 (2.84)	36.34 (4.83)	<b>32.47</b> (4.55)
W vs C	73.86 (1.52)	76.49 (1.76)	73.70 (2.1)	74.14 (2.79)	<b>68.44</b> (1.63)	74.07 (1.63)	74.64 (1.39)	68.51 (1.81)
W vs A	71.84 (1.24)	73.07 (1.69)	70.35 (1.83)	71.07 (2.92)	<b>66.28</b> (1.53)	72.13 (1.55)	71.27 (2.24)	68.08 (1.45)
W vs D	<b>25.10</b> (3.60)	43.69 (4.96)	47.01 (3.07)	40.76 (3.04)	27.13 (4.61)	25.61 (3.83)	28.29 (4.01)	29.81 (5.20)
RSME	61.59 (14.71)	68.23 (9.35)	63.64 (9.15)	62.76 (14.81)	59.67 (15.39)	59.50 (13.63)	64.33 (6.88)	<b>10.65</b>
Error Reuters	SVM	PCVM	PCTKVM $_{\theta_{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Orgs vs People	23.01 (1.58)	26.77 (3.18)	20.65(1.93)	20.45 (1.95)	22.78 (3.14)	24.88 (2.61)	26.95 (2.65)	<b>19.29</b> (1.73)
People vs Orgs	21.07 (1.72)	27.77 (2.19)	15.94(1.96)	14.05 (8.68)	19.68 (2.00)	23.23 (1.93)	28.23 (2.46)	<b>12.76</b> (1.16)
Orgs vs Places	30.62 (2.22)	33.42 (6.10)	30.82 (11.08)	23.78 (2.26)	28.38 (3.00)	28.30 (1.51)	33.46 (1.83)	<b>22.84</b> (1.62)
Places vs Orgs	35.45 (2.24)	35.49 (8.19)	36.14 (7.38)	27.62 (2.66)	32.42 (3.91)	35.37 (4.39)	35.73 (2.69)	<b>18.33</b> (3.75)
Places vs People	39.68 (2.35)	41.01 (6.98)	33.96 (9.29)	29.71 (5.03)	40.58 (4.11)	42.41 (2.59)	40.24 (4.37)	<b>29.55</b> (1.46)
People vs Places	41.08 (1.98)	40.69 (5.52)	33.96 (8.17)	41.08 (1.33)	41.39 (3.26)	43.51 (2.23)	42.56 (2.46)	<b>33.42</b> (3.28)
RSME	31.88 (8.43)	34.48 (5.93)	30.78 (10.35)	26.28 (9.32)	31.03 (9.00)	33.05 (8.81)	34.63 (6.28)	<b>22.81</b> (7.63)

Table B.2: Cross-validation comparison of the tested algorithms on 18 domain adaptation datasets by the error and RMSE metrics. It demonstrates the mean of the ten runs of cross-validation per dataset with the standard deviation in brackets. The winner is marked with a bold performance value.



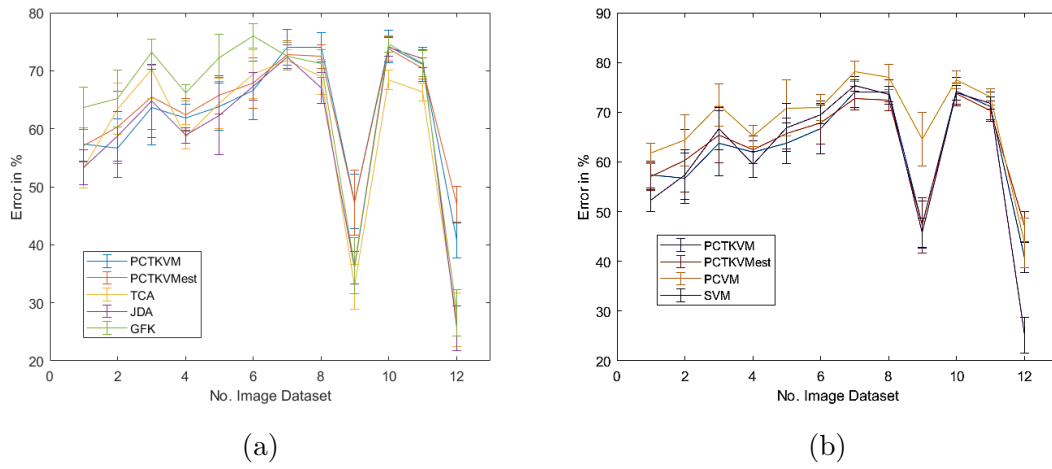


Figure B.1: The plot of mean errors with standard deviation of the cross-validation. The left shows the transfer PCVM in comparison with the other transfer learning solutions. The right side shows all PCVM's in comparison with SVM and TKL. A graph shows the error and a vertical bar shows the standard deviation.

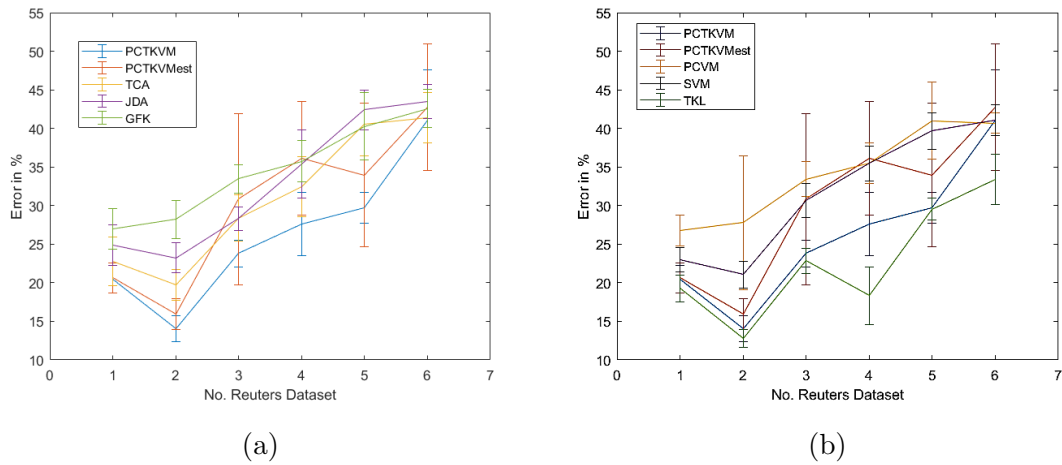


Figure B.2: The plot of mean errors with standard deviation of the cross-validation. The left shows the transfer PCVM in comparison with the other transfer learning solutions. The right side shows all PCVM's in comparison with SVM and TKL. A graph shows the error and a vertical bar shows the standard deviation.

<b>AUC Image</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
C vs A	79.62	81.10	78.34	79.18	65.65	79.98	74.55
C vs D	64.36	89.85	95.12	96.98	58.86	72.98	68.76
C vs W	78.21	73.85	89.95	85.59	71.47	84.85	83.41
A vs C	74.93	73.48	74.31	72.29	66.36	71.37	69.78
A vs D	70.03	76.08	75.77	88.04	58.23	65.28	81.72
A vs W	77.70	73.40	81.35	82.82	66.82	72.82	80.80
D vs C	68.13	49.74	57.9	56.38	71.43	74.28	67.84
D vs A	76.25	35.30	45.64	45.02	79.09	78.97	77.51
D vs W	94.87	48.71	74.78	71.91	89.79	96.73	95.01
W vs C	58.39	56.29	56.08	57.20	62.92	70.04	63.22
W vs A	73.54	43.79	61.4	62.70	73.25	76.16	70.78
W vs D	72.48	89.56	93.61	97.21	73.36	79.56	92.26
<b>AUC Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
Orgs vs People	81.14	80.29	88	88.62	83.45	80.34	90.43
People vs Orgs	82.23	81.08	91.48	92.64	85.98	82.39	93.61
Orgs vs Places	73.99	73.51	74.95	82.56	76.31	75.47	84.00
Places vs Orgs	69.50	69.97	77.35	80.21	72.86	68.59	89.50
Places vs People	59.36	63.11	71.77	77.15	60.95	58.72	76.57
People vs Places	47.77	57.78	55.25	54.97	54.35	49.15	66.63

Table B.3: Cross-validation comparison of the tested algorithms on 18 domain adaptation datasets by the AUC and ACC value. It shows the mean of ten runs of cross-validation per dataset. The positive class is one, and the negative class is the composition of remaining classes.

<b>N. SV. Image</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
C vs A	556.70	125.80	87.6	95.60	459.70	516.10	503.90
C vs D	557.60	124.60	78.3	87.80	466.80	522.20	522.30
C vs W	557.30	126.40	82.2	93.40	468.80	520.30	526.70
A vs C	458.00	93.70	75.9	79.20	356.30	416.70	412.30
A vs D	455.80	92.70	62.4	73.40	357.60	416.90	432.80
A vs W	457.70	93.50	66.9	76.70	357.00	416.10	429.30
D vs C	78.50	20.50	18.7	19.30	69.40	75.90	71.20
D vs A	78.50	21.20	18.6	18.90	69.30	76.00	72.60
D vs W	78.50	20.80	19.2	19.60	71.50	77.00	71.40
W vs C	146.00	33.60	29.9	31.70	119.70	137.00	127.90
W vs A	146.70	33.90	29.5	30.80	121.20	137.50	130.90
W vs D	146.90	34.10	31	32.10	122.30	137.50	126.70
<b>N. SV. Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
Orgs vs People	514.60	33.80	2.7	2.80	163.90	207.10	327.30
People vs Orgs	526.20	54.10	2.6	3.00	197.50	240.20	389.00
Orgs vs Places	427.20	25.30	5.3	3.10	155.50	187.20	355.00
Places vs Orgs	464.00	30.10	2.2	2.80	189.10	212.90	428.80
Places vs People	477.70	58.30	2.9	3.40	219.20	261.20	350.50
People vs Places	484.40	80.00	2.4	3.00	171.00	213.10	444.60

Table B.4: A complete comparison of used support vectors in the cross-validation test over the 18 domain adaptation datasets. It shows the mean of the ten runs of cross-validation.

### B.1.1 Time Comparison

In this section, we will discuss the time comparisons. Note that one should do a critical review of this results because of it's subjectivity and therefore they are not included in the main part. However, a point worth mentioning is that we have greatly improved the speed of the PCVM through the PCTKVM.

The baseline takes part in the time measure for the transfer learning solution because it is about to test, for example, if the SVM can handle the subspaces or a different kernel in proper time. The table B.5 shows the mean for the dataset categories Reuters and images.

The computer which is used to determine the needed time was an Acer Aspire V5-573G, containing an Intel Core i5 4200U Processor with 2x 1.60 GHz, 8GB of main memory and a NVIDIA GeForce GT 750M with 4GB of memory as graphic card.

Time	SVM	PCVM	PCTKVM <sub><math>\theta_{Est}</math></sub>	PCTKVM	TCA	JDA	GFK	TKL
Reuters	<b>11.87</b>	543.71	<i>12.78</i>	<i>16.20</i>	16.23	18.80	36.67	12.46
Image	0.05	258.78	<i>19.95</i>	<i>17.69</i>	0.71	0.66	0.45	<b>0.38</b>

Table B.5: The comparison of the calculation time in seconds. The first row shows the mean time for Reuters and the second row shows the average time over image datasets. The means are calculated with 10 runs per dataset.

<b>Time Image</b>	SVM	PCVM	PCTKVM $_{\theta_{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
C vs A	0.11	468.91	33.19	35.10	1.95	1.37	0.71	0.78
C vs D	0.06	478.03	3.66	3.44	0.60	0.65	0.54	0.12
C vs W	0.07	490.87	6.03	6.87	0.74	0.74	0.56	0.16
A vs C	0.11	385.44	67.81	42.08	1.97	1.35	0.68	0.91
A vs D	0.04	385.38	4.59	4.37	0.41	0.51	0.46	0.07
A vs W	0.05	385.79	8.64	7.59	0.55	0.60	0.47	0.12
D vs C	0.04	68.15	25.94	26.44	0.55	0.61	0.40	0.71
D vs A	0.03	67.68	23.82	23.01	0.40	0.50	0.35	0.56
D vs W	0.01	69.87	5.14	5.54	0.06	0.13	0.21	0.07
W vs C	0.05	100.65	27.54	27.94	0.68	0.69	0.43	0.62
W vs A	0.04	102.59	29.39	26.62	0.51	0.58	0.38	0.43
W vs D	0.01	101.94	3.66	3.27	0.05	0.13	0.22	0.03
<b>Time Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Orgs vs People	15.59	694.52	15.32	20.35	20.95	24.13	44.65	15.82
People vs Orgs	14.66	656.70	15.61	20.29	21.08	24.72	43.79	15.69
Orgs vs Places	9.93	451.34	11.03	13.37	14.10	16.00	34.36	10.71
Places vs Orgs	10.20	474.43	11.3	14.87	13.72	15.72	35.77	10.88
Places vs People	11.55	525.75	11.92	15.46	15.23	17.39	39.16	11.98
People vs Places	9.27	459.54	11.48	12.83	12.31	14.82	22.30	9.67

Table B.6: Comparison of the absolute running time in seconds in the cross-validation test over the 18 datasets. It shows the mean time of the ten runs of cross-validation

## B.2 Complete-Dataset Comparison

This section shows the results of the classifiers trained on the whole dataset. In general, the tables are showing the mean over five runs and are separated in the Reuters and image dataset. In table B.7, error and RMSE values shown. This results are also plotted in figure B.1 for images and plotted in figure B.2 for Reuters. The numbers of a dataset is the order of the sets, shown for example in B.2. The AUC values are shown in table B.8. For a complete overview over the used support vectors, see table B.9. Finally, the computation time in second as the mean over five runs is shown in B.10.

<b>Error Image</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	GFK	TKL
C vs A	46.66	55.80(1.06)	50.71 (2.18)	46.26 (1.75)	44.89	42.17	59.92	44.05
C vs D	51.59	64.97 (2.25)	50.83 (4.51)	50.83 (4.06)	54.14	56.69	58.60	54.78
C vs W	59.32	66.31 (2.73)	55.05 (3.95)	51.93 (2.48)	54.58	51.19	64.41	49.15
A vs C	56.10	62.85 (1.62)	56.81 (0.21)	55.10 (0.69)	54.23	54.59	59.75	54.05
A vs D	54.78	68.15 (4.18)	60 (2.6)	52.61 (3.17)	62.42	65.61	61.78	52.87
A vs W	61.69	71.12 (2.01)	64.2 (3.22)	58.51 (1.55)	60.68	55.93	62.03	57.29
D vs C	70.44	73.00 (1.3)	67.14 (0.41)	66.45 (1.11)	63.05	65.63	71.42	61.00
D vs A	69.83	72.11 (1.63)	66.12 (0.75)	63.82 (1.30)	61.59	62.63	66.39	60.96
D vs W	37.63	60.20 (4.96)	29.83 (3.12)	29.08 (3.99)	14.58	15.59	17.29	13.90
W vs C	74.35	72.45 (0.84)	67.16 (0.7)	67.03 (1.09)	67.94	68.12	71.15	65.00
W vs A	67.85	71.84 (1.4)	64.86 (1.87)	64.61 (1.84)	64.41	66.81	66.91	62.00
W vs D	15.29	37.71 (3.03)	33.12 (1.49)	26.75 (1.01)	15.92	11.46	11.46	15.29
RSME	55.46	64.71 (10.1)	55.80(12.28)	52.75 (13.4)	51.53	51.37	55.93	49.19
<b>Error Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	GFK	TKL
Orgs vs People	23.26	30.17(6)	19.97 (1.42)	19.72 (0.86)	22.93	24.83	25.33	18.05
People vs Orgs	21.75	25.16(2.83)	19.97 (1.42)	13.76 (0.38)	20.45	24.17	27.49	12.29
Orgs vs Places	30.01	36.82 (5.24)	33.27 (15.92)	24.20 (0.62)	28.19	27.04	32.69	23.20
Places vs Orgs	37.01	37.48 (2.34)	33.27 (15.92)	33.94 (9.65)	30.31	34.65	37.70	16.73
Places vs People	39.28	41.30 (1.97)	43.03 (14.37)	29.38 (0.33)	41.04	43.73	42.62	29.71
People vs Places	41.60	41.54 (0.90)	43.03 (14.37)	39.74 (0)	47.08	44.57	43.64	33.05
RSME	32.15	35.41(6.5)	34.09 (8.0)	26.79 (9.67)	31.67	33.16	34.91	22.17

Table B.7: The test result of the classifiers by using the whole 18 datasets. It shows the mean error over five runs.

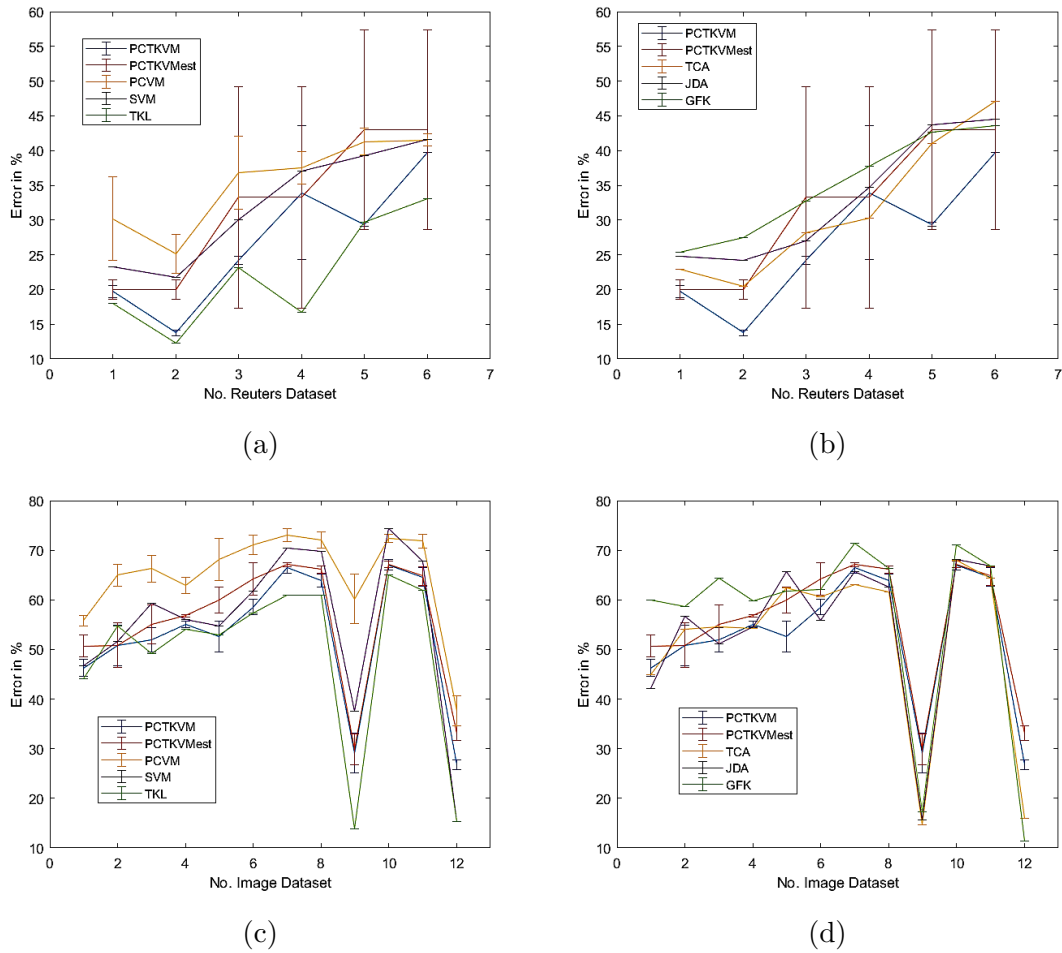


Figure B.3: The plot of mean errors with a standard deviation on the whole datasets. The left side shows all PCVM's in comparison with SVM and TKL. The right side shows all PCVM's in comparison to the other transfer learning solutions. The first row shows the result on Reuters and the second row illustrates the result on image dataset. A graph shows the error and a vertical bar shows the standard deviation.

<b>AUC Image</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
C vs A	71.75	85.73	84.53	85.81	64.21	71.67	65.76
C vs D	56.78	94.85	93.37	90.78	60.40	68.45	51.49
C vs W	67.15	85.61	90.15	95.74	50.75	64.92	78.47
A vs C	74.87	73.38	78.09	76.09	58.27	73.57	74.39
A vs D	72.93	78.89	97.26	88.37	57.53	71.84	78.68
A vs W	86.48	68.09	86.57	80.38	77.50	75.36	82.81
D vs C	76.33	52.52	61.55	50.66	82.63	80.23	78.78
D vs A	79.78	54.52	66.8	52.47	85.83	86.81	82.57
D vs W	96.36	52.31	73.54	70.18	76.33	98.03	97.24
W vs C	66.91	54.76	69.74	66.82	70.11	77.41	71.79
W vs A	72.46	49.93	62.46	64.64	74.36	84.21	77.75
W vs D	80.69	84.91	93.05	98.70	84.02	90.29	93.28
<b>AUC Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
Orgs vs People	79.97	79.00	89	89.54	83.80	81.13	90.70
People vs Orgs	80.35	81.60	89	92.57	84.27	81.95	93.69
Orgs vs Places	72.83	70.96	74.47	83.55	77.08	76.46	84.79
Places vs Orgs	66.40	69.89	74.47	87.33	74.64	69.55	91.34
Places vs People	56.36	60.64	66.2	77.03	56.78	56.43	76.31
People vs Places	42.59	51.82	66.2	58.40	39.85	45.87	66.95

Table B.8: The test result of the classifiers by using the whole 18 datasets. It shows the mean AUC value over five runs. The positive class is one, and the negative class is the composition of remaining classes.



<b>N. SV. Image</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
C vs A	1098.00	242.80	109.8	153.00	818.00	973.00	937.00
C vs D	1098.00	257.20	105.8	130.60	828.00	976.00	983.00
C vs W	1098.00	246.20	116.2	137.00	834.00	980.00	976.50
A vs C	898.00	187.40	113	128.80	577.00	726.00	749.00
A vs D	898.00	184.00	93	114.60	581.00	729.00	829.00
A vs W	898.00	188.40	101.6	112.60	588.00	742.00	791.00
D vs C	157.00	39.80	35.6	37.60	130.00	149.00	137.00
D vs A	157.00	37.20	32.8	35.00	134.00	149.00	139.00
D vs W	157.00	41.40	33	36.00	138.00	149.00	130.00
W vs C	295.00	65.20	55	53.80	214.00	273.00	248.00
W vs A	295.00	64.00	51.4	53.80	210.00	270.00	248.00
W vs D	295.00	63.40	53.4	55.20	224.00	275.00	239.00
<b>N. SV. Reuters</b>	SVM	PCVM	PCTKVM $_{\theta_{\text{Est}}}$	PCTKVM	TCA	JDA	TKL
Orgs vs People	850.00	66.60	3	2.40	280.00	329.00	604.00
People vs Orgs	867.50	53.60	3	2.80	309.00	356.50	667.00
Orgs vs Places	708.00	60.80	2	2.80	265.00	274.00	679.00
Places vs Orgs	752.00	43.00	2	3.00	291.50	297.00	746.50
Places vs People	845.00	48.00	4	4.00	362.00	447.00	659.00
People vs Places	838.50	54.00	4	2.20	315.50	390.50	748.50

Table B.9: Comparison of the number of used support vectors for the whole 18 datasets. It shows the mean number over five runs.

<b>Time Image</b>	SVM	PCVM	PCTKVM $_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
C vs A	0.41	1561.72	124.38	95.19	12.72	5.66	1.58	1.70
C vs D	0.19	1534.87	13.51	10.20	3.85	2.21	1.04	0.28
C vs W	0.22	1523.58	17.97	39.27	4.58	2.61	1.10	0.71
A vs C	0.36	1167.97	133.97	77.53	12.07	5.51	1.53	2.01
A vs D	0.14	1188.29	11.61	9.76	2.55	1.76	0.97	0.22
A vs W	0.16	1175.60	17.16	63.73	3.27	2.09	1.00	0.61
D vs C	0.12	104.43	80.73	70.93	3.31	2.09	0.81	1.43
D vs A	0.10	106.09	72.52	50.58	2.27	1.68	0.72	1.04
D vs W	0.02	106.48	14.7	14.68	0.24	0.36	0.29	0.43
W vs C	0.15	179.01	81.85	92.44	4.30	2.62	0.93	1.65
W vs A	0.12	177.99	66.65	62.21	3.12	2.03	0.78	1.12
W vs D	0.02	177.49	10.66	8.14	0.26	0.37	0.32	0.11
<b>Time Reuters</b>	SVM	PCVM	PCTKVM $_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Orgs vs People	49.48	2256.50	64.23	61.25	73.74	103.73	33.24	49.19
People vs Orgs	47.16	2024.11	64.23	67.62	69.82	100.04	32.68	47.86
Orgs vs Places	34.79	1529.51	45.37	47.62	51.44	69.37	27.66	34.53
Places vs Orgs	33.18	1488.56	45.37	52.41	46.38	66.13	27.28	36.25
Places vs People	38.01	1794.56	51.49	56.30	63.67	88.54	30.87	43.54
People vs Places	37.11	1641.59	51.49	60.23	53.68	77.32	28.37	38.70

Table B.10: Comparison of the absolute running time in second for the whole 18 datasets as mean. It shows the mean time of five runs.

# Appendix C

## Details of Algorithms

In this appendix, the detailed steps for the PCTKVM algorithm are presented. In the main part of this thesis, we have omitted some steps for clarity. The complete algorithm of the PCTKVM is shown in algorithm 4.

We also have extended the algorithm with the faster approach, where the dissimilar matrix is reused over tasks. Furthermore, we have extended the pseudo code with the option, which allows the user to decide, whether he wants to specify a  $\theta$  or proceed with the estimation. If one decides to choose a  $\theta$  on his own, then the model has two parameters: The width of the Gaussian kernel and the  $\zeta$  eigenspectrum dumping factor. The PCTKVM supports the standard Gaussian kernel, new Gaussian kernel from (5.3) and the Laplacian kernel. However, we strongly recommend the use of the new Gaussian kernel.

Furthermore, the number of iterations *niter* for the PCVM and the convergence criteria as a *threshold* can be specified. In algorithm 3 the pseudo code for the PCVM and a line explanation can be found.

**Algorithm 3** Probabilistic Classification Vector Machine

---

**Input:**  $\mathbf{I} = \{\mathbf{X}, \mathbf{Y}\} = \{(x_n, y_n)\}_{n=1}^N$  as  $N_Z$  sized training set;  $ker$  as kernel type;

The kernel parameter  $\theta$ ;  $niter$  as maximal number of iterations; *threshold*  $\tau$  as convergence criteria; **InitVector** as  $N_Z$ -sized initialization vector.

**Output:** Weight Vector  $\mathbf{w}$ , bias  $b$  and the kernel parameter  $\theta$ .

```

1:  $[\mathbf{w}, b] = \text{initialize}(\text{initVector})$ ;
2:  $\text{nonZero} = \text{determine\_nonZero\_Vector}(\mathbf{w})$ ;
3: for  $i = 1$  to  $niter$  do
4:    $\Phi = \text{Calculate\_Kernel}(\mathbf{X}, \mathbf{Y}, ker, \theta)$ ;
5:    $\mathbf{w}^{new} = \text{weight\_update}(\Phi, \mathbf{w}, \mathbf{Y}, \text{nonZero})$ ;
6:    $b^{new} = \text{bias\_update}(\Phi, b, \mathbf{Y}, \text{nonZero})$ ;
7:    $\theta^{new} = \text{parameter\_update}(\Phi, \mathbf{X}, \mathbf{Y}, ker, \theta, \mathbf{w}^{new}, b^{new}, \text{nonZero})$ ;
8:    $\text{nonZero}^{new} = \text{determine\_nonZero\_Vector}(\mathbf{w}^{new})$ ;
9:   if  $\max(\text{abs}(\mathbf{w}^{new} - \mathbf{w})) < \tau$  then
10:     break;
11:   else
12:     continue;
13:   end if
14: end for

```

---

Explanation of the Steps within the Algorithm.

1. Line 1 and 2: (Random) initialization of the weight vector  $\mathbf{w}$  with  $N$ -sized **initVector** and bias  $b$ . Determining non-zero elements in  $\mathbf{w}$  and save it as an index to **nonZero**.
2. Line 4: Calculate the kernel  $\Phi$ .
3. Line 5 and 6: Update  $\mathbf{w}$  and  $b$  with (2.18) and (2.19) respectively.
4. Line 7: Solve optimization problem based on (2.20).
5. Line 9-15: Determine the convergence by comparing the difference of  $\mathbf{w}^{new}$  and  $\mathbf{w}$  against the threshold. Break if convergence happens, otherwise continue.

---

**Algorithm 4** Probabilistic Classification Transfer Kernel Vector Machine

**Input:** Input Data  $\mathbf{K} = [\mathbf{Z}; \mathbf{X}]$  as  $N_Z$  sized training and  $N_X$  sized text set;  $\mathbf{Y}$  as  $N$  sized training label vector; kernel(-type)  $ker$ ; eigenspectrum dumping factor  $\zeta$ ;  $\theta$  as kernel parameter;  $niter$  as maximal number of iterations; *threshold*  $\tau$  as convergence criteria; **InitVector** as  $N_Z$ -sized initialization vector.

**Output:** Weight vector  $\mathbf{w}$ ; bias  $b$ , kernel parameter  $\theta$ ; transfer kernel  $\bar{\mathbf{K}}_A$ .

```

1:  $\mathbf{D} = \text{calculate\_Dissimilarity\_Matrix}(\mathbf{K});$ 
2: if  $\theta == -1$  then
3:    $\theta = \text{theta\_Estimation}(\mathbf{D});$  ▷ According to section (4.3.1)
4: end if
5:  $[\mathbf{K}_Z, \mathbf{K}_X, \mathbf{K}_{ZX}] = \text{compute\_Gaussian\_Kernel}(\mathbf{D}, ker, \theta),$ 
6:  $[\{\mathbf{\Lambda}_X, \mathbf{U}_X\}] = \text{eigenDecompose}(\mathbf{K}_X)$  ▷ Eq. (4.5)
7:  $[\bar{\mathbf{U}}_Z] = \text{extrapolate\_Eigensystem}(\mathbf{K}_{ZX}, \mathbf{U}_X, \mathbf{\Lambda}_X^{-1})$  ▷ Eq. (4.13)
8:  $[\mathbf{\Lambda}] = \text{quadratic\_Programming}(\mathbf{Q}, \mathbf{C}, r)$  ▷ Eq. (4.16) and constrains (4.17)
9:  $[\bar{\mathbf{K}}_A] = \text{compute\_Extrapolated\_Kernel}(\bar{\mathbf{U}}_Z, \mathbf{\Lambda}, \mathbf{U}_X)$  ▷ Eq. (4.18)
10:  $[\mathbf{w}, b] = \text{initialize}(\text{initVector});$ 
11: nonZero =  $\text{determine\_nonZero\_Vector}(\mathbf{w});$ 
12: for  $i = 1$  to  $niter$  do
13:    $\mathbf{w}^{new} = \text{weight\_Update}(\bar{\mathbf{K}}_Z, \mathbf{w}, \mathbf{Y}, \text{nonZero});$  ▷ Eq. (2.18)
14:    $b^{new} = \text{bias\_update}(\bar{\mathbf{K}}_Z, b, \mathbf{Y}, \text{nonZero});$  ▷ Eq. (2.19)
15:    $\theta^{new} = \text{parameter\_update}(\bar{\mathbf{K}}_Z, \mathbf{Y}, ker, \theta, \mathbf{w}^{new}, b^{new}, \text{nonZero});$  ▷ Eq. (2.20)
16:    $\text{nonZero}^{new} = \text{determine\_nonZero\_Vector}(\mathbf{w}^{new});$ 
17:   if  $\max(\text{abs}(\mathbf{w}^{new} - \mathbf{w})) < \tau$  then
18:     break;
19:   else
20:     continue;
21:   end if
22: end for

```

---

# Bibliography

- [1] ABE, Shigeo: *Support Vector Machines for Pattern Classification*. London : Springer London, 2010. – ISBN 978–1–84996–098–4
- [2] AGGARWAL, Charu C. (Hrsg.): *Data classification: Algorithms and applications*. Boca Raton, FL : CRC Press, 2015 (Chapman & Hall / CRC data mining and knowledge discovery series). – ISBN 9781466586741
- [3] ALBERT, James H. ; CHIB, Siddhartha: Bayesian Analysis of Binary and Polychotomous Response Data. In: *Journal of the American Statistical Association* 88 (1993), Nr. 422, S. 669–679. – ISSN 01621459
- [4] ALPAYDM, Ethem: Combined  $5 \times 2$  cv F Test for Comparing Supervised Classification Learning Algorithms. In: *Neural Computation* 11 (1999), Nr. 8, S. 1885–1892. – ISSN 0899–7667
- [5] ARNOLD, A. ; NALLAPATI, R. ; COHEN, W. W.: A Comparative Study of Methods for Transductive Transfer Learning. In: *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, 2007, S. 77–82
- [6] BERKES, Pietro ; ORBÁN, Gergo ; LENGYEL, Máté ; FISER, József: Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. In: *Science (New York, N.Y.)* 331 (2011), Nr. 6013, 83–87. <http://science.sciencemag.org/content/suppl/2011/01/04/331.6013.83.DC1>. – ISSN 1095–9203
- [7] BISHOP, Christopher M.: *Neural Networks for Pattern Recognition*. New York, NY, USA : Oxford University Press, Inc, 1995. – ISBN 0198538642
- [8] BISHOP, Christopher M.: *Pattern Recognition and Machine Learning // Pattern recognition and machine learning*. Corrected at 8. printing 2009. New York, NY : Springer, 2009 (Information science and statistics). – ISBN 0–387–31073–8
- [9] BISHOP, Christopher M. ; TIPPING, Michael E.: Variational Relevance Vector Machines. In: *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc, 2000 (UAI'00). – ISBN 1–55860–709–9, S. 46–53

- [10] BORSBOOM, Denny ; MELLENBERGH, Gideon J. ; VAN HEERDEN, Jaap: The theoretical status of latent variables. In: *Psychological Review* Bd. 110. 2003, S. 203–219
- [11] BORTZ, Jürgen ; SCHUSTER, Christof: *Statistik für Human- und Sozialwissenschaftler*. 7., vollständig überarbeitete und erweiterte Auflage. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010 (Springer-Lehrbuch). – ISBN 978-3-642-12769-4
- [12] BUCHHOLTZ, Christiane: *Vieweg Studium Grundkurs Biologie*. Bd. 53: *Grundlagen der Verhaltensphysiologie*. Wiesbaden : Vieweg+Teubner Verlag, 1982. – ISBN 9783322887856
- [13] BURGESS, Chris J. C. ; SCHÖLKOPF, Bernhard: Improving the Accuracy and Speed of Support Vector Machines. In: *Advances in Neural Information Processing Systems 9*, MIT Press, 1997, S. 375–381
- [14] BURLINA, Philippe ; PACHECO, Katia D. ; JOSHI, Neil ; FREUND, David E. ; BRESSLER, Neil M.: Comparing Humans and Deep Learning Performance for Grading AMD. In: *Comput. Biol. Med.* 82 (2017), Nr. C, S. 80–86. – ISSN 0010-4825
- [15] CARO-LOPERA, Francisco J. ; LEIVA, Víctor ; BALAKRISHNAN, N.: Connection between the Hadamard and matrix products with an application to matrix-variate Birnbaum–Saunders distributions. In: *Journal of Multivariate Analysis* 104 (2012), Nr. 1, 126–139. <http://www.sciencedirect.com/science/article/pii/S0047259X11001461>. – ISSN 0047-259X
- [16] CHAI, T. ; DRAXLER, R. R.: Root mean square error (RMSE) or mean absolute error (MAE)? - Arguments against avoiding RMSE in the literature. In: *Geoscientific Model Development* 7 (2014), Nr. 3, 1247–1250. <https://www.geosci-model-dev.net/7/1247/2014/>
- [17] CHATTOPADHYAY, Rita ; SUN, Qian ; FAN, Wei ; DAVIDSON, Ian ; PANCHANATHAN, Sethuraman ; YE, Jieping: Multisource Domain Adaptation and Its Application to Early Detection of Fatigue. In: *ACM Trans. Knowl. Discov. Data* 6 (2012), Nr. 4, S. 18:1–18:26. – ISSN 1556-4681
- [18] CHEN, H. ; TINO, P. ; YAO, X.: Efficient Probabilistic Classification Vector Machine With Incremental Basis Function Selection. In: *IEEE Transactions on Neural Networks and Learning Systems* 25 (2014), Nr. 2, S. 356–369. – ISSN 2162-237X

- [19] CHEN, Huanhuan ; TINO, Peter ; YAO, Xin: Probabilistic classification vector machines. In: *IEEE transactions on neural networks* 20 (2009), Nr. 6, S. 901–914. – ISSN 1045–9227
- [20] COMMENGES, Daniel: *Information Theory and Statistics: an overview*. <http://arxiv.org/pdf/1511.00860v1>
- [21] CZADO, Claudia ; SCHMIDT, Thorsten: *Mathematische Statistik*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. – ISBN 9783642172618
- [22] DAI, Wenyuan ; YANG, Qiang ; XUE, Gui-Rong ; YU, Yong: Boosting for transfer learning. In: GHAHRAMANI, Zoubin (Hrsg.): *the 24th international conference*, S. 193–200
- [23] DAVID D. LEWIS: *Reuters-21578 text categorization test collection Distribution 1.0: README file (v 1.3)*. <http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt>. Version: 2004
- [24] DEMPSTER, A. P. ; LAIRD, N. M. ; RUBIN, D. B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39 (1977), Nr. 1, S. 1–38. – ISSN 00359246
- [25] DIETTERICH, Thomas G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. In: *Neural Comput* 10 (1998), Nr. 7, S. 1895–1923. – ISSN 0899–7667
- [26] DUAN, L. ; XU, D. ; TSANG, I. W. H. ; LUO, J.: Visual Event Recognition in Videos by Learning from Web Data. In: *IEEE transactions on pattern analysis and machine intelligence* 34 (2012), Nr. 9, S. 1667–1680. – ISSN 1939–3539
- [27] EDELMAN, Shimon ; BÜLTHOFF, Heinrich H. ; SKLAR, Erik: *Task and object learning in visual recognition*. <http://www.dtic.mil/get-tr-doc/pdf?AD=ADA259961>
- [28] EVGENIOU, Theodoros ; PONTIL, Massimiliano ; ELISSEEFF, André: Leave One Out Error, Stability, and Generalization of Voting Combinations of Classifiers. In: *Machine Learning* 55 (2004), Nr. 1, S. 71–97. – ISSN 0885–6125
- [29] FAWCETT: ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. (2004). <http://www.hpl.hp.com/techreports/2003/HPL-2003-4.pdf>
- [30] FERNANDO, Basura ; HABRARD, Amaury ; SEBBAN, Marc ; TUYTELAARS, Tinne: *Subspace Alignment For Domain Adaptation*. <http://arxiv.org/pdf/1409.5241v2>



- [31] GÄRTNER, Bernd ; MATOUSEK, Jiří: *Approximation Algorithms and Semidefinite Programming*. 2012. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. – ISBN 978-3-642-22015-9
- [32] GENTLE, James E.: *Matrix Algebra: Theory, Computations, and Applications in Statistics*. New York, NY : Springer Science+Business Media LLC, 2007 (Springer Texts in Statistics). – ISBN 978-0-387-70872-0
- [33] GOLUB, G. ; KAHAN, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix. In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2 (1965), Nr. 2, S. 205–224
- [34] GONG, Boqing: *Kernel Methods for Unsupervised Domain Adaptation*. <http://www-scf.usc.edu/~boqinggo/>. Version: 2015
- [35] GONG, Boqing ; SHI, Yuan ; SHA, Fei ; GRAUMAN, K.: Geodesic flow kernel for unsupervised domain adaptation. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 2066–2073
- [36] GRAEPEL, Thore: Kernel Matrix Completion by Semidefinite Programming. In: DORRONSORO, José R. (Hrsg.): *Artificial Neural Networks — ICANN 2002: International Conference Madrid, Spain, August 28–30, 2002 Proceedings*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002. – ISBN 978-3-540-46084-8, S. 694–699
- [37] GREG GRIFFIN, ALEX HOLUB, PIETRO PERONA: *Caltech-256 Object Category Dataset*. <http://authors.library.caltech.edu/7694/>
- [38] GRETTON, Arthur ; BORGWARDT, Karsten M. ; RASCH, Malte J. ; SCHÖLKOPF, Bernhard ; SMOLA, Alexander: A Kernel Two-sample Test. In: *J. Mach. Learn. Res.* 13 (2012), 723–773. <http://dl.acm.org/citation.cfm?id=2188385.2188410>. – ISSN 1532-4435
- [39] HARTMANN, Peter: *Mathematik für Informatiker: Ein praxisbezogenes Lehrbuch*. 6., überarb. Aufl. Wiesbaden : Springer Vieweg, 2015. – ISBN 978-3-658-03415-3
- [40] HE, Li ; RAY, Nilanjan ; ZHANG, Hong: Error bound of Nyström-approximated NCut eigenvectors and its application to training size selection. In: *Neurocomputing* 239 (2017), 130–142. <http://www.sciencedirect.com/science/article/pii/S0925231217302813>. – ISSN 0925-2312
- [41] HINTON, Perry R.: *SPSS explained*. London : Routledge, 2004. – ISBN 0415274095

- [42] HUANG, Pipei ; WANG, Gang ; QIN, Shiyin: Boosting for Transfer Learning from Multiple Data Sources. In: *Pattern Recognition Letters* 33 (2012), Nr. 5, S. 568–579. – ISSN 01678655
- [43] IGUAL, Laura ; SANTI, Seguí: Descriptive Statistics. In: *Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications*. Cham : Springer International Publishing, 2017. – ISBN 978–3–319–50017–1, S. 29–50
- [44] IMAN, Ronald L. ; DAVENPORT, James M.: Approximations of the critical region of the friedman statistic. In: *Communications in Statistics - Theory and Methods* 9 (2007), Nr. 6, S. 571–595. – ISSN 0361–0926
- [45] JANEZ DEMSAR: Statistical Comparisons of Classifiers over Multiple Data Sets. In: FINLAYSON, Bruce A. (Hrsg.): *Introduction to Chemical Engineering Computing*. Hoboken, NJ, USA : John Wiley & Sons, Inc, 2006. – ISBN 9780471776680, S. 1–4
- [46] JING GAO, WEI FAN, JING JIANG, JIAWEI HAN: *Knowledge Transfer via Multiple Model Local Structure Mapping*. New York, NY : ACM, 2008. – ISBN 9781605581934
- [47] KAI ZHANG ; VINCENT ZHENG ; QIAOJUN WANG ; JAMES KWOK ; QIANG YANG ; IVAN MARSIC: Covariate Shift in Hilbert Space: A Solution via Sorrogate Kernels. In: SANJOY DASGUPTA (Hrsg.) ; DAVID MCALLESTER (Hrsg.): *Proceedings of the 30th International Conference on Machine Learning (ICML-13)* Bd. 28, JMLR Workshop and Conference Proceedings, 2013, 388–395
- [48] KITAYAMA, Satoshi ; YAMAZAKI, Koetsu: Simple estimate of the width in Gaussian kernel with adaptive scaling technique. In: *Applied Soft Computing* 11 (2011), Nr. 8, S. 4726–4737. – ISSN 15684946
- [49] KOBTI, Ziad ; WU, Dan: *Lecture Notes in Computer Science*. Bd. 4509: *Advances in Artificial Intelligence: 20th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2007, Montreal, Canada, May 28-30, 2007. Proceedings*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007. – ISBN 978–3–540–72665–4
- [50] KOUROU, Konstantina ; EXARCHOS, Themis P. ; EXARCHOS, Konstantinos P. ; KARAMOUZIS, Michalis V. ; FOTIADIS, Dimitrios I.: Machine learning applications in cancer prognosis and prediction. In: *Computational and Structural Biotechnology Journal* 13 (2015), 8–17. <http://www.sciencedirect.com/science/article/pii/S2001037014000464>. – ISSN 2001–0370

- [51] KUMAR, M. P. ; PACKER, Benjamin ; KOLLER, Daphne: Self-paced Learning for Latent Variable Models. In: *Proceedings of the 23rd International Conference on Neural Information Processing Systems*. USA : Curran Associates Inc, 2010 (NIPS'10), 1189–1197
- [52] KUMAR, Sanjiv ; MOHRI, Mehryar ; TALWALKAR, Ameet: Sampling Methods for the Nyström Method. In: *J. Mach. Learn. Res.* 13 (2012), S. 981–1006. – ISSN 1532–4435
- [53] LESKOVEC, Jure ; RAJARAMAN, Anand ; ULLMAN, Jeffrey D.: *Mining of Massive Datasets*. Cambridge : Cambridge University Press, 2014. – ISBN 9781139924801
- [54] LI, M. ; BI, W. ; KWOK, J. T. ; LU, B. L.: Large-Scale Nyström Kernel Matrix Approximation Using Randomized SVD. In: *IEEE Transactions on Neural Networks and Learning Systems* 26 (2015), Nr. 1, S. 152–164. – ISSN 2162–237X
- [55] LONG, Mingsheng ; WANG, Jianmin ; DING, Guiguang ; PAN, Sinno J. ; YU, Philip S.: Adaptation Regularization: A General Framework for Transfer Learning. In: *IEEE Transactions on Knowledge and Data Engineering* 26 (2014), Nr. 5, S. 1076–1089. – ISSN 1041–4347
- [56] LONG, Mingsheng ; WANG, Jianmin ; DING, Guiguang ; SUN, Jianguang ; YU, Philip S.: Transfer Feature Learning with Joint Distribution Adaptation. In: *2013 IEEE International Conference on Computer Vision (ICCV)*, S. 2200–2207
- [57] LONG, Mingsheng ; WANG, Jianmin ; SUN, Jianguang ; YU, Philip S.: Domain Invariant Transfer Kernel Learning. In: *IEEE Transactions on Knowledge and Data Engineering* 27 (2015), Nr. 6, S. 1519–1532. – ISSN 1041–4347
- [58] LOWE, David G.: Distinctive Image Features from Scale-Invariant Keypoints. In: *Int. J. Comput. Vision* 60 (2004), Nr. 2, S. 91–110. – ISSN 0920–5691
- [59] MA, Changxue ; KAMP, Y. ; WILLEMS, L. F.: A Frobenius norm approach to glottal closure detection from the speech signal. In: *IEEE Transactions on Speech and Audio Processing* 2 (1994), Nr. 2, S. 258–265. – ISSN 1063–6676
- [60] MIHAIL, Milena ; PAPADIMITRIOU, Christos H.: On the Eigenvalue Power Law. In: *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*. London, UK, UK : Springer-Verlag, 2002 (RANDOM '02). – ISBN 3–540–44147–6, 254–262

- [61] MOHAMAD, Ismail B. ; USMAN, Dauda: Standardization and its effects on K-means clustering algorithm. In: *Research Journal of Applied Sciences, Engineering and Technology* 6 (2013), Nr. 17, S. 3299–3303
- [62] NEMTSOV, Arik ; AVERBUCH, Amir ; SCHCLAR, Alon: Matrix compression using the Nyström method. In: *Intelligent Data Analysis* 20 (2016), Nr. 5, S. 997–1019. – ISSN 1088467X
- [63] PALUSZEK, Michael ; THOMAS, Stephanie: *MATLAB Machine Learning*. Berkeley, CA : Apress, 2017. – ISBN 978–1–4842–2249–2
- [64] PAN, Sinno J. ; KWOK, James T. ; YANG, Qiang: Transfer Learning via Dimensionality Reduction. In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI Press, 2008 (AAAI'08). – ISBN 978–1–57735–368–3, 677–682
- [65] PAN, Sinno J. ; TSANG, Ivor W. ; KWOK, James T. ; YANG, Qiang: Domain adaptation via transfer component analysis. In: *IEEE transactions on neural networks* 22 (2011), Nr. 2, S. 199–210. – ISSN 1045–9227
- [66] PAN, Sinno J. ; YANG, Qiang: A Survey on Transfer Learning. In: *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), Nr. 10, S. 1345–1359. – ISSN 1041–4347
- [67] PORTER, M. F.: An Algorithm for Suffix Stripping: Readings in Information Retrieval. Version: 1997. <http://dl.acm.org/citation.cfm?id=275537.275705>. In: SPARCK JONES, Karen (Hrsg.) ; WILLETT, Peter (Hrsg.): *Readings in information retrieval*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc, 1997. – ISBN 1–55860–454–5, 313–316
- [68] QI, Guo-Jun ; AGGARWAL, Charu ; HUANG, Thomas: Towards Semantic Knowledge Propagation from Text Corpus to Web Images. In: *Proceedings of the 20th International Conference on World Wide Web*. New York, NY, USA : ACM, 2011 (WWW '11). – ISBN 978–1–4503–0632–4, S. 297–306
- [69] QUATTONI, Ariadna ; COLLINS, Michael ; DARRELL, Trevor: Transfer learning for image classification with sparse prototype representations. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 1–8
- [70] QUIÑONERO-CANDELA, Joaquin (Hrsg.): *Dataset shift in machine learning*. Cambridge, Mass : MIT Press, 2009 (Neural information processing series). – ISBN 9780262170055

- [71] RENNIE, Jasson D. M. ; SREBRO, Nathan: Fast Maximum Margin Matrix Factorization for Collaborative Prediction. In: *Proceedings of the 22Nd International Conference on Machine Learning*. New York, NY, USA : ACM, 2005 (ICML '05). – ISBN 1–59593–180–5, S. 713–719
- [72] SAENKO, Kate ; KULIS, Brian ; FRITZ, Mario ; DARRELL, Trevor: Adapting Visual Category Models to New Domains. In: *Proceedings of the 11th European Conference on Computer Vision: Part IV*. Berlin, Heidelberg : Springer-Verlag, 2010 (ECCV'10). – ISBN 3–642–15560–X, 213–226
- [73] SCHLEIF, F. M. ; CHEN, H. ; TINO, P.: Incremental probabilistic classification vector machine with linear costs. In: *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, S. 1–8
- [74] SCHOLKOPF, Bernhard ; SMOLA, Alexander J.: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA : MIT Press, 2001. – ISBN 0262194759
- [75] SHEWCHUK, Jonathan R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Pittsburgh, PA, USA, 1994
- [76] SINGH, B. ; SINGH, H. K.: Web Data Mining research: A survey. In: *2010 IEEE International Conference on Computational Intelligence and Computing Research*, 2010, S. 1–10
- [77] TAYLOR, Matthew E. ; STONE, Peter: Transfer Learning for Reinforcement Learning Domains: A Survey. In: *J. Mach. Learn. Res.* 10 (2009), S. 1633–1685. – ISSN 1532–4435
- [78] TESCHL, Gerald ; TESCHL, Susanne: *Mathematik für Informatiker: Bd. 2: Analysis und Statistik*. 3., überarb. Aufl. Berlin and Heidelberg : Springer Vieweg, 2014. – ISBN 978–3–642–54273–2
- [79] THEODORIDIS, Sergios: *Machine learning: A Bayesian and optimization perspective*. First edition. London : Elsevier, 2015. – ISBN 978–0–12–801522–3
- [80] THEODORIDIS, Sergios ; KOUTROUMBAS, Konstantinos: *Pattern Recognition, Fourth Edition*. 4th. Academic Press, 2008. – ISBN 1597492728
- [81] TIPPING, Michael E.: Sparse Bayesian Learning and the Relevance Vector Machine. In: *J. Mach. Learn. Res.* 1 (2001), S. 211–244. – ISSN 1532–4435
- [82] TOMMASI, Tatiana ; CAPUTO, Barbara: The more you know, the less you learn: From knowledge transfer to one-shot learning of object categories. In:

- Proceedings of the British Machine Vision Conference*, BMVA Press, 2009. – ISBN 1–901725–39–1, S. 80.1–80.11
- [83] TOMMASI, Tatiana ; ORABONA, Francesco ; CAPUTO, Barbara: Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In: *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 3081–3088
- [84] TREVOR HASTIE ; ROBERT TIBSHIRANI ; JEROME FRIEDMAN: *The Elements of Statistical Learning*. New York, NY : Springer New York, 2009. – ISBN 978–0–387–84857–0
- [85] VAN BAY, Herbert ; TINNE, Tuytelaars ; LUC, Gool: SURF: Speeded Up Robust Features. In: LEONARDIS, Aleš (Hrsg.) ; HORST, Bischof (Hrsg.) ; AXEL, Pinz (Hrsg.): *Computer Vision – ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–33833–8, S. 404–417
- [86] VERT, JP. ; TSUDA, K. ; SCHÖLKOPF, B.: A Primer on Kernel Methods. In: *Kernel Methods in Computational Biology*. Cambridge, MA, USA : MIT Press, 2004, S. 35–70
- [87] WANG, Zheng ; SONG, Yangqiu ; ZHANG, Changshui: Transferred Dimensionality Reduction. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Part II*. Berlin, Heidelberg : Springer-Verlag, 2008 (ECML PKDD '08). – ISBN 978–3–540–87480–5, S. 550–565
- [88] WEISS, Karl ; KHOSHGOFTAAR, Taghi M. ; WANG, DingDing: A survey of transfer learning. In: *Journal of Big Data* 3 (2016), Nr. 1, S. 1817. – ISSN 2196–1115
- [89] WENYUAN DAI ; GUI-RONG XUE ; QIANG YANG ; YONG YU: *Co-clustering based Classification for Out-of-domain Documents: Proceedings of the thirteenth ACM SIGKDD international conference on knowledge discovery and data mining, August 12-15, 2007, San Jose, CA, USA*. New York, NY : ACM, 2007. – ISBN 978–1–59593–609–7
- [90] WILLIAMS, Christopher K. I. ; SEEGER, Matthias: Using the Nyström Method to Speed Up Kernel Machines. In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA : MIT Press, 2000 (NIPS'00), 661–667

- [91] WU, Xindong ; VIPIN, Kumar ; J., Ross Q. ; JOYDEEP, Ghosh ; QIANG, Yang ; HIROSHI, Motoda ; J., McLachlan G. ; ANGUS, Ng ; BING, Liu ; S., Yu P. ; ZHI-HUA, Zhou ; MICHAEL, Steinbach ; J., Hand D. ; DAN, Steinberg: Top 10 algorithms in data mining. In: *Knowledge and Information Systems* 14 (2008), Nr. 1, S. 1–37. – ISSN 0219–1377
- [92] YANG, Yiming ; PEDERSEN, Jan O.: A Comparative Study on Feature Selection in Text Categorization. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc, 1997 (ICML '97). – ISBN 1–55860–486–3, 412–420
- [93] YI WANG ; NEVIN LIANWEN ZHANG: Severity of Local Maxima for the EM Algorithm: Experiences with Hierarchical Latent Class Models 2006, Prague, Czech Republic. Electronic Proceedings. In: MILAN STUDENÝ (Hrsg.) ; JIR\`I VOMLEL (Hrsg.): *Third European Workshop on Probabilistic Graphical Models, 12-15 September 2006, Prague, Czech Republic. Electronic Proceedings*, 2006, 301–308
- [94] ZHANG, Kai ; TSANG, Ivor W. ; KWOK, James T.: Improved Nyström Low-rank Approximation and Error Analysis. In: *Proceedings of the 25th International Conference on Machine Learning*. New York, NY, USA : ACM, 2008 (ICML '08). – ISBN 978–1–60558–205–4, S. 1232–1239