Abstrakt

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 Trainings Daten, 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 Bereich nicht ausreichen oder von einem anderen Bereich kommen und daher verschieden sind. Dies ist üblich 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, den für die Wissenschaft üblichen, Datensätzen und außerdem auf einem realen Anwendungsfall durchgeführt. Die Unterschiede wurden mit Hilfe der Kreuzvalidierung und dem Friedman Test festgestellt. Es konnte bewiesen werden, dass die vorgenommenen Änderungen die Leistung des Algorithmus signifikant verbessern. Generell kann gesagt werden, dass unter den gängigen Labortests die Qualität recht gut ist, jedoch konnte gezeigt werden, dass der Unterschied bei realen Anwendung nicht die Signifikanz der Labortests aufweist. Daher müssen weitere Überlegungen zur (verbesserten) Anwendungen von Transfer-Algorithmen in realen Fällen vorgenommen werden.

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 creating their model based on pre-classified training data. This model can predict future events and can classify new information. However, a problem where 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 being 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. It can be shown that our improvements are resulting in a significant rise of performance, which makes it competitive to other transfer learning approaches. Summarizing, although the performance under lab conditions is pretty well, when it comes to real-world scenario, the benefits are lowered. Therefore, future research must be done to make it more applicable to real-world problems.

Contents

List of Figures

List of Tables

List of Abbreviations

PSD	Positive Semidefinite Kernel	47
QP	Quadratic Program	49
RAP	Relational Adaptive bootstrapping	36
RBF	Radial basis function	65
RKHS	Reproducing Kernel Hilbert Space	19
RMSE	Root Mean Square Error	68
ROC	Receiver Operating Characteristics	67
RVM	Relevance Vector Machine	4
SDP	Semi Definite Program	27
SIFT	Scale Invariant Feature Transform	61
SURF	Speeded Up Robust Features	61
SVD	Single Value Decomposition	29
SVD	Singular Value Decomposition	29
SVM	Support Vector Machine	3
TCA	Transfer Component Analysis	26
TFIDF	Term-Frequency Inverse-Document-Frequency	59
TKL	Transfer Kernel Learning	46
TTI	Text to Image	38
TTL	Transductive Transfer Learning	22

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 patients will get and heart attack or not. They are playing a major part in the engineering of autonomous cars or can predict the future changes in the stock market. Besides this extreme examples, machine learning can play a role in everyday life.
It is used to characterise a new costumer on a website to identify his needs and his wishes regarding a certain product of this site. Another use case is to create an individual advertisement for a customer based on his behaviour to optimise the shop 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 categorisation of new upcoming websites into trustworthy or shady, based on the already analysed pool of websites, which results finally in spam protection software.
As the previous implies, the classification of information is one task in machine learning. 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 pattern.
The question, which algorithms provide the best model can be a challenging question. In 2007, the ten most influential algorithms in the research communities were identified. For example k-Means, Support Vector Machines, A priori, Expectation-Maximisation, AdaBoost and Naive Bayes. However, carefully, these algorithms can not ad-hoc used for any type problem concerning the data. These algorithms have prerequisites. The category supervised learning because an algorithm 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 algorithm are doing a pretty decent job.
From the above examples, 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 using supervised learning for creating the regarding model. In the course of this work, we will dive deeper in the techniques and algorithms of these vectors machine and discuss improvements concerning the support vector machine.
However, most of the supervised classification algorithms suffer all from a certain problem. The problem that the training data and the test data is somehow different, which may is 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 other domain has attracted some research interest.
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, which can even be observed at insects. In fact, human kind (primates) masters this task before all other species.
This idea on which is one reason for many fundamental insights of our world are based on is currently applied to the 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 algorithms, it should end in an improved performance of the algorithms.
The topic transfer learning is already established in the research community with over 700 published academic articles until 2016. Therefore we will discuss the current state of transfer learning research and explain some ideas of it. Furthermore, we will see how transfer learning can be integrated into the Probabilistic Classification Vector Machine and how the performance is influenced by it. [?][?][?][?][?][?] [?] [?][?][?][?][?][?]

Chapter 2

Probabilistic Vector Machines

In this chapter, the Probabilistic Classification Vector Machine (PCVM) and previous work is described. It is developed by Huanhuan Chen et al. in the corresponding article [?]. However, first, we look at some disadvantages of the Support Vector Machine (SVM). The idea of the RVM are mainly motivated to improve and solve the SVM.[?, p. 1-2] Moreover, finally, the PCVM is developed to cure some conception assumptions, which are leading in the opinion of Chen et al. to unstable results. Regarding to Chen et al. sees the PCVM as improved version of the RVM.[?]

2.1 Disadvantages of Support Vector Machines

At first the SVM, which is described as baseline classifier in section 5.2.1, provides a optimal solution through the convex optimization problem [?, p. 325], but suffers from a few disadvantages.
The SVM is non probabilistic. The problem is that the hard binary decisions which are made of the SVM are not made to 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.[?, 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 [?, p. 239-240]. This uncertainty can be interpreted as Bayesian probability.[?, p. 21]
Second, 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.[?]

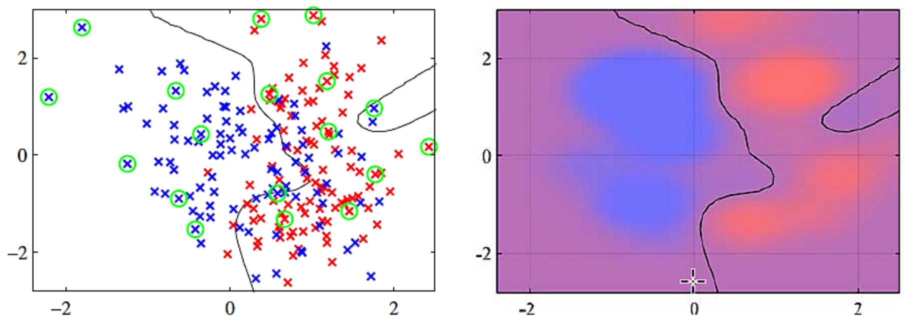


Figure 2.1: The probabilistic estimate of the RVM on a synthetic dataset. The green circled points are relevance vectors.[?, p. 356]

An example would be to find a set of 'reduced' vectors with $N_{\mathcal{Z}}$ entries, which are approximate the original set of support vectors $N_{\mathcal{S}}$. 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 $N_{\mathcal{Z}}$ with $N_{\mathcal{Z}} \ll N_{\mathcal{S}}$ for that the loss in the generalisation performance is acceptable. However, this approach is computational very expensive.[?] Therefore the wanted reduction of computational complexity is maybe not yet achieved.

Furthermore, the SVM has several parameters, that needs to be tuned by cross-validation. For example the C parameter, explained in section 5.2.1, or the parameters for the kernel function for example the width of the Gaussian kernel, section 5.2.3. Cross-validation is done by a grid search in a certain range. For example evaluate the performance of the SVM for $C = 1, 2, 5, 10, ..., 100, 200$ and select the parameter according to the best performance.[?]

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 data very well, because they are the closest points from one class to the other class.[?, p. 326]

Here a key feature from the Relevance Vector Machine (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.[?, p. 335-356] This effect can be seen in figure 2.1. One the right, the circled points are the relevance vectors of the model and on the right the posterior probability for the classes as colour gradient respectively.

2.2 Relevance Vector Machine

The RVM is introduced by Tipping in the already noted work [?]. 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 representing the importance of a relevance vector.[?, p. 213] Note that although in this thesis the same letter is used for the parameter $\mathbf{w}$, there is an important difference between them. The SVM defines with this parameter the hyperplane and the probabilistic vector machines are more prototypical vectors, which are representing the corresponding class in a probabilistic manner shown in figure 2.1.[?, p. 222]
Another advantage of the RVM against the SVM is , that the kernel function for a SVM has to satisfy Mercer's conditions. The RVM kernel does not have this constraint.[?, p. 213]
Without going deeper in this, a kernel which satisfies the Mercer's conditions is positive semi-definite.[?] The complexity of the algorithm is $\mathcal{O}(M^3)$ with M as number of basis functions and is also similar to the SVM.[?, p. 236-237]
The RVM makes predictions for a new point $\mathbf{x}$ with equation 2.1.[?, p. 211]

$$\mathbf{t} = y(\mathbf{x}; \mathbf{w}) = \sum_{i=1}^{N} \phi_i(\mathbf{x}) w_i + w_0 = \mathbf{\Phi}(\mathbf{x})\mathbf{w} + w_0 \tag{2.1}$$

With the bias w_0 and basis function $\mathbf{\Phi}(\mathbf{x}) = (\phi(\mathbf{x_1}), \ldots, \phi(\mathbf{x_n}))$. The weight parameter has the form $\mathbf{w} = (w_1, \ldots, w_N)^T$ and $\mathbf{T} = t_1, \ldots, t_N$ as function value.
In (2.1), 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 t is the regression value. [?, p. 662]
As a consequence, to being able to solve (2.1), the weight $\mathbf{w}$ with size N and w_0 has to be determined. For the classification, the RVM uses the Bayesian theorem in (2.2) do determine the weights $\mathbf{w}$.

$$p(\mathbf{w}|\mathbf{t}) \propto P(\mathbf{t}|\mathbf{w}) p(\mathbf{w}|\boldsymbol{\alpha}) \tag{2.2}$$

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, with $y = y(\mathbf{x})$, which gives the probabilistic output:

$$\sigma = 1/(1 + e^{-y}) \tag{2.3}$$

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

$$P(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} \sigma\{y(\mathbf{x}_n; \mathbf{w})\}^{t_n} [1 - \sigma\{y(\mathbf{x}_n; \mathbf{w})\}]^{1-t_n} \tag{2.4}$$

The Bernoulli distribution can be obtained from [?, p. 685] . Additionally, the prior $p(\mathbf{w}|\boldsymbol{\alpha})$ is obtained by a zero-mean Gaussian:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{N} \mathcal{N}(w_i|0, a_i^{-1}) \tag{2.5}$$

With the inverse hyperparameter vector $\boldsymbol{\alpha}$ of size $N+1$ as the precision of the Gaussian. At this point the parameter w_0 from (2.1) is determined. The parameter $\boldsymbol{\alpha}$ itself is Gamma distributed.[?, p. 214-215, 218-219]
An idea of a Gamma distribution and the Bernoulli distribution can be obtained from [?, 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 2.2. 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:

$$\begin{aligned} \boldsymbol{\Sigma} &= (\boldsymbol{\Phi}^T \mathbf{B} \boldsymbol{\Phi} + \mathbf{A})^{-1} \\ w_{MP} &= \boldsymbol{\Sigma} \Phi^T \mathbf{B} \mathbf{t} \end{aligned} \tag{2.6}$$

Where $\mathbf{A} = diag(\alpha_1, \ldots, \alpha_N)$ and $\mathbf{B} = diag(\beta_1, \ldots, \beta_N)$ with $\beta_n = \sigma\{y(\mathbf{x}_n)\}[1 - \sigma\{y(\mathbf{x}_n)\}]$. After the update of $\mathbf{w}_{MP}$ and $\boldsymbol{\Sigma}$ the hyperparameters are updated. This is repeated until $\boldsymbol{\alpha}$ satisfies a stable convergence criteria.[?, p. 219]
Although Tipping solved the problems of the SVM with the RVM, Chen et al. proposed that there are some concept problems left.[?] These are discussed in the following sections, especially section 2.3.2.

2.3 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.[?]
The PCVM uses a latent variable model because of the assumption that the data is incomplete or latent. Because of this and because the integral of the Bayesian Inference is in this case intractable, it uses an Expactation-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 optimises the parameters within, and therefore the PCVM does not need cross-validation because the free model parameter is optimised within the algorithm.[?]
An Attribute of the EM algorithm is that it is likely to be trapped in local maxima and therefore finds no global solution.[?] Unlike to the SVM which always can determine the global maxima, see section 5.2.1.

2.3.1 Model Specification

The PCVM is made for binary classification[?], unlike the RVM, which has a specification for multi class.[?, 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 (2.1) and (5.1).[?] As well as the RVM in section 2.2, the PCVM uses a link function to match the linear to a probabilistic output. In the PCVM it is done with the probit link function defined in 2.7:[?]

$$\Psi(\mathbf{x}) = \int_{-\infty}^{x} N(t|0,1)dt \tag{2.7}$$

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

With a CDF the probability for a new point 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.[?, p. 270]

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

$$l(\mathbf{x};\mathbf{w};b) = \Psi\left(\sum_{i=1}^{N} w_i \phi_{i,\theta}(\mathbf{x}+b\right) = \Psi(\mathbf{\Phi}_\theta(\mathbf{x})\mathbf{w}+b) \tag{2.8}$$

Where $\mathbf{\Phi(x)}_\theta = (\Phi_{1,\theta}(\mathbf{x}),\ldots,\Phi_{N,\theta}(\mathbf{x}))$ and $\mathbf{w} = (w_1,\ldots,w_N)^T$is the basis function vector with N training data points. θ is the parameter of the basis function. As the PCVM uses the Gaussian kernel as basis function therefore the parameter θ is referred as the width of the kernel.[?] The Gaussian kernel is described in the section 5.2.3.

2.3.2 Prior over Weights

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

It is important 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 $y_i \in \{-1,+1\}$.[?]

The SVM ensures this by defining the weight vectors as $\mathbf{w} = \{v_1y_1,\ldots,v_ny_n\}$, where v_i are non negative Lagrange multipliers. Therefore the same sign is guaranteed.[?]

But consider, that some of the data points which are not support vectors have $v_i = 0$.[?, p. 330]

The RVM estimates the weights according to equation (2.2). The likelihood is always positive because exponential function e is always positive [?, p. 355] and hence the logistic sigmoid link function is it. Therefore the sign depends on the prior, which

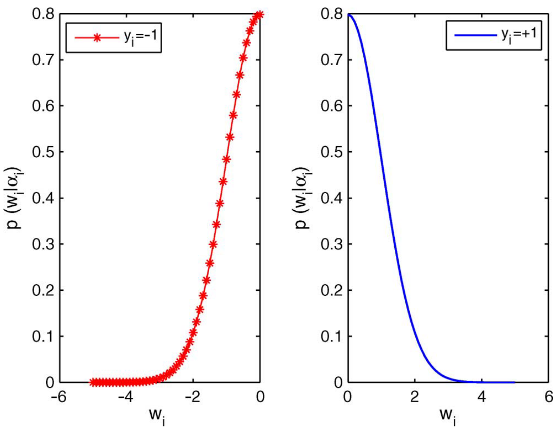


Figure 2.2: The truncated Gaussian priors over weights.On the left side the non positive left-truncated one in case of a negative class labels. On the right side the non negative right-runcated one for the positive class.[?]

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 the weight of a positive relevance vector can get a negative weight assigned and vice versa. This may lead to an unstable result corresponding to nonreliable vectors. As a consequence, the RVM is more likely to overfit the noise in comparison with the SVM and PCVM, with the same selection of parameters.[?]
At this point the improvements of the PCVM take place. Instead of the zero-mean Gaussian prior, it uses the zero-mean truncated Gaussian as prior. [?]

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \sum_{i=1}^{N} p(w_i|\alpha_i) = \sum_{i=1}^{N} N_t(w_i|0, \alpha_i^{-1}) \tag{2.9}$$

Here N_t is the truncated Gaussian function with α as inverse variance.
With that, the relevance vector is always getting the proper signed weight attached. Because for $y_i = +1$ the prior is selected from the right-truncated non-negative Gaussian and for $y_i = -1$ it is the left-truncated non-positive Gaussian.[?]
Now to avoid errors corresponding to noise or unreliable points the above is formulated in (2.10).[?]

$$p(w_i|\alpha_i) = \begin{cases} 2N(w_i|0, \alpha_i^{-1}), & if y_i w_i \geq 0 \\ 0, & if y_i w_i < 0 \end{cases} \tag{2.10}$$

With that, the PCVM assigns for an unreliable point a zero weight, and therefore there it is no longer considers as relevance vector for the model. The truncated Gaussian priors are illustrated in figure 2.2.
The bias b which is referred as w_0 in (2.5) is still determined with the zero-mean Gaussian:[?]

$$p(b|\beta) = N(b|0, \beta^{-1}) \tag{2.11}$$

2.3.3 Expectation Maximization Algorithm

The PCVM uses a simple latent variable model within the EM algorithm.[?] A latent variable can either be missing data or is unobservable.[?, p. 276-277] The latter is sometimes called hidden and can be for example class labels or parameters and has to be determined.[?, p. 84] Hidden in this context means this variable can not be directly observed and his existence is implicit.[?]
The difference between a parameter and a latent variable in this thesis is that the parameter should be learned through the learning algorithm based on the data. The latent variable is a part of the data which is not available (hidden) but is involved in the model. Therefore the algorithm takes assumptions to deal with the uncertainty of this latent variables.
The standard probabilistic assumption is that the formulation in (2.1) is affected. Therefore (2.1) can be reformulated as (2.12).[?]

$$h_\theta(\mathbf{x}) = \mathbf{\Phi}_\theta(\mathbf{x})\mathbf{w} + b + \epsilon \tag{2.12}$$

With the θ 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 ϵ is unobservable thus latent, the $h_\theta(\mathbf{x})$ itself will be considered as latent variable.[?]
The PCVM uses the probit link function, which is used for regression.[?]. It gives a variable $l = 1$, when the corresponding $h_\theta(\mathbf{x}) \geq 0$ and $l = 0$ for $h_\theta(\mathbf{x}) < 0$.[?]. Therefore, to obtain the probit mode, the Gaussian CDF is used.

$$P(l = 1|x, w, b) = p(\mathbf{\Phi}_\theta(\mathbf{x})\mathbf{w} + b + \epsilon \geq 0) = \Psi(\mathbf{\Phi}_\theta(\mathbf{x})\mathbf{w} + b) \tag{2.13}$$

It states, given the data $\mathbf{x}$ and the parameter $\mathbf{w}$ and b, how likely is it that it belongs to the positive class. To get the probability for the other class it is simply $P(l = 0|x, w, b) = 1 - P(l = 1|x, w, b)$.[?]
Revisiting (2.1), we need to determine $\mathbf{w}$ for the model. Again the Bayesian theorem is used to determine most probably parameter values.[?]
The prior for $\mathbf{w}$ is already obtain through 2.9 and the bias is specified with (2.11) 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:[?]

$$\begin{aligned} p(\mathbf{H}_\theta(\mathbf{x})|w, b) &= N(\mathbf{H}_\theta(\mathbf{x})|\mathbf{\Phi}_\theta(x)\mathbf{w} + b, 1) \\ &= (2\pi)^{N/2} \exp\{-\frac{1}{2}\,||\mathbf{H}_\theta - \mathbf{\Phi}_\theta\mathbf{w} + b\mathbf{I}||^2\} \end{aligned} \tag{2.14}$$

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

$\mathbf{\Phi}_\theta(\mathbf{x})$ refers to the row corresponding to $\mathbf{x}$.
For the complete log-posterior of the parameters $\mathbf{w}$ and b the corresponding parameters, α and β 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:[?]

$$\begin{aligned}\log p(\mathbf{w}, b|\mathbf{y}, \mathbf{H}_\theta, \alpha, \beta) &\propto \log p(\mathbf{H}_\theta|\mathbf{w}, b) + \log p(\mathbf{w}|\alpha) + \log p(b|\beta)\\ &\propto \mathbf{w}^T\mathbf{\Phi}_\theta^T(2\mathbf{H}_\theta - \mathbf{\Phi}\mathbf{w}) + 2b\mathbf{I}^T\mathbf{H}_\theta - 2b\mathbf{I}^T\mathbf{\Phi}_\theta - b^2N - \mathbf{w}^T\mathbf{A}\mathbf{w} - \beta b^2\end{aligned} \tag{2.15}$$

With that, we can proceed with the EM algorithm.[?]
The EM algorithm was originally introduced by Dempster et al. and is made for determining the maximum likelihood for incomplete data [?]. Based on their work, many other applications are made to determine parameters. For example Bishop used a EM algorithm within the algorithm k-means cluster.[?, p. 426-428] Furthermore, Hasti et al. using it in [?, 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 2.2 is similar.[?, p. 233-234]. Furthermore, he introduced an Expectation-Maximisation Update approach for determining the parameters.[?, . 235]
As the name says, it consists of two steps. The algorithm has the E and M steps per iteration, and will repeat as long as a convergence criterion is reached or the maximum iteration is reached, which can be specified in algorithm 2.3.4. The first, the Expectation-Step finds expectations for the latent variables and furthermore an expectation for posterior of the parameters, which is referred as $\mathcal{Q}$ function. In the case of the PCVM it has the form:[?]

$$\begin{aligned}\mathcal{Q}(\mathbf{w}, b|\mathbf{w}^{old}, b^{old}) &= E_{\mathbf{H}_\theta,\alpha,\beta}[\log p(\mathbf{w}, b|\mathbf{y}, \mathbf{H}_\theta, \alpha, \beta)|y, \mathbf{w}^{old}, b^{old}]\\ &= 2\mathbf{w}^T\mathbf{\Phi}_\theta^T\bar{\mathbf{H}}_\theta - \mathbf{w}^T\mathbf{\Phi}_\theta^T\mathbf{\Phi}_\theta\mathbf{w} + 2b\mathbf{I}^T\bar{\mathbf{H}}_\theta - b^2N + 2b\mathbf{I}^T\mathbf{\Phi}_\theta\mathbf{w} - \mathbf{w}^T\bar{\mathbf{A}}\mathbf{w} - \bar{\beta}b^2\end{aligned} \tag{2.16}$$

With $\bar{\mathbf{H}}_\theta = E[\mathbf{H}_\theta|y, \mathbf{w}^{old}, b^{old}]$, $\bar{\mathbf{A}} = diag(E[\alpha_i|y_i, \mathbf{w}^{old}, b^{old}])$ and $\bar{\beta} = E[\beta|y_i, \mathbf{w}^{old}, b^{old}]$.
For details to the determination the expectation, see Appendix B in [?].
This step relies on the 'old' latent variables and parameters from the previous iteration. The next part in the iteration is the Maximization-Step.[?]
Revisiting 2.16, $\mathcal{Q}$ gives the expectation of the posterior distribution based on the current state of all variables and parameters. Therefore by analysing 2.16, the algorithm gives the most probable expectations for the posterior for the parameters with respect to the latent variables and the old parameters.
In this algorithm, the maximum is found by analysing the derivatives of the $\mathcal{Q}$ function with respect to $\mathbf{w}$, b and θ. The latter is optimized because the Algorithm is designed to optimize his selectable parameter to avoid cross validation.[?]
A joint maximization of $\mathcal{Q}$ can not be done and therefore the parameters are optimized one after the other by set the derivatives to $\frac{\partial \mathcal{Q}}{\partial \mathbf{w}} = 0$ and $\frac{\partial \mathcal{Q}}{\partial b} = 0$ and solving

the equation for $\mathbf{w}$ and b respectively the update rules are obtained:[?]

$$\mathbf{w}_{new} = (\mathbf{\Phi}_\theta^T \mathbf{\Phi}_\theta + \bar{\mathbf{A}})^{-1} (\mathbf{\Phi}_\theta^T \bar{\mathbf{H}}_\theta - b \mathbf{\Phi}_\theta^T \mathbf{I}) \tag{2.17}$$

$$b^{new} = \frac{\mathbf{I}^T \bar{\mathbf{H}}_\theta - \mathbf{I}^T \mathbf{\Phi}_\theta \mathbf{w}}{\beta + N} \tag{2.18}$$

With that the maximum is found and therefore the 'new' parameters for $\mathbf{w}^{old}$ and b^{old} for 2.16 are found for the next iteration. The optimization for θ is done by a conjugate gradient algorithm[1] which is done on basis of (2.19).[?]

$$\frac{\partial Q}{\partial \theta_k} = 2 \sum_{i=1}^{N} \sum_{j=1}^{N} \left\{ (\mathbf{\Phi}_\theta \mathbf{w} - \bar{\mathbf{H}}_\theta) \mathbf{w}^T \odot (\frac{\partial \mathbf{\Phi}_\theta}{\partial \theta_k}) \right\}_{(i,j)} \tag{2.19}$$

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

2.3.4 Algorithm

In the following the pseudo code of PCVM is presented in algorithm 1.

Algorithm 1 Probabilistic Classification Vector Machine

```
Input: I ={X,Y} = $\{(x_n, y_n)\}_{n=1}^N$ as $N$-sized training set; ker as kernel type; The
    kernel parameter $\theta$; niter as maximal number of iterations; threshold as conver-
    gence criteria; InitVector as $N$-sized initialization vector.
Output: Weight Vector $\mathbf{w}$, bias $b$ and the kernel parameter $\theta$.
 1: $[\mathbf{w}, b]$ = initialize(initVector);
 2: nonZero = determine_nonZero_Vector($\mathbf{w}$);
 3: for $i = 1$ to niter do
 4:     $\Phi$ = Calculate_Kernel($\mathbf{X}, \mathbf{Y}$, ker, $\theta$);
 5:     $\mathbf{w}^{new}$ = weight_update($\mathbf{\Phi}, \mathbf{w}, \mathbf{Y}$, nonZero);
 6:     $b^{new}$ = bias_update = bias_update($\mathbf{\Phi}, b, \mathbf{Y}$, nonZero);
 7:     $\theta^{new}$ = parameter_update($\mathbf{\Phi}, \mathbf{X}, \mathbf{Y}$, ker, $\theta$, $\mathbf{w}^{new}$, $b^{new}$, nonZero);
 8:     nonZero$^{new}$ = determine_nonZero_Vector($\mathbf{w}^{new}$);
 9:     if max(abs($\mathbf{w}^{new} - \mathbf{w}$)) < threshold then
10:         break;
11:     else
12:         continue;
13:     end if
14: end for
```

Explanation of the Steps within the Algorithm.

[1]http://learning.eng.cam.ac.uk/carl/code/minimize/

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 index to **nonZero**.
2. Line 4: Calculate the kernel Φ.
3. Line 5 and 6: Update $\mathbf{w}$ and b with (2.17) and (2.18) respectively.
4. Line 7: Solve optimization problem based on (2.19).
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.

It can be seen in algorithm 1 that the convergence of the algorithm is only measured with the convergence behaviour of $\mathbf{w}$ and is set to $||\mathbf{w} - \mathbf{w}^{old}|| = 1.0e^{-3}$ in practice. The time complexity of the PCVM is $\mathcal{O}(N^3)$, with N as the number of training points. In practice the Cholesky decomposition is used for the matrix inversion. This has the computational complexity of $\mathcal{O}(M)^3$ and a memory complexity of $\mathcal{O}(M)^2$, where M is the number of non zero basis functions and is the overall complexity of the PCVM.[?]
However, because of the sparsity of the prior, the PCVM reduces the basis function from the initial start with $M = N$ to $M < N$. In the best case it be even $M \ll N$. An practical example of this reduction can be found in table B.3 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 function.

2.3.5 Hierachical Hyperpriors

In the section 2.3.2 the priors are introduced with the N sized parameter vector $\boldsymbol{\alpha}$ and β. Although they are already defined, they can alternatively be controlled with hierarchical hyperparameters regarding the Bayesian Framework. This means that hyperparameters are controlling the distribution of the prior. [?, p. 71] Because these hyperparameters are controlled by a prior of another distribution, this prior is called hyperprior.[?, .p 423]
The PCVM uses the gamma distribution for modelling the hyperprior:[?]

$$\begin{aligned} p(\boldsymbol{\alpha}) &= \prod_{i=1}^{N} Gamma(\alpha_i|c, d) \\ p(\beta) &= Gamma(\beta|e, f) \end{aligned} \tag{2.20}$$

With using the hyperparameters c, d, e and f the prior can be obtained by marginalizing out α_i in 2.21 and β in 2.22:[?]

$$\begin{aligned} p(w_i|c,d) &= \int_0^\infty p(w_i|\alpha_i)p(\alpha_i|,c,d)d\alpha_i \\ &= \begin{cases} \frac{2d^c\Gamma(c+\frac{1}{2})}{\sqrt{2\pi}\Gamma(c)}\left(\frac{w_i^2}{2}+d\right)^{-(c+\frac{1}{2})} & ,ify_iw_i \geq 0 \\ 0, & ify_iw_i < 0 \end{cases} \end{aligned} \tag{2.21}$$

$$\begin{aligned} p(b|e,f) &= \int_0^\infty p(b|\beta)p(\beta|e,f)d\beta \\ &= \frac{f^e\Gamma(e+\frac{1}{2})}{\sqrt{2\pi}\Gamma(e)}\left(\frac{b^2}{2}+f\right)^{-e+\frac{1}{2})} \end{aligned} \tag{2.22}$$

Which form the hierarchical hyperprior.[?] These are equivalent to the truncated student-t prior over **w** and the student-t prior over b.[?, p. 226]
This student-t prior is more sharply and peaked about zero than the Gaussian prior[?], which can be seen in [?, p. 688, 691-692].
In practice the hyperparameters c, d, e and f are set to zero and therefore the prior of **w** can be formalized as in 2.23.

$$p(w_i) == \begin{cases} \frac{2\Gamma(\frac{1}{2})}{\sqrt{2\pi}\Gamma(0)}\left(\frac{w_i^2}{2}\right)^{-\frac{1}{2}} & ,ify_iw_i \geq 0 \\ 0, & ify_iw_i < 0 \end{cases} \tag{2.23}$$

The hierarchical hyperprior for b can be obtained analogue to equation (2.23). This prior can be interpreted as Laplacian prior and leads to sparsity.[?] An idea of the multivariate Laplacian distribution can be obtained from [?].

2.3.6 Implementation Details

In the last section of this chapter, some changes in the formulations are introduced to make the algorithm more stable in practice. The Expectation of the latent variables for $\bar{\alpha}_i$ and $\bar{\beta}$ are calculated with:[?]

$$\bar{\alpha}_i = \frac{c+\frac{1}{2}}{w_i^2+d} \qquad \bar{\beta} = \frac{c+\frac{1}{2}}{b^2+f} \tag{2.24}$$

See Appendix B of [?] for details of the expectation derivation of $\bar{\alpha}_i$ and $\bar{\beta}$.
With that according to 2.3.5 the hyperparameters are set to zero. Because of the fact that α_i are inverse proportional to the square w_i, the parameter vector can be rewritten as $\bar{A} = diag[(\sqrt{2}w_1)^{-2}, \ldots, (\sqrt{2}w_N)^{-2}]$. As consequence the expectation changes to $\bar{\alpha}_i = (1/2w_i)$ and because of the sparse model many values of w_i tends towards zero and therefore it may become unstable.[?]

Revisiting the weight may become very small and leads to large numbers in $\bar{\mathbf{A}}$. Therefore the matrix $\mathbf{M} = diag[\sqrt{2}w_1, \ldots, \sqrt{2}w_N]$ with $(\mathbf{M}^{-1})^{-2}$ is used, which is suggested in [?]. This avoids the inversion of the elements w_i at the update state (M-Step). This modification is applied to b and $\mathbf{w}$ and therefore the equations from 2.17 and 2.18 changes to:[?]

$$\begin{aligned} \mathbf{w}^{new} &= \mathbf{M}(\mathbf{M}\mathbf{\Phi}_\theta^T\mathbf{\Phi}_\theta\mathbf{M} + \mathbf{I}_n)^{-1}\mathbf{M}(\mathbf{\Phi}_\theta^T\bar{\mathbf{H}}_\theta - b\mathbf{\Phi}_\theta^T\mathbf{I}_N) \\ b^{new} &= t(1 + tNt)^{-1}t(\mathbf{I}^T\bar{H}_\theta - \mathbf{I}^T\mathbf{\Phi}_\theta\mathbf{w}) \end{aligned} \tag{2.25}$$

With the $N \times N$ identity matrix $\mathbf{I}$ and $t = \sqrt{2}|b|$. Furthermore, the diagonal matrix elements of M are:

$$m_i = \bar{\alpha_i}^{-1/2} = \begin{cases} \sqrt{2}w_i, & if y_i w_i \geq 0 \\ 0, & if y_i w_i < 0 \end{cases} \tag{2.26}$$

This changes should allow a stable numerical calculation.
However, this changes may not avoid the fact that some diagonal elements of $\bar{\mathbf{A}}$ in $(\mathbf{\Phi}_\theta\mathbf{\Phi}+\bar{\mathbf{A}})$ becoming very large and therefore the matrix becomes numerical singular. This happens when w_i tends towards zero. To avoid this, the kernel column of the corresponding very small weight is deleted. [?]
As already mention for the matrix inversion the Cholesky decomposition is used.[?] The Cholesky decomposition is considered as numerical stable.[?, p. 101]
Note that because of the zero-mean Gaussian assumption, i e. $N(0, 1)$ the datasets for training and testing for the PCVM has to be z-scored. Refer to equation (4.21) in section 4.3.1 for this.

2.4 Conclusion

Summarizing, Chen et al. have proposed the PCVM as improvement to the SVM and RVM. They have evaluated this statement, in a performance study on three synthetic datasets and 12 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 outperform the RVM and SVM. In fact the PCVM has the same advantages over SVM as the RVM has and furthermore uses a more reasonable prior.[?]

Chapter 3

Transfer Learning

In this chapter, the technique of transfer learning will be introduced. It can be considered as a large interpretation of related work.
This Chapter is organized as follows: First of all, we discuss some challenges of the 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.
Furthermore, the settings of transfer learning will be introduced. The settings implying the different setups to solve the transfer problem.
In the next section will introduce and explain the two main types of the transfer learning which describes the conditions of the feature space and probability distributions. 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 5.2.3.

3.1 Challenges of Machine Learning Algorithms

The current state of 'traditional' machine 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. Reason for this may that training

data, which has to be labelled, is expensive or difficult to achieve.[?, p. 1]
When it comes to supervised learning, the algorithms are designed in a way to learn from the pre-labelled data and hence it is crucial for this types of algorithms to have some.[?, p. 6-7]
In the 'traditional' manner, which applies to the most statistical models, to solve the problem of differences, they have to collect new training data for the new feature space or distribution of the test data and rebuild the model.[?] 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, which is the main motivation of Transfer Learning.[?, p. 1.]
A simple interpretation of Transfer Learning in the real world is given in the following: Imaging two people, who want to learn piano. One has never played any musical instrument before, while the second has previous collected knowledge on how to play music through experience with other instruments. The latter will be able to learn the new instrument much faster than the first with no previous experience because he can transfer his general music knowledge to boost his learning of the new instrument.[?, p. 1]
When going back to machine learning, there are be 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 labelling. If we want to classify our test data, which may not come from a university page and as a consequence the data features or distribution may be different, based on the structure or topics of the new web page. Therefore, we can not directly apply the classifier trained on the university web page 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 can transfer knowledge from one domain of pages into another web page domain.[?]
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 can 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 that it can be learned on the source

domain (one product category) and can help to learn the classification model (other product category).[?]

3.2 Definition of Transfer Learning

After more practical examples are 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 $\mathcal{F}$ and a marginal probability distribution $P(\mathbf{X})$. 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 $\mathcal{F}_{\mathcal{Z}} \neq \mathcal{F}_{\mathcal{X}} \vee P(x) \neq P(z)$.[?, . 542]
A task $\mathcal{T}$ with a given domain D is put together with a label set $\mathcal{Y}$ and a classifier $f(\cdot)$, which is $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ with $\mathbf{y} \in \mathcal{Y}$. The function $f(\cdot)$ is a predictive function to make prediction for unseen, i. g. new data points $\mathbf{x}$. From a probabilistic viewpoint we can interpret $f(x)$ as $P(y \mid x)$. In other words given x how probable is it that label y is correctly assigned.
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 $\mathcal{Y} = \{1, 2, .., N\}$ with N 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 $\mathcal{Y}_{\mathcal{X}} \neq \mathcal{Y}_{\mathcal{Z}} \vee P(y \mid x) \neq P(y \mid z)$.[?, 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 $\mathbf{Z} = \{(\mathbf{z}_i, y_{\mathcal{Z}_i})\}_{i=1}^{M}$ as source domain data where $\mathbf{z}_i \in \mathcal{X}_{\mathcal{Z}}$ as single observation and $y_{\mathcal{Z}_i} \in \mathcal{Y}_{\mathcal{Z}}$ is the corresponding class label, which forms the whole label vector $\mathbf{y}_{\mathcal{Z}}$. Additional, the target domain data $\mathbf{X} = \{(\mathbf{x}_i, y_{\mathcal{X}_i})\}_{i=1}^{N}$. Analogue, $\mathbf{x}_i$ is a single observation with $\mathbf{x}_i \in \mathcal{X}_{\mathcal{X}}$ and the corresponding class label $y_{\mathcal{X}_i} \in \mathcal{Y}_{\mathcal{X}}$, which is again summarized in $\mathbf{y}_{\mathcal{X}}$.[?, p. 2]
Note that we are doing the assumption here that our problem has by default more than one dimension and therefore we write a single observation as a vector of features. The whole data is therefore 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 $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$.[?, p. 542]*

Summarizing, our problem consist of $\mathcal{Z} = \{\mathcal{X}_{\mathcal{Z}}, P(\mathbf{Z})\}$ and $\mathcal{T}_{\mathcal{Z}} = \{\mathcal{Y}_{\mathcal{Z}}, f_{\mathcal{Z}}(\mathbf{Z})\}$ with $f_{\mathcal{Z}}(\mathbf{Z}) = P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z})$ and $f_{\mathcal{X}}(\mathbf{X}) = P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$, respectively. 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.
According to our challenges from section 3.1, the definition can be explained with the help of the web document classification example. 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 in either the feature space $\mathcal{X}_{\mathcal{Z}}, \neq \mathcal{X}_{\mathcal{X}}$ 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 focusing on different topics, which would be lead to different terms.[**?**]
The second case is that the tasks are different, which implies $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$ and furthermore can divided into 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 second 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 a relationship somehow between the feature spaces of the two domains, then they are *related*.[**?**]
Note that in the context of transfer learning the term Domain Adaption (DA) may appear. Based on the discussion in [**?**], the following definition can be extracted.

Definition 2 (Domain Adaptation) *Given a source domain $\mathcal{Z} = \{\mathcal{X}_{\mathcal{Z}}, P(\mathbf{Z})\}$ and learning task $\mathcal{T}_{\mathcal{Z}} = \{\mathcal{Y}_{\mathcal{Z}}, f_{\mathcal{Z}}(\mathbf{Z})\}$ with $f_{\mathcal{Z}}(\mathbf{Z}) = P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z})$ and a target domain $\mathcal{X} = \{\mathcal{X}_{\mathcal{X}}, P(\mathbf{X})\}$ and learning task $\mathcal{T}_{\mathcal{X}} = \{\mathcal{Y}_{\mathcal{X}}, f_{\mathcal{X}}(\mathbf{X})\}$ with $f_{\mathcal{X}}(\mathbf{X}) = P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$. Let $\mathcal{T}_{\mathcal{Z}} = \mathcal{T}_{\mathcal{X}}$ and the domains $\mathcal{Z} \neq \mathcal{Z}$ in a way that $\mathcal{X}_{\mathcal{Z}} = \mathcal{X}_{\mathcal{X}}$ 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 covariance shift problem.[**?**] It assumes that the domain data generation is made in a way that the sampling, which the data is base on, is $P(\mathbf{y}|\mathbf{X})P(\mathbf{X})$. Furthermore, the marginal distribution $P(\mathbf{X})$ changes through the domain shift so that $P(\mathbf{X}) \neq P(\mathbf{X})$ for source domain and target domain data sampling, respectively.[**?**, p. 8-9]
In practice, we can observe that the key challenge of Domain Adaption methods is to focus on aligning the differences in the marginal distribution, which can be observed in [**?**], [**?**], [**?**] and [**?**], which describe by [**?**] too.

3.3 Measurement of Difference

From the previous examples 3.1 and definitions 3.2, it seems that the domains are somehow 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 domain distributions.

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 **X** and **Y** drawn from p and q respectively introduced by [**?**, 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 having the same distribution.[**?**, p. 7-8]
The MMD is calculated by samples drawn from each of the distributions. It measures the distance between the expectations of two samples, $\mathbf{X} = \{x_1, ...x_m\}$ and $\mathbf{Y} = \{y_1, ...y_n\}$ in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$. Assuming that $\mathbf{X} \sim p$ and $\mathbf{Y} \sim q$. Furthermore, if $\mathcal{H}$ is a Hilbert space, then $f(x) = \langle f, \phi(x) \rangle_{\mathcal{H}}$ with $\phi(x) = k(x, \cdot)$ as positive definite kernel which does the feature mapping. Another important characteristics of a RKHS is, that it is continues and has the reproducing property.[**?**, p. 36] Finally, the MMD and his empirical estimation is defined as (3.1) and (3.2):[**?**, p. 726-727]

$$MMD[\mathcal{F}, p, q]^2 := \left[sup_{\|f\|_{\mathcal{H}} \leq 1}(E_{x \sim p}[f(x)] - E_{y \sim q}[f(y)])\right]^2 \tag{3.1}$$

$$MMD_e[X, Y] := \left\|\frac{1}{M}\sum_{i=1}^{M} f(x_i) - \frac{1}{N}\sum_{i=1}^{M} f(y_i)\right\|_{\mathcal{H}} \tag{3.2}$$

Where $f : \mathcal{X} \to \mathbb{R}$ is a member of the function class $\mathcal{F}$.
This means that considering the RKHS $f(x)$ and $f(y)$ are based on the same $\phi(\cdot)$, i. e. kernel function for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}$.[**?**, p. 36]
Note that the empirical MMD is biased. If p and q both are equal probability distributions, respectively $p = q$, then the $\text{MMD}_\text{e} = 0$.[**?**, 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, it can be divergence can be computed.[**?**]
The code used in this thesis can be obtained from the website of Gatsby Computational Neuroscience Unit[1], which uses the standard Gaussian kernel as feature mapping.

[1]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 or -Distance. It needs two probability distributions f and g of a random Variable X. For discrete variables $x_i, i = 1, .., m$ with $p_i = f(x_i)$ and $q_i = g(x_i)$ the KLD is calculated as:

$$KL(g \mid f) = \sum_{i=1}^{m} p_i \log \frac{p_i}{q_i} \tag{3.3}$$

For continues variables the KLD is computed:

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

For all f and g it is $KL(g \mid f) \geq 0$. Next $KL(f \mid f) = 0$ and finally, if $KL(g \mid f) = 0$, then $f = g$, which implies that the two distribution are nearly the same. Note although it is sometimes named as distance it is not a distance because it is not symmetric and does not satisfy the triangular inequality.[**?**, 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 [**?**]. The source code can be found at github[2].

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 labelled 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 combined with the attributes are related to some traditional machine learning task.[**?**]
As we will see there are some intersections with the homogeneous and heterogeneous transfer learning approach in 3.5 and 3.6.[**?**, p. 5-6]
In general, we will stick to these definitions, but these are more general than the settings. Therefore the settings are used to formulate the categories of the transfer learning methods more precisely. Additionally, but more importantly, we want to focus on the fact, whether labelled or unlabeled target data are required for the knowledge transfer. The methods will be explained within the section of homogeneous and heterogeneous transfer learning. The relationship is shown in figure 3.1.

[2]https://github.com/pberkes/neuro-kl

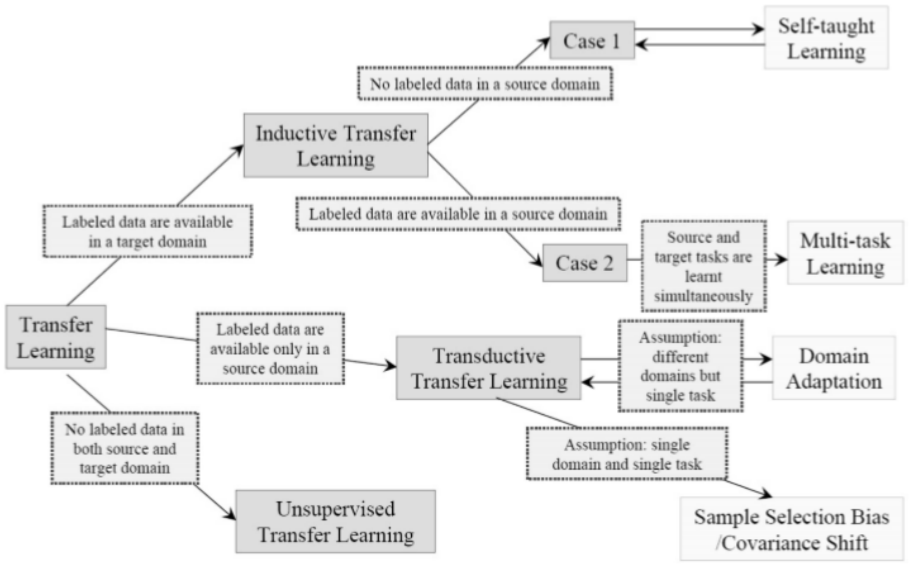

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure 3.1: The various settings of transfer learning approaches and there relation to traditional machine learning problems.[?]

3.4.1 Inductive Transfer Learning

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

Definition 3 (Inductive Transfer learning [?]) *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 labelled test data but lots of labelled 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.[?]
In the second case, there are no labelled source data at all available. The goal is similar to the self-thought learning setting. The challenge of self-thought learning is 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 can not be used directly, we have to find are parts of the data which can be combined with a few labelled target data to reach the goal.[?]
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 aligning the marginal distributions with some labelled data, which is little different to the original definition of inductive learning.

3.4.2 Transductive Transfer Learning

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

Definition 4 (Transductive Transfer Learning [?]**)** *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, which 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.[?]
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 3.6. The second case is that the feature spaces are the same, but the marginal probability distributions differ, i. e. $P(\mathbf{Z}) \neq P(\mathbf{X})$.[?]
In this case, the transductive transfer learning is related to the domain adaptation task from 2. An example of transductive transfer learning method is given in [?]. In this, they cluster the unlabeled target data for pseudo label generation and using a custom dimension reduction with the clustered target data and the labelled 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 one, the unsupervised transfer learning section, which is naturally not part of the category of supervised learning, but will be described for completeness. It assumes that in both domains no labelled data is available. Furthermore, the task of the source and target data are different. In comparison with the supervised learning transfer learning methods, the goal is not to find labels for the target data but solving clustering, dimensionally reduction and density estimation.[?] Finally, we can give the following definition.

Definition 5 (Unsupervised Transfer Learning [?]**)** *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 $\mathcal{T}_{\mathcal{Z}}$, where $\mathcal{T}_{\mathcal{Z}} \neq \mathcal{T}_{\mathcal{X}}$ 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 centres. 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.[?]

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 is one is to align the marginal distribution difference. The second is to correct the difference in the conditional probability distributions, and the last one tries to align both, the marginal and conditional distribution difference.[?, p. 6] This is formalised in the same way as the previous definition, which is influenced by the formulation of Pan et al. from [?].

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 $\mathcal{T}_{\mathcal{T}}$, 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 $\mathcal{X}_{\mathcal{Z}} = \mathcal{X}_{\mathcal{X}}$ and $\mathcal{Y}_{\mathcal{Z}} = \mathcal{Y}_{\mathcal{X}}$, but $P(\mathbf{Z}) \neq P(\mathbf{X})$ and/or $P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z}) \neq P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$.[?, p. 4]*

If the differences only exit in the marginal distribution, then we can build another relation to Domain Adaption from definition 2. Refer to section 3.1, for examples on why the distribution can be different.[?, p. 6-7]

The homogeneous transfer learning category can be split up 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.[?, p. 6-7] Unlike the previous definitions, settings and explanations, which describing the problem and only giving an abstract solution with the predictive function.

3.5.1 Instance-transfer

The instance transfer method tries to full fill the goal of aligning the marginal distribution by reweighting some source data. This reweighted 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 [?, p. 6]
In the following, we will present such an Instance transfer algorithm. The so-called *TrAdaBoost* which is proposed by Dai et al.[?]

TrAdaBoost

The TrAdaBoost[3] algorithm extends the boosting-based learning algorithm. The Algorithm follows the instance transfer definition and assumes that the marginal probability distribution are different, but the conditional probability distribution are the same.[?] It is originally proposed as two class solution, but there are extension like MsTrAdaBoost for multi-class problems.[?]
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 labelled 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.[?]
The voting filters via the boosting approach. It boosts the part of the data which is not too far away from the same-distribution training data. The part of the source data, which is outvoted and therefore has a different distribution is together with the near 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 with weighting.[?]
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 iterations is T and the unlabeled target dataset $\mathbf{X}$. Consider $\mathbf{Z}_D$ as diff-distribution

[3]https://github.com/LinZhineng/transfer-learning/tree/master/tradaboost

training dataset with labels of size N and the same-distribution data (from the target data) with labels $\mathbf{Z}_S$ has M points. This forms the complete training set $\mathbf{Z} = \mathbf{Z}_D \cup \mathbf{Z}_S$, with $N + M$ entries, with $\mathbf{Z} = \{(\mathbf{z}_i, y_i))\}$, where y_i is the label for the ith-data point with $y \in 0, 1$ and x_i with [?]:

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

In every iteration, the learner is trained with this training set and predicts for the training $\mathbf{Z}$, and the unlabeled target dataset $\mathbf{X}$. The prediction in this context is called a hypothesis. In this article, the training loss is defined as the difference between the training data prediction and the ground truth label. Furthermore, the prediction error is the difference between the ground truth target labels and the prediction. Note that the label for $\mathbf{Z}_S$ is predicted, too.[?]
At this point the filtering takes place. If a diff-distribution data point is mistakenly predicted, then it may be 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 [?]:

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

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 ϵ_t is a weighted error function and $\beta = 1/1 + \sqrt{2 \ln n/T}$. Note that ϵ_t function value is required to be less than 0.5.
In equation (3.5) can be seen that the same distribution training data uses another weighting function instead of the weighting function for the diff-distribution training data. This causes that the weight gain for the target data can be much higher as for the source data.
The β, 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 $\{\lceil T/2 \rceil, \dots, T\}$ are considered.[?]
We proceed by describing some theoretical ideas of the *TrAdaBoost*, which is claimed to be theoretical similar to the underlying *AdaBoost*. The convergence rate of the algorithm over the diff-distribution training data $\mathbf{Z}_D$ can be bound by $\mathcal{O}\big(\sqrt{\ln N/T}\big)$ With an increasing number of iterations, the error rate can be minimized. 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.

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

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

3.5.2 Symmetric-Feature-Transfer

The solutions, which implementing 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 Transfer Component Analysis (TCA), which will be discussed in the following.[?, p. 6]

Transfer Component Analysis

The TCA method is presented by Pan et al. in [?][4]. The approach originally supports unsupervised, and semi-supervised learning. However, the resulting kernel and the transformation matrix can be used to train a baseline learner like the SVM. It has the assumption, which makes it a homogeneous transfer method that the conditional distribution is the same, i. e. $P(\mathbf{y}_\mathcal{Z}|\mathbf{Z}) = P(\mathbf{y}_\mathcal{X}|\mathbf{X})$, but the marginal probability distribution are different. Furthermore they assume that there exists a feature transformation ϕ, which can align the differences, i. e. $P(\phi(\mathbf{Z})) \approx (\phi(\mathbf{X}))$ and preserve $P(\mathbf{y}_\mathcal{Z}|\phi(\mathbf{Z})) \approx P(\mathbf{y}_\mathcal{X}|\phi(\mathbf{X}))$.[?]
Additionally, we will see that TCA requires target data in the training process and is therefore transductive.
In general there may exits underlying data structures and relationships in between them. Some of them do cause the distribution difference, and some of them are the 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.[?]
Pan et al. are using their previous work from [?]. In this they have presented the Maximum Mean Discrepancy Embedding (MMDE), which is an extension of the MMD metric defined in section 3.3.1. It states that finding a transformation which minimizes the MMDE, then transformation will draw the marginal distribution closer together.
Consider **K** as a cross-domain kernel, as a composition of the source and target

[4]https://github.com/LinZhineng/transfer-learning/tree/master/tca

domain, in the embedded subspace which has the form:[?]

$$\mathbf{K} = \begin{bmatrix} K_{\mathcal{Z}} & K_{\mathcal{Z}\mathcal{X}} \\ K_{\mathcal{X}\mathcal{Z}} & K_{\mathcal{X}\mathcal{X}} \end{bmatrix} \tag{3.7}$$

Where $\mathbf{K} \in \mathbb{R}$ with size $N \times N = (N_{\mathcal{Z}} + N_{\mathcal{X}}) \times (N_{\mathcal{Z}} + N_{\mathcal{X}})$. 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 is rewritten as $tr(\mathbf{KM})$. Where $\mathbf{M}$ is the MMD matrix as:[?]

$$(M)_{ij} = \begin{cases} \frac{1}{N_{\mathcal{Z}}^2}, & \mathbf{x_i}, \mathbf{x_j} \in \mathcal{Z} \\ \frac{1}{N_{\mathcal{X}}^2}, & \mathbf{x_i}, \mathbf{x_j} \in \mathcal{X} \\ \frac{-1}{N_{\mathcal{Z}} N_{\mathcal{X}}}, & otherwise \end{cases} \tag{3.8}$$

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. Furthermore, the MMDE and the resulting TCA objective function can be formulated as:[?]

$$\max_{\mathbf{K} \succeq 0} tr(\mathbf{KM}) - \lambda tr(\mathbf{K}) \tag{3.9}$$

In this objective, if the first term is minimized, then the aligning happens. Furthermore, if the second term, which is the constrain of the optimization problem, is maximized, then the subspace variance is maximized. The parameter $\lambda \geq 0$ is a trade off parameter and However, this seems that the kernel from equation (3.7) is expensive to calculate via a Semi Definite Program (SDP). A SDP is an extension of a linear optimisation problem, where the non-negative constrained is replaced with $\mathbf{K} \succeq 0$, which is the PSD attribute of the Matrix. Additionally, the vector space for linear problems $\mathbb{R}^N$ is replaced with the vector space of symmetric $N \times N$ matrices.[?] Furthermore, with this objective function, they can not achieve both in a good manner: Aligning and maximizing the variance. Finally, there is no dimensional reduction done.[?]

To get rid of this problems, they approximate the kernel in the following way. First, they are using the empirical kernel map. With this, the kernel can be rewritten as $K = KK^{-1}K = (KK^{-1/2})(K^{-1/2}K)$.[?] The empirical kernel map is designed to approximate kernels for new patterns by incorporating a finite set of training patterns.[?]

Moreover to create the subspace they integrating a transformation matrix to the empirical kernel map. Consider $\tilde{\mathbf{W}} \in \mathbb{R}^{N \times M}$, where M is the number of reduced dimensions. Then the subspace kernel can be decomposed as:[?]

$$\tilde{\mathbf{K}} = (\mathbf{KK}^{-1/2}\tilde{\mathbf{W}})(\tilde{\mathbf{W}}^T\mathbf{K}^{-1/2}\mathbf{K}) = \mathbf{KWW}^T\mathbf{K} \tag{3.10}$$

With $\mathbf{W} = \mathbf{K}^{-1/2}\tilde{\mathbf{W}}$. Revisiting MMD, it can also be reformulated using equation (3.10) to write an MMD measure in the subspace, which is $MMD' = tr(\mathbf{W}^T\mathbf{KMKW})$. Note that the variance matrix of the subspace is defined as $\mathbf{W}^T\mathbf{KHKW}$ with size $N \times N$, where $\mathbf{H} = \mathbf{I}_{N\times N} - (1/N)\mathbf{1}_{(N\times N)})$ is the centering matrix with $\mathbf{1}$ as all ones matrix. The goal is now to determine $\mathbf{W}$. Considering the new subspace kernel and the variance in the subspace, the optimisation problem from (3.9) can be rewritten as follows:[?]

$$\begin{gathered}\min_{\mathbf{W}} = tr(\mathbf{W}^T\mathbf{KMKW}) + \lambda tr(\mathbf{W}^T\mathbf{W}) \\ s.t. \mathbf{W}^T\mathbf{KHKW} = \mathbf{I}_{N\times N}\end{gathered} \tag{3.11}$$

Where λ is again the trade-off parameter. Pan et al. proposed that with this the latent spaced spanned by the learned components, the variance is obtained, and the marginal distribution is aligned.
Although they showed that the constraint from problem (3.11) is non-convex, it can still be efficiently solved by reorganising:[?]

$$\max_{\mathbf{W}} = tr((\mathbf{W}^T(\mathbf{KMK} + \lambda\mathbf{I})\mathbf{W})^{-1}\mathbf{W}^T\mathbf{KHKW}) \tag{3.12}$$

Furthermore, they mentioned solving (3.12) is equivalent to finding the L leading eigenvectors of $(\mathbf{KMK} + \lambda\mathbf{I})^{-1}\mathbf{KHK}$. The details of the proof are discussed in [?] Finally, the (simple) algorithm for TCA can be given: First compute the kernel according to (3.7), $\mathbf{M}$ from (3.8) and create the centring matrix. Second create $\mathbf{W}$ by searching the L eigenvectors with $(\mathbf{KMK}+\lambda\mathbf{I})^{-1}\mathbf{KHK}$. With this, the subspace kernel can be computed, or the source and target data can be reduced with $\mathbf{W}$. The computational complexity is given by $\mathcal{O}(L(N_{\mathcal{Z}} + N_{\mathcal{X}})^2)$, where again L is the number of eigenvectors.[?]

Geodesic Flow Kernel

The next symmetric feature transfer method is the Geodesic Flow Kernel (GFK) and is proposed by Gong et al. in [?][5]. Following definition of homogeneous methods and similar to TCA, the Geodesic Flow Kernel is trying to lower the marginal distribution differences.[?, p. 13]
Furthermore, it can be seen as unsupervised transfer learning method, but because the results of this method is, in fact, a kernel, it can be trained with a kernel machine.[?] Again because it needs target data in the learning process, it will be considered as inductive transfer learning method.
The idea of GFK can be summarized in the following steps: The source and the target data has to be embedded in a Grassmann manifold. Consider a subspace $\mathcal{P} \in \mathbb{R}^{N\times d}$ with N observations (dimensionality of the data) and d is the feature

[5]http://www-scf.usc.edu/ boqinggo/domain_adaptation/GFK_v1.zip

space dimension. Then the collection of the d dimensional subspaces are forming the Grassmannian $\mathbb{G}(d, N)$ in the original N dimensional feature space. Next, construct a geodesic flow and merge source and target data as points in it with a certain distance between them. Proceed by integrating an infinite number of subspaces along the flow $\mathbf{\Phi}(t)$. After that, the raw features $\mathbf{x}$ are projected into the 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 the original feature space. The kernel in the space $\mathcal{H}^{\infty}$ should then encapsulate the underlying difference and commonness of the source and target data.[?] The approach is summarised in figure 3.2
In order to create and apply the Geodesic Flow Kernel, there are four major steps necessary. First, reduce the dimension of the original feature space to create the embedding subspace $\mathbf{P}_{\mathcal{Z}}$ and $\mathbf{P}_{\mathcal{X}}$ with $\mathbb{R}^{N \times d}$ as N samples and d dimensions. This is done via the Principal Component Analysis (PCA).[?]
The PCA is a linear transformation and uses the d dominant eigenvectors, which are orthogonal to each other, of the covariance matrix of the data to transform the data into a lower d dimensional subspace. With this, the PCA tries to maximize the projected variance.[?, p.561-563]
Furthermore, the d as the number of dimensions becomes a free parameter. Note that in according to [?], originally the optimal number of dimension is obtained analytically. However, it seems that in practice (public implementation from above, which is used in chapter 5) the dimensionally is, in fact, a free parameter and therefore this determination will not be further discussed.
The second step is to construct the geodesic flow. This means projecting the samples on the flow, which is done by using the transformation $\mathbf{\Phi} : t \in [0, 1] \rightarrow \mathbf{\Phi}(t) \in G(d, N)$. Note the constraints $\mathbf{\Phi}(0) = \mathbf{P}_{\mathcal{Z}}$ and $\mathbf{\Phi}(1) = \mathbf{P}_{\mathcal{X}}$, which are the 'start' and 'end' points. For other $t \in (0, 1)$ they construct the following transformation:[?]

$$\mathbf{\Phi}(t) = \mathbf{P}_{\mathcal{Z}}\mathbf{U}_1\mathbf{\Gamma}(t) - \mathbf{R}_{\mathcal{Z}}\mathbf{U}_2\mathbf{\Sigma}(t) \tag{3.13}$$

Where $\mathbf{R}_{\mathcal{Z}} \in \mathbb{R}^{N \times (N-d)}$ is the orthogonal complement of the source subspace with $\mathbf{P}_{\mathcal{Z}}\mathbf{R}_{\mathcal{Z}} = 0$. Furthermore $\mathbf{U}_1 \in \mathbb{R}^{d \times d}$ and $\mathbf{U}_2 \in \mathbb{R}^{(N-d) \times d}$ are orthogonal matrices which are given by the pair of Singular Value Decomposition (SVD) values.[?]
In general the SVD has the form of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, which can decompose a rectangular or symmetric matrix $\mathbf{A}$.[?].
The SVD is applied and reorganized to:

$$\mathbf{P}_{\mathcal{Z}}^T\mathbf{P}_{\mathcal{X}} = \mathbf{U}_1\mathbf{\Gamma}\mathbf{V}^T, \quad \mathbf{R}_{\mathcal{Z}}^T\mathbf{P}_{\mathcal{X}} = -\mathbf{U}_2\mathbf{\Sigma}\mathbf{V}^T \tag{3.14}$$

Where $\mathbf{\Gamma}$ and $\mathbf{\Sigma}$ are $d \times d$ diagonal matrix for the transformation (resulting from SVM), with elements $\cos(\theta_i)$ and $\sin(\theta_i)$. Furthermore they are the 'overlapping' between the two subspaces, which is the smallest series of angles between the bases.[?]

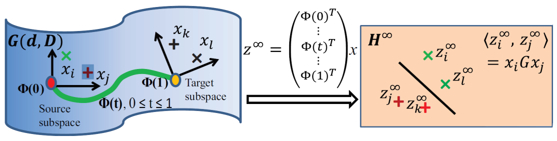

$G(d,D)$
x_i
x_j
$\Phi(0)$
Source subspace
$\Phi(t), 0 \leq t \leq 1$
x_k
x_l
$\Phi(1)$
Target subspace
$z^\infty = \begin{pmatrix} \Phi(0)^T \\ \vdots \\ \Phi(t)^T \\ \vdots \\ \Phi(1)^T \end{pmatrix} x$
H^∞
$\langle z_i^\infty, z_j^\infty \rangle = x_i G x_j$
z_i^∞
z_l^∞
z_j^∞
z_k^∞

Figure 3.2: The geodesic flow with the start and endpoints from source and target data and the induced kernel.

The θ_i in the matrices have a ascending order. For $\mathbf{\Sigma}(t)$ are the diagonal matrix $\cos(t\theta_i)$, the other diagonal transformation matrix consists of $\sin(t\theta_i)$.[?]
The third step is to compute the GFK itself. The created geodesic flow parametrises how smooth the source data changes to the target data through the subspaces. Consider a training data in subspace, which is near to the target endpoint, then a classifier trained on this subspace projection can use the characteristics of both domains and vice versa. Therefore Gong et al. conclude to use every subspace, which is in the flow.[?]
Although, they have may an infinite number of subspaces, the problem of involving 'all' subspaces is solved by threatening t as a continues variable with $\mathbf{\Phi}(t) \in (0,1)$ and solving the integral of inner products between zero and one.

Furthermore, the problem of calculating the inner product of two infinite vectors is in fact solved by the kernel trick:[?]

$$\langle \mathbf{z}_i^\infty, \mathbf{z}_i^\infty \rangle = \int_0^1 (\mathbf{\Phi}(t)^T \mathbf{x}_i)^T (\mathbf{\Phi}(t)^T \mathbf{x}_j) dt = \mathbf{x}_i^T \mathbf{G} \mathbf{x}_j \tag{3.15}$$

Where $\mathbf{G} \in \mathbb{R}^{D \times D}$ is the PSD kernel.
The kernel trick is known, to induce the inner product between two infinite dimensional vectors and the kernel is computed as:[?]

$$G = \begin{bmatrix} \mathbf{P}_\mathcal{Z}\mathbf{U}_1 & \mathbf{R}_2\mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}_1 & \mathbf{\Lambda}_2 \\ \mathbf{\Lambda}_4 & \mathbf{\Lambda}_3 \end{bmatrix} \begin{bmatrix} \mathbf{U}_1^T \mathbf{P}_\mathcal{Z}^T \\ \mathbf{U}_2^T \mathbf{R}_\mathcal{Z}^T \end{bmatrix} \tag{3.16}$$

The $\mathbf{\Lambda}$ are diagonal matrices and preserving the angles between the subspaces. For details please see the derivation in the thesis [?, p. 45;110-113].
With the previous steps the kernel is applicable, to use it with a kernel machine.[?]

3.5.3 Asymmetric-Feature-Transfer

The 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 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.[?, p. 6; 10]

Joint Distribution Adaptation

The Joint Domain Adaption algorithm tries to not only solve the marginal probability distribution but the conditional probability distribution, too. This solution is proposed by Long et al. in [**?**].[6]
Because it solves the differences between the domains, it can be considered as transductive setting, with the need for 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. However, this problem is assumed as non-trivial, and therefore they created an approximation to solve it. They approximate it by searching an adaptation matrix $\mathbf{A} \in \mathbb{R}^{K \times M}$, which can transform any new data in the subspace and is, in fact, an orthogonal transformation matrix.[**?**]
Furthermore, the embedding matrix $\mathbf{Z}$ is searched to transform the already existing data in the subspace.

This approach consists of the four steps, which are feature transformation, marginal distribution adaptation, conditional distribution adaptation and solving the resulting optimisation problem.[**?**]
Revisiting our transfer learning definition, source and target data can be represented as data matrix $T = [\mathbf{Z}; \mathbf{X}] \in \mathbf{R}^{N \times M}$ where $N = N_{\mathcal{Z}} + N_{\mathcal{X}}$, with the number of source and target data respectively. The adaptation matrix is using the PCA, as standard dimensionality reduction method, coupled with a centring matrix, which is $\mathbf{H} = \mathbf{I} - \frac{1}{N}\mathbf{1}$. It is represented as optimization problem which:[**?**]

$$\max_{\mathbf{A}^T\mathbf{A}=\mathbf{I}} = tr\big(\mathbf{A}\mathbf{X}^T\mathbf{H}\mathbf{X}\mathbf{A}^T\big) \tag{3.17}$$

However, the end of the day the adaptation matrix is found by solving the eigenvalue equation with $\mathbf{X}^T\mathbf{H}\mathbf{X}\mathbf{A} = \mathbf{A}\mathbf{\Phi}$ where $\mathbf{\Phi} \in \mathbb{R}^{K \times K}$ is the diagonal matrix with the K largest eigenvalues, which finally results in $\mathbf{Z} = \mathbf{A}\mathbf{X}^T$, with $\mathbf{Z} \in \mathbb{R}^{N \times K}$. But equation (3.17), will be important later on.[**?**]
The marginal distribution is adapted by using the empirical MMD from equation (3.1), by incorporating the adoption matrix, the MMD will compare the source and target means in the K dimensional subspace:[**?**]

$$\min_{\mathbf{A}^T\mathbf{A}} tr(\mathbf{A}\mathbf{X}^T\mathbf{M}_0\mathbf{X}\mathbf{A}^T), \tag{3.18}$$

With $\mathbf{M}_0$ as MMD matrix to compute the empirical estimate in matrix form. It has

[6]http://ise.thss.tsinghua.edu.cn/ mlong/doc/joint-distribution-adaptation-iccv13.zip

the following elements.

$$(M_0)_{ij} = \begin{cases} \frac{1}{N_{\mathcal{Z}} N_{\mathcal{Z}}}, & \mathbf{x_i}, \mathbf{x_j} \in \mathcal{Z} \\ \frac{1}{N_{\mathcal{X}} N_{\mathcal{X}}}, & \mathbf{x_i}, \mathbf{x_j} \in \mathcal{X} \\ \frac{-1}{N_{\mathcal{Z}} N_{\mathcal{X}}}, & otherwise \end{cases} \tag{3.19}$$

This is done because of estimating the probability density for distribution is often considered as a non-trivial problem. Furthermore, the differences in the marginal distributions are still large. By minimising equation (3.18) in a way that equation (3.17) is maximised, the domains are drawn closer in the subspace. Which is in fact similar to TCA from 3.5.2.[?]
It seems that adopting the conditional probability distribution is a kind of bigger problem. Since the goal is to be able to solve the transfer problem without labelled target data the $P(\mathbf{y}_{\mathcal{X}}|\mathbf{X})$ cannot be directly modelled or estimated with some statistics like it is done by the marginal adaptation.[?]
Because of this, they created the pseudo label technique. This is done by applying a base classifier, e. g. SVM to predict the not 'final' class labels for the target domain, which are the pseudo labels. Since $P(\mathbf{y}_{\mathcal{Z}}|\mathbf{Z})$ is unknown, the class conditional probability function in form of $P(\mathbf{X}|\mathbf{y}_{\mathcal{X}} = c)$, with the true class labels and $P(\mathbf{X}|\mathbf{y}_{\mathcal{X}} = c)$, with the pseudo labels, the conditional probability distributions can be matched. Here c represents the classes of the label space $\mathcal{Y}$, i. e. $c \in 1, \dots, C$ for C labels. By incorporating the class conditional distribution in equation (3.18), not the sample means, but the distance between the class conditional distributions are measured.[?]

$$\left\| \frac{1}{N_{\mathcal{Z}}^{(c)}} \sum_{\mathbf{x}_i \in \mathcal{Z}^{(c)}}^{N_{\mathcal{Z}}^{(c)}} \mathbf{A}\mathbf{x}_i^T - \frac{1}{N_{\mathcal{X}}^{(c)}} \sum_{\mathbf{x}_j \in \mathcal{X}^{(c)}}^{N_{\mathcal{X}}^{(c)}} \mathbf{A}\mathbf{x}_j^T \right\|^2 = \min_{\mathbf{A}^T\mathbf{A}} tr(\mathbf{A}\mathbf{X}^T\mathbf{M}_c\mathbf{X}\mathbf{A}^T) \tag{3.20}$$

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

$$(M_c)_{ij} = \begin{cases} \frac{1}{N_{\mathcal{Z}}^{(c)} N_{\mathcal{Z}}^{(c)}}, & \mathbf{x_i}, \mathbf{x_i} \in \mathcal{Z}^{(c)} \\ \frac{1}{N_{\mathcal{X}}^{(c)} N_{\mathcal{X}}^{(c)}}, & \mathbf{x_i}, \mathbf{x_i} \in \mathcal{X}^{(c)} \\ \frac{-1}{N_{\mathcal{Z}}^{(c)} N_{\mathcal{X}}^{(c)}}, & \begin{cases} \mathbf{x_i} \in \mathcal{Z}^{(c)}, \mathbf{x_j} \in \mathcal{X}^{(c)} \\ \mathbf{x_j} \in \mathcal{Z}^{(c)}, \mathbf{x_i} \in \mathcal{X}^{(c)} \end{cases} \\ 0, & otherwise \end{cases} \tag{3.21}$$

Again by minimising equation (3.20) it will maximise equation (3.17). 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.[?]
With this tools, the last step can be made. The optimisation problem is made to minimise the differences in both types of distributions. This means they reformulated equation (3.17) with (3.18) and (3.20) to:[?]

$$\min_{\mathbf{A}\mathbf{X}^T\mathbf{M}_c\mathbf{X}\mathbf{A}^T=\mathbf{I}} \sum_{c=0}^{C} tr(\mathbf{A}\mathbf{X}^T\mathbf{M}_c\mathbf{X}\mathbf{A}^T) + \lambda\|\mathbf{A}\|_F^2 \tag{3.22}$$

Where λ is again the regulation parameter to guarantee a distinct function for the optimisation problem.
In fact, the PCA is designed for linear problems.[?, p. 428-429] To get rid of this constraint, Long et al. used the Representer Theorem, for example, defined in [?, p. 90], to a kernel-JDA for a non-linear problem. Note that it is similar to equation (3.22) when the data matrix $\mathbf{X}$ is replaced with some kernel $\mathbf{K}$. Revisiting the merged optimisation function, they showed how to formulate the problem as Lagrange function. This problem is solved by analysing the derivatives concerning the $\mathbf{A}$, i. e. $\delta L/\delta\mathbf{A}$, which results in the following in the eigendecompension:[?]

$$\left(\mathbf{X}^T \sum_{c=0}^{C} \mathbf{M}_c\mathbf{X}^T + \lambda\mathbf{I}\right)\mathbf{A}^T = \mathbf{X}^T\mathbf{H}\mathbf{X}\mathbf{A}^T\mathbf{\Phi} \tag{3.23}$$

Where $\mathbf{\Phi} = diag(\phi_1, \ldots, \phi_n)$ are the Lagrange multiplier. The problem from equation (3.22) is reduced to finding the K smallest eigenvectors for the eigenproblem of equation (3.23).[?]
Finally, we can describe the presented algorithm for JDA. First construct the $\mathbf{M}_0$ for the marginal distribution alignment problem from equation (3.18) and setting $\{\mathbf{M}_c\}_{c=1}^{C} = 0$ for the pseudo label matrix. Furthermore, repeat the eigendecomposition in equation (3.23), the training and prediction from the classifier for the pseudo labels and construct the MMD matrix $\mathbf{M}_0$ from equation (3.21) until convergence.[?]
With that, a classifier trained in the subspace and the adaptive matrix $\mathbf{A}$ can be obtained. Through $\mathbf{A}$, any new points can be transformed in the subspace.[?]
The JDA complexity for $T \leq 50$ as the number of iterations until convergence and the subspace bases with $K \leq 500$ the computational complexity is given by $\mathcal{O}(TKM^2 + TCN^2 + TMN)$.[?]

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 reweight them to create an improved target learner. One approach in this category is based Multi Model Knowledge Transfer (MMKT).[?, p. 7-8]

Multi Model Knowledge Transfer

The MMKT approach is proposed by Tommasi et al. in [?]. 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 se in the following. Furthermore, they want a classifier, which can be trained only on few training examples and being able to learn more only from few data than the standard LS-SVM does. The transfer knowledge method is adopted from his previous work in [?].
Note that this method does not try to explicit minimise the distribution differences, as per definition of the homogeneous transfer learning from 6. Again, consider a labeled source dataset with $\{\mathbf{z}, y_i\}_{i=1}^N$, where $y_i \in \mathcal{Y}_{\mathcal{Z}} = \{1, -1\}$. Furthermore, the feature mapping function $\phi(\cdot)$, which maps the features in the infinite space and the inducing kernel function $\mathbf{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$. Revisiting the LS-SVM, it has the following optimisation problem for determining the hyperplane parameter $\mathbf{w}$ and the bias b.[?]

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{N}[y_i - \mathbf{w}\phi(\mathbf{x}_i) - b]^2 \tag{3.24}$$

It can be shown that the optimal value of $\mathbf{w}$ is found by $\mathbf{w} = \sum_{i=1}^N \alpha_i \phi(\mathbf{x}_i)$. Furthermore, $\boldsymbol{\alpha}$ and b can be found by solving:[?]

$$\begin{bmatrix} \mathbf{K} + \frac{1}{C}\mathbf{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} \tag{3.25}$$

Where $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is a vector of ones The solution is found by inverting the left matrix of the left-hand side, which is further called $\mathbf{G}$.[?]
Now suppose that we have prior knowledge from K 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. $\mathbf{w}'_j, j = \{1, \dots, K\}$, for each class.[?]
With that Tommasi et al. assuming that the transfer happens when the constraints of the hyperplane of the new class $K+1$ are set in a manner that it is close to the already trained classes of K. Keeping this in mind, the optimisation problem can be rewritten as model for multi knowledge transfer:[?]

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w} - \sum_{j=1}^{K}\beta_j \mathbf{w}'_j\|^2 + \frac{C}{2}\sum_{i=1}^{N}\zeta_i[y_i - \mathbf{w}\phi(\mathbf{x}_i) - b]^2 \tag{3.26}$$

With $\mathbf{w}'_j$ as a parameter for the already trained model of K models and ζ_i as a parameter for weighting, which helps to balance the contribution of positive and negative examples in the training data in equation (3.26). It is determined with the following chase:[?]

$$\zeta_i = \begin{cases} \frac{N}{2N^+}, & y_i = +1, \\ \frac{N}{2N^-}, & y_i = -1 \end{cases} \tag{3.27}$$

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:[?]

$$\mathbf{w} = \sum_{j=1}^{K} \beta_j \mathbf{w}'_j + \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i) \tag{3.28}$$

In this the $\mathbf{w_j}'$ is weighted through β_j. Because of the reformulated optimization problem the optimal determination of $\boldsymbol{\alpha}$ and b can be expressed as:[?]

$$\begin{bmatrix} \mathbf{K} + \frac{1}{C}\mathbf{W} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} \tag{3.29}$$

Where $\mathbf{W} = diag\{\zeta_1^{-1}, \dots \zeta_N^{-1}\}$ with ζ_i as defined as (3.27).
The $\beta_j \in (0,1)$ parameter forms $\boldsymbol{\beta} = \{\beta_1, \dots, \beta_K\}$ for controlling the degree of closeness of the new model to the j-th model. Note that $\boldsymbol{\beta}$ has to be choose 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 $\boldsymbol{\beta}$ has to be found.[?]
In this method the error of the LS-SVM is quantified with the Leave One Out (LOO)-Error.[?]
The 'classical' definition of the LOO-Error is, as the name says, 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.[?, p.74-76]
They have incorporated this error with the LS-SVM and obtained the LOO-Error with:[?]

$$r_i^{-1} = y_i - \tilde{y}_i \frac{\alpha_i}{=} \mathbf{G}_{ii}^{-1} - \sum_{j=1}^{k} \beta_j \frac{\alpha'_{i(j)}}{\mathbf{G}_{ii}^{-1}} \tag{3.30}$$

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

is the matrix obtained, when the i-th sample is omitted.
Tommasi et al. showed that the best values for β_j are those minimising equation (3.30). However minimising directly would result in a non-convex formulation with many local minima. Therefore they formulated a loss function, which is rather similar to the hinge loss in SVM, which leads to a smooth function:[?]

$$loss(y_i, \tilde{y}_i) = \zeta_i max[1 - y_i\tilde{y}_i, 0] = max\left[y_i\zeta_i\left(\mathbf{G}_{ii}^{-1} - \sum_{j=1}^{k}\beta_j\frac{\alpha'_{i(j)}}{\mathbf{G}_{ii}^{-1}}\right), 0\right] \tag{3.31}$$

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

3.5.5 Relational-Knowledge-Transfer

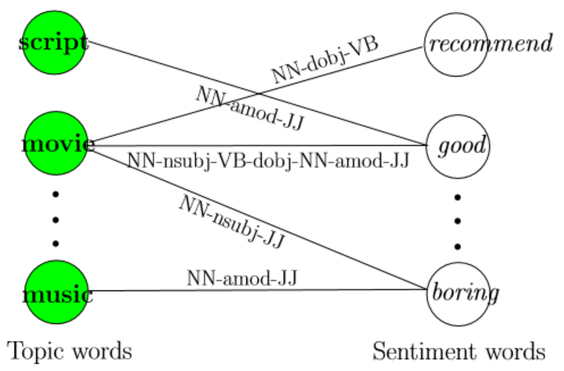
The relational knowledge transfer aim to find some relationship between the source and target data.[?, p. 7] Because the Transfer Kernel learning algorithm from section 4.2 finds this relationship in the eigenvalues of source and training data, we could assign this category to the method. The details of the algorithm are discussed in this chapter because, the Probabilistic Classifcation Transfer Kernel Vector Machine is based on it.

3.5.6 Domain Dependent Transfer Learning

The previous methods of transfer learning are all generic solutions. They are not domain depended, for example are not working with only with text. However, there are solutions, which are not domain independent. For example, the Relational Adaptive bootstrapping (RAP) algorithm, which will be very briefly discussed to point out the key features of domain-dependent transfer learning method, in this case, text-dependent. Note that the RAP algorithm originally is part of the relational-knowledge transfer category and homogeneous.[?, p. 6-8]
The RAP algorithm is proposed by Li et al. in [?]. Because they are using labelled target data in the learning process, the method is inductive. The primary goal is to create a transfer method in the field of opinion mining. In this field sentiment, lexicon construction and topic lexicon extraction are two subtasks. The sentiment lexicon is a list of expressions, which are used to indicate the polarity of a sentiment, which is decided in positive or negative. This lexicon is domain depended, e. g. for films, because people are using different terms to describe their opinions in different domains.[?]

Domain	Review
camera	The **camera** is *great*.
	it is a very *amazing* **product**.
	i highly *recommend* this **camera**.
	takes *excellent* **photos**.
	photos had some *artifacts* and *noise*.
movie	This **movie** has *good* **script**, *great* **casting**, *excellent* **acting**.
	I *love* this **movie**.
	Godfather was the most *amazing* **movie**.
	The **movie** is *excellent*.

(a) Small Value



(b) Large Value

Figure 3.3: The selection of topic and sentiment words and the graph to transfer sentiment words over topics.[**?**]

On the other hand, there is a topic lexicon, which has a list of topic-based terms to express their opinion. They are topic depended too. Both aspects are important because users not only care about the sentiment in general but which aspects are mentioned.[**?**]
In comparison with the other transfer learning methods, they have a very 'low' level understanding of the data. In other words, they care about texts not vectors in feature space in the first place. In general, they distinguish terms in three categories. The first one is the sentiment word, which includes sentiment terms in general. The second category is devoted for topic terms, and the last category is assigned when a word is neither sentiment nor a topic term. Furthermore, consider many ratings in a domain, i e. Film review. The goal of the algorithm is to create a graph, where the sentiment and topic words over the domain are connected to identify sentiment in another domain better. The process is summarized in figure 3.3. In sub-figure 3.3a the topic words are bold, and the sentiment words are italics. In the other sub-figure 3.3b, the cross-domain graph connects the sentiment with topics over the domains. Note that the 'weights' on the graph are shortest paths of word types (Point of Speech-Tags) to get from a topic term to the sentiment term.

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 distribution of the feature spaces are equal, but the feature spaces or the label space are not equal.[**?**, 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. [**?**, p. 6]
We can again give the 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 $\mathcal{T}_{\mathcal{T}}$, heterogeneous 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 $P(\mathbf{Z}) = P(\mathbf{X})$, but $\mathcal{X}_{\mathcal{Z}} \neq \mathcal{X}_{\mathcal{X}}$ and $\mathcal{Y}_{\mathcal{Z}} \neq \mathcal{Y}_{\mathcal{X}}$.[**?**, p. 4]*

Because the examples in challenges, section 3.1, may not covering the heterogeneous definition, 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 $\mathcal{X}_{\mathcal{Z}}$ which has M metrics and the prediction should be done for a different software which is reviewed on metric $\mathcal{X}_{\mathcal{X}}$ and N metrics, then it is a heterogeneous transfer problem.[**?**, p 3-4]
Again, there is a distinction between asymmetric and symmetric feature space transformations. The first approach category aims to transfer the source domain in the target domain to align the differences. Solutions from the second category, are trying to find a common latent feature space where the differences of the feature spaces are aligned.[**?**, p. 19]
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 [**?**]. It is considered as symmetric transfer, and it can be regarded as domain depended to image classification with a text extension and is, therefore, no general solution.[**?**, 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) to a latent feature space called *topic space*, as we will see in the following.
However, first, the general motivation of the authors are explained. Note that in the previous transfer learning methods the source and target domains, where separated by learning the source domain and evaluating at the target domain. This is not the case by TTI because source and target are just symbolic for two domains, which should be merged (transformed). However, the method, in general, requires target

domain data.[?]
In general, image classification suffers from a few problems. 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. For example, a text is described with a sparse vector of words, but an image is represented with vectors based on colour histograms or texture features. With that, it is no surprise that text classification works 'naturally' better because they have the ability of semantic interpretation and the labels are often designed according to the semantic criteria, which is more difficult with images.[?]
Another point is that in comparison with text classification there an abundant amount of labelled text, but not that amount of labelled data in the image domain. The authors claim that through Wikipedia, which uses a category index to 'label' documents, or other web text collection websites, the amount of labeled text documents are very large. However, at this point the improvement takes place. There are various websites and social networks, which have images linked to the text. This text can be used to increase the amount of target data (images) when this text can be transformed and used for image classification. They are using these co-occurrences (Image linked to a text, which is linked to a category) to build a semantic bridge from text to images. This bridge can further be used to transfer semantic of text into images. With that unlabeled images but linked with text can be used in the classification process as labelled data. [?]
In general, the text is not transformed into the image feature space. Both the text and image features are transformed to the topic space. The goal is now to find a transformation matrix. Revisiting the previous notation from, we can formulate the source (text) domain as $\mathcal{Z}$, where the samples are $\mathbf{Z} \in \mathbb{R}^{N_S \times A}$ 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 $\mathbf{X} \in \mathbb{R}^{N_T \times B}$ or a single image with $\mathbf{x_i} \in \mathbb{R}^B$. However, we have the assumption, that the size of images is much smaller than the size of the text.[?]
The linkage through the co-occurrence of text and image is then formalized with $\mathcal{C} = \left\{ \left(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l, c(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l) \right) \right\}$. This means that $\bar{\mathbf{z}}_k$ is the corresponding text feature vector to an image feature vector $\bar{\mathbf{x}}_l$. The $c(\cdot)$ function is abbreviated with $c_{k,l}$.[?]
The so called *translator* function with $T : \mathbb{R}^A \times \mathbb{R}^B \to \mathbb{R}$ weights the linkage strength between a document and image. Now the label for an image is found with:[?]

$$f_T(\mathbf{x}) = \sum_{i=1}^{N_S} y_i T(\mathbf{z}_i, \mathbf{x}) \tag{3.32}$$

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, which maximises the classification accuracy or rather minimises a loss function, which is in general formulated with:[?]

$$\min_{T} = \gamma \sum_{j=1}^{N_T} l\big(y_j \cdot f_T(\mathbf{x}_j)\big) + \lambda \sum_{C} \chi\big(c_{k,l} \cdot T(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l)\big) + \Omega(T) \tag{3.33}$$

Note that y_j, in this case, is the image label. The parameter γ and λ 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 (3.33) the general performance of the translation process is measured and can be divided into three parts.
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 summation over the C weighted co-occurrences $c_{k,l}$ with a monotonic exponential decreasing function $\chi = \exp(-z)$. They pretend with this choice, the optimisation problem is in a closed algebraic format and is easy to optimise. The last term $\omega(T)$ is for regulating the learning of the translator. It should be useful when the auxiliary examples of image are fewer.[?]
Another aspect is to transfer the images and documents into the underlying latent topic space. They extend the translation function to do a transformation into the P dimensional space, with P topics, where the inner product of a document and a image representing the weight of a certain topic. In other words how good does the document with the image matching. This is done via:[?]

$$T(\mathbf{z}_i \mathbf{x}_j) = \langle \mathbf{W}_{\mathcal{Z}} \mathbf{z}_i, \mathbf{W}_{\mathcal{X}} \mathbf{x}_i \rangle = \mathbf{z}_i \mathbf{W}_{\mathcal{Z}} \mathbf{W}_{\mathcal{X}}^T \mathbf{x}_i^T = \mathbf{z}_i \mathbf{S} \mathbf{x}_i^T \tag{3.34}$$

With $\mathbf{S} = \mathbf{W}_{\mathcal{Z}} \mathbf{W}_{\mathcal{X}}^T$ with size $A \times B$ and $\mathbf{W}_{\mathcal{Z}}$ and $\mathbf{W}_{\mathcal{X}}$ are the transformation matrix respectively.
In the following it is sufficient to just learn $\mathbf{S}$ rather than the transformation matrices. Note with that the integration of $\mathbf{S}$ the regularisation term changes. They incorporating the Frobenius norm, which can be found here [?], and the trace norm, for example defined in [?], to reformulate $\Omega(\cdot)$ to:

$$\|\mathbf{S}\|_{\Sigma} = \inf_{\mathbf{S} = \mathbf{W}_{\mathcal{Z}} \mathbf{W}_{\mathcal{X}}^T} \frac{1}{2} \left(\|\mathbf{W}_{\mathcal{Z}}\|_F^2 + \|\mathbf{W}_{\mathcal{X}}\|_F^2 \right) = \Omega(T) \tag{3.35}$$

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 SVD. They argue that by minimising this function, will result in the minimum topics used to explain the correspondence between text and image. Furthermore by incorporating equation (3.35) in equation (3.33), the final problem

can be obtained:[?]

$$\min_{S} = \gamma \sum_{j=1}^{N_T} l\big(y_j \cdot f_T(\mathbf{x}_j)\big) + \lambda \sum_{C} \chi\big(c_{k,l} \cdot T(\bar{\mathbf{z}}_k, \bar{\mathbf{x}}_l)\big) + \|\mathbf{S}\|_{\Sigma} \tag{3.36}$$

For finding the optimum, they used a proximal gradient method to find an approximation of the optimal solution. Note that equation (3.36) can be expressed as $F(S) + \|\mathbf{S}\|_{\Sigma}$.[?]
In order to find a good approximation with the proximal gradient method, they are using the Taylor expansion, also used in [?, p. 214], and the coefficient α to quadratically approximate it at the point $\mathbf{S} = \mathcal{S}\tau$, which has the form:[?]

$$\mathcal{Q}(\mathbf{S}, \mathbf{S}_\tau) = F(\mathbf{S}_\tau) + \langle \nabla F(\mathbf{S}_\tau), \mathbf{S} - \mathbf{S}_\tau \rangle + \frac{\alpha}{2}\|\mathbf{S} - \mathbf{S}_\tau\|_F^2 + \|\mathbf{S}\|_{\Sigma} \tag{3.37}$$

The approximation is adequate if $F(\mathbf{S}_{\tau+1}) + \|\mathbf{S} + \mathbf{1} \leq \mathcal{Q}(\mathbf{S}, \mathbf{S}_\tau)$. For further information about the algorithm, the reader may consider [?].

3.7 Negative Transfer

A major key assumption of transfer learning is that the source and target domains are somehow 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 relate, then the transfer learning solution can impact the performance of the underlying classifier for the target data in a negative way.[?, 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 relationship exists or not.[?, 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 transfer learning community and may not consider the fact of negative transfer. However, the negative transfer is a phenomenon that exits, 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

In this chapter are various transfer learning definitions and methods discussed. It is again worth to mention that there are not only many relationships 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.[?, p. 5-6] However, fair enough, many of the definitions are useful to distinguish how the transfer

is done. For example, is labelled 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 and furthermore one have to additional manual label target data before a solution can be used. Therefore the mentioned definition of inductive transfer 3 and transductive transfer 4 are solid extension to the homogeneous 3.5 and heterogeneous 3.6 transfer learning definitions.
In contrast to the mentioned survey papers of [?] and [?], we choose to derive into the transfer learning methods into detail, to get an idea of how the general problem is formulated. Furthermore, the process of how the get a solution for the problem is described in detail, instead of giving a high-level description. This seems reasonable, because of the already pointed out relations among the solutions, which does not omit the need for further research, but has a more detailed understanding of how the transfer problem can be solved.
Another point worth to mention is that the definitions and solutions may vary between supervised, semi-supervised and unsupervised. However, as seen in this chapter, the output of the solutions are often kernels, transformation or embedding matrices, which can be used with any kernel-based algorithm. However, to be careful, for example from [?], [?] [?] appears that their solutions can be applied to any standard classifier. In practice, from the documentation of the standard Matlab implementation of the Support Vector Machine[7] appears that they do not support pre-calculated kernels. This is the same with the standard k-nearest neighbour implementation[8]. Additionally, in the implementation of [?] they are using their implementation of k-nearest neighbour. The point is that the many current standard implementations do not support external information in general, e. g. Kernels, or may have dependencies in the algorithm itself, see section 4.3, and therefore the integration may need further investigation.

[7]https://de.mathworks.com/help/stats/fitcsvm.html
[8]https://de.mathworks.com/help/stats/classificationknn-class.html

Chapter 4

Integration of Transfer Learning

In this chapter, two versions of the Probabilistic Classifcation Transfer Kernel Vector Machine (PCTKVM) are proposed. The PCTKVM which handles the θ as fixed parameter and the PCTKVM$_{\theta\mathrm{Est}}$, which does a theta estimation. The extensions 'TK' to original term PCVM should indicate the new Kernel approach to solve the transfer learning problem. As already mentioned in 3 there are various solutions to solve the transfer learning problem. Revisiting section 3.5, the homogeneous transfer learning problem can be summarised in four approaches.
However, Long et al. in [?] proposed a different approach, which differs from the approaches and therefore called it transfer kernel learning. However, because the relational knowledge approach transfers information through defined relationships between the source and target data, from section 3.5.5, the TKL suits well in this category. Furthermore, because it needs target data for the learning process and according to the definition of transductive transfer learning from section 3.4.2, it can be categorised 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.[?] On the other side [?], proposed a PCVM with linear costs which is also achieved via the Nyström approximation. Therefore it seems reasonable to go one step further an combine these two ideas to form a transfer learning 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 prelabeled test data. As we will see in the following the TKL using only the training and test dataset as it is 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 5.2.3 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).[**?**]
It is done by selecting N randomly observations (rows/columns) without replacement of $\mathbf{K}$ for calculating the approximation.[**?**] Considering a symmetric positive semi definite kernel $k(\mathbf{z}, \mathbf{x})$ which satisfies Mercers Theorem, then K has the following properties and can be represented as:[**?**]

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

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

$$\int_{i=1}^{N} k(\mathbf{x}, \mathbf{z}) \phi_i(\mathbf{z}) p(\mathbf{z}) d\mathbf{x} = \lambda_i \phi_i(\mathbf{x}) \tag{4.2}$$

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

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

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

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

Where N are the number of used observation from the kernel and the N eigenfunctions are still orthogonal. If we consider the N columns from $\mathbf{K}$ 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 $i, j = 1, \ldots, N$, then we can use this for the eigenvalue problem of this kernel:[**?**]

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

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

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

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

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

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

4.1.1 Nyström Kernel Learning

Willams 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 $\mathcal{O}(N^3)$, with N as the number of training examples, to find a solution.[?]
With the Nyström approximation the complexity for predictors can be bound by $\mathcal{O}(M^3 N)$ with using a smaller system with size M, where $M < N$.[?]
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, using the technique of Nyström, a kernel $\mathbf{K}$ which satisfies Mercers theorem is needed.[?] Furthermore, we draw M IID samples of $\mathbf{K}$ and therefore we can rewrite the symmetric kernel according to:[?]

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

Where $\mathbf{K}_{M,M}$ is the sample matrix and $\mathbf{K}_{N-M,M}$ is the submatrix of $\mathbf{K}$, which is the kernel of the sample, 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 from the general eigenvalue problem, which can obtained from [?, p. 221] and shown in (4.5), a kernel can be calculated with $\tilde{\mathbf{K}} = \tilde{\mathbf{U}}\tilde{\mathbf{\Lambda}}\tilde{\mathbf{U}}^T = \sum_{i=1}^{p} \tilde{\lambda}_i^{(N)} \tilde{\mathbf{u}}_i^{(N)} (\tilde{\mathbf{u}}_i^{(N)})^T$ and is in this case the approximation matrix.[?] 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 choosing $P \leq M < N$ as the size of the subset and applying the equations of (4.6) and (4.7), the approximation bases on the approximation sample matrix can be made with:[?]

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

With $i = 1, \dots, P$ and $u_i^{(M)}$ and $\lambda_i^{(M)}$ from the eigenproblem and from the $M \times M$ sample kernel $K_{M,M}$.
After obtaining the approximated eigenvalues and vectors the full approximated kernel can be restored:[?][?]

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

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:

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

From (4.11) can be seen that the kernel approximation can fully be done by the sample matrix and the submatrix $K_{N-M,N}$ of the 'original' kernel.[?]
The Nyström Kernel approximation is bounded by the error $E_{Nys} = ||\mathbf{K} - \tilde{\mathbf{K}}||$.[?] This error can be minimised by choosing the samples carefully and a reasonable amount of it. If the original Kernel $\mathbf{K}$ has to rank M and the samples are linearly independent columns from the kernel, then the approximation is exact. As a consequence from above choosing $P < M$ the error rises the fewer columns are chosen.[?] The problem with choosing (sampling) columns/rows is that we usually do not know which columns are linearly independent. May, therefore, some sampling methods or approximations are made in the past to make this sampling more deterministic. For example, [?], which tries to find good samples through various sampling techniques, or [?], where the authors are drawing a large sample but approximate the submatrices with SVD.
To summarise the Nyström Kernel approximation, first, we select P of N 'good' samples from the data reorganising the kernel according to (4.8) At this point, we have two options. First calculating the kernel just for the $\mathbf{K}_{P,P}$ and the submatrix $\mathbf{K_{N-P,P}}$, 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 4.11. Note that the second one needs a matrix inversion, which should be reflected when it comes to for large datasets.

4.2 Domain Invariant Kernel Learning

The Transfer Kernel Learning (TKL) approach is made by Long et al. in [?][1]. They are interpreting the kernel approximation from a different viewpoint. The main idea is to see the train dataset-matrix, which has to be approximated, regarding Nyström, which will be approximated by the test dataset. Proceed by finding eigenvalues in a way that the difference between the approximation and ground truth train kernel

[1]http://ise.thss.tsinghua.edu.cn/ mlong/doc/transfer-kernel-learning-tkde15.zip

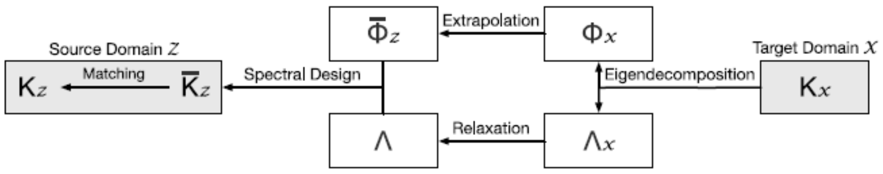


Figure 4.1: The Process of the Domain Invariant Kernel learning.[?]

will be minimal. This forms a domain invariant kernel for transfer learning.[?] The main Process is shown in figure 4.1.
From the above consider $\mathbf{z}$ as $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{N_{\mathcal{Z}}}\}$ as training dataset which is sampled from $p(\mathbf{Z})$ within training domain $\mathcal{Z}$ and $\mathbf{x}$ as $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_{\mathcal{X}}}\}$ 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 $\mathbf{K}_{\mathcal{Z}}$ for training and $\mathbf{K}_{\mathcal{X}}$ for testing.
As fundamental insight of the RKHS is that any Positive Semidefinite Kernel (PSD) kernel can be reconstructed based on the eigensystem of it. Furthermore any new datasets can be incorporated within a kernel.[?]
This means given a kernel $\mathbf{K}_{\mathcal{X}}$ 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 $\mathbf{K}_{\mathcal{Z}}$. Again we could approximate the needed training kernel through:[?]

$$\mathbf{K}_{\mathcal{Z}} \simeq \mathbf{U}_{\mathcal{Z}} \mathbf{\Lambda}_{\mathcal{X}} \mathbf{U}_{\mathcal{Z}}^T = \mathbf{K}_{\mathcal{Z}\mathcal{X}} \mathbf{K}_{\mathcal{X}}^{-1} \mathbf{K}_{\mathcal{X}\mathcal{Z}} \tag{4.12}$$

Where $\mathbf{U}_{\mathcal{Z}}$ as eigenvectors of the training kernel, $\mathbf{\Lambda}_{\mathcal{X}}$ as eigenvalues of the target kernel and $\mathbf{K}_{\mathcal{Z}\mathcal{X}} = \mathbf{K}_{\mathcal{Z}\mathcal{X}}^T$. 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, where we have taken in section 3.2 the assumption that test and training feature spaces following different probability distributions. This means that $P(\mathbf{z}) \neq P(\mathbf{z})$, the error of the approximation from (4.12) can be arbitrary large.[?]
However, to align the distribution difference is one goal of transfer learning. The main improvement done by Long et al. is to tackle this problem and to learn the domain invariant approximated kernel $\bar{\mathbf{K}}_{\mathcal{A}}$.[?]

4.2.1 Learning Approach

From a 'kernel' viewpoint the aligning distribution of data, where the kernel is base on, can be formulated as $P(\phi(\mathbf{z})) \simeq P(\phi(\mathbf{x}))$.[?] However, handling the data distribution in Hilbert space is difficult, because the kernel-induced feature map cannot be explicitly represented.[?]
Therefore Long et al. take the assumption that it is sufficient that $\mathbf{K}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{X}}$. 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 size of the kernels is not equal, i. g. $\mathbf{K}_{\mathcal{Z}} \in \mathbb{R}^{N_{\mathcal{Z}} \times N_{\mathcal{Z}}}$ and $\mathbf{K}_{\mathcal{X}} \in \mathbb{R}^{N_{\mathcal{X}} \times N_{\mathcal{X}}}$ with $N_{\mathcal{Z}} \neq N_{\mathcal{X}}$. Therefore the extrapolated source kernel $\bar{\mathbf{K}}_{\mathcal{Z}} \in \mathbb{R}^{N_{\mathcal{Z}} \times N_{\mathcal{Z}}}$, by using the eigensystem of the target kernel $\mathbf{K}_{\mathcal{X}}$. The kernel $\bar{\mathbf{K}}_{\mathcal{Z}}$ will be compared to the actual training kernel $\mathbf{K}_{\mathcal{Z}}$.[?]
For clarity the terms, Long et al. proposed the term extrapolated instead of approximation.[?]
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:[?]

$$\bar{\mathbf{U}}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{Z}\mathcal{X}} \mathbf{U}_{\mathcal{X}} \mathbf{\Lambda}_{\mathcal{X}}^{-1} \tag{4.13}$$

According to Nyström, the eigenvalues $\mathbf{\Lambda}_{\mathcal{X}}$ of the test kernel are indispensable. However, in this approach a *relaxation* of the eigenvalues is done by considering the eigenvalues as learnable parameters $\mathbf{\Lambda} = diag[\lambda_1, \ldots, \lambda_{N_{\mathcal{X}}}]$. Therefore the kernel extrapolation can be expressed as:[?]

$$\bar{\mathbf{K}}_{\mathcal{Z}} = \bar{\mathbf{U}}_{\mathcal{Z}} \mathbf{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T \tag{4.14}$$

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 viewpoint is that the kernel is extrapolated from the target data but evaluated by the training data.[?]
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 definite kernel, then the produced kernel will positive semi-definite itself.[?]. In general kernel matrices which are positive semi-definite have only non-negative eigenvalues.[?, p. 30]
Therefore it seems reasonable to set $\lambda \geq 0$ to guarantee a PSD kernel. The error of the approximation $\bar{\mathbf{K}}_{\mathcal{Z}}$ to the original ground truth kernel $\mathbf{K}_{\mathcal{Z}}$ is determined using the squared loss, which is presented in the following:[?]

$$\begin{gathered}\min_{\mathbf{\Lambda}} ||\bar{\mathbf{K}}_{\mathcal{Z}} - \mathbf{K}_{\mathcal{Z}}||_F^2 = ||\bar{\mathbf{U}}_{\mathcal{Z}} \mathbf{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T - \mathbf{K}_{\mathcal{Z}}||_F^2 \\ \lambda_i \geq \zeta \lambda_{i+1}, i = 1, \ldots, N_{\mathcal{X}} - 1 \\ \lambda_i \geq 0, i = 0, \ldots, N_{\mathcal{X}}\end{gathered} \tag{4.15}$$

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

4.2.2 Learning Algorithm

In the above section, we defined the squared error loss function for the TKL approach. In this subsection, we will introduce the resulting optimisation 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 constrains 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:[?]

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

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

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

Where $\mathbf{I} \in \mathbb{R}^{N_{\mathcal{X}} \times N_{\mathcal{X}}}$ as the identity matrix and $\odot$ as hardamatrix multiplication. Note that the eigenspectrum dumping factor ζ is the only free parameter which needs to be tuned because the $\mathbf{\Lambda}$ is learned within the optimisation problem.
Finally, similar to equation 4.8, we can reconstruct the 'joint' Kernel:[?]

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

Where $\mathcal{A} = \mathcal{Z} \cup \mathcal{X}$ as one dataset and therefore it forms the matrix $\bar{\mathbf{U}}_{\mathcal{A}} = [\bar{\mathbf{U}}_{\mathcal{A}}; \mathbf{U}_{\mathcal{X}}]$, which are the extrapolated eigenvectors of the whole dataset.
The complete TKL algorithm is summarized in algorithm 2.[?] The underlying kernel

Algorithm 2 Transfer Kernel Learning Algorithm

```
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 $\{\mathbf{\Lambda}_{\mathcal{X}}, \mathbf{U}_{\mathcal{X}}\}$ like (4.5).
3: Extrapolate for source eigensystem $\bar{\mathbf{U}}_{\mathcal{Z}}$ with (4.13).
4: Solve the QP for eigenspectrum $\mathbf{\Lambda}$ with (4.16).
5: Merge results and return it as (4.18).
```

which is calculated in the first row of algorithm 2 is the Gaussian Kernel from (5.4). 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 this errors, if $\mathbf{\Lambda} = \mathbf{\Lambda}_{\mathcal{X}}$, then the

errors of $E_{NystKer}$ and E_{TKL} are equal. However, because the $\mathbf{\Lambda}$ is a free parameter and the approximation error is directly minimised, the error of the TKL approximation is bound by $E_{TKL} \leq E_{NystKer}$.[?]
In many real-world problems, the eigenvalues are following the power law distribution. This means that there a few large eigenvalues and many small, in comparison with the large, eigenvalues.[?] 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 amount of eigenvectors is therefore fixed to $R = min(500, N_{\mathcal{X}})$ and therefore the eigenvectors can be reduced to $\bar{\mathbf{U}}_{\mathcal{Z}} \in \mathbb{R}^{R \times R}$ or $\lambda \in \mathbb{R}^{R \times N_{\mathcal{X}}}$.[?]
The complexity of the TKL algorithm can be given with $\mathcal{O}((D + R)(N_{\mathcal{Z}} + N_{\mathcal{X}})^2)$, where R denotes the number of used eigenvectors, D refers to the kernel.[?]

4.3 PCTKVM Algorithm

In this section the Probabilistic Classifcation Transfer Kernel Vector Machine algorithm will be discussed in detail. As shown above we have obtained our transfer learning kernel with algorithm 2. According to [?] 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.

$$\bar{\mathbf{K}}_{\mathcal{Z}} = \bar{\mathbf{U}}_{\mathcal{Z}} \mathbf{\Lambda} \bar{\mathbf{U}}_{\mathcal{Z}}^T \tag{4.19}$$

However, simply using it in algorithm 1 has some disadvantages. Revisiting the PCVM algorithm the kernel is recalculated in every iteration based on the optimised theta from the previous iteration. As a consequence, we have to recalculate the entire transfer kernel too. With the complexity of the PCVM which is $\mathcal{O}(K^3)$ with K as number of basis functions. We consider the worst case and substitute the number of basis function with the number of training samples $N_{\mathcal{Z}}$, which results in $\mathcal{O}(N_{\mathcal{Z}}^3)$. The complexity of the TKL is $\mathcal{O}((D+R)(N_{\mathcal{Z}}+N_{\mathcal{X}})^2)$, combining the two, we would end up with a computational complexity of $\mathcal{O}(N_{\mathcal{Z}}^3(D+R)(N_{\mathcal{Z}}+N_{\mathcal{X}})^2)$. This may be resulting in a long computational time. In fact, the question is, whether we explicit need the theta optimisation in every iteration or not. However, one advantage of the PCVM was, to optimise the parameters within and therefore omit the need for cross-validation. Furthermore, the performance of the PCTKVM depends hardly on the quality of the theta, which can be seen in figure 4.2. To keep this advantage and the performance up, a new approach must be found. One approach could be to optimise 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

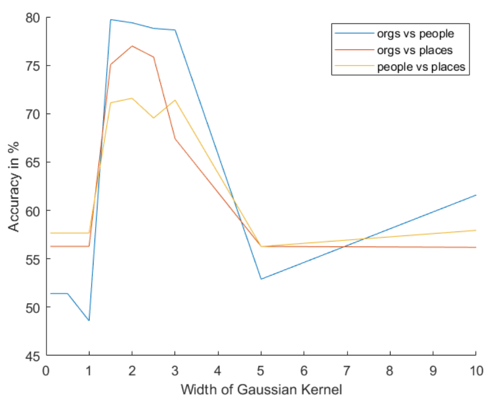


Figure 4.2: The performance of the PCTKVM algorithm for different thetas shown as accuracy in %, evaluated at the Reuters dataset.

value, where we optimised 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 optimisation completely. Fair enough we can observe a good performance, which can be seen in section 5. Furthermore, we could obtain the complexity of $\mathcal{O}(N_{\mathcal{Z}}^3+(D+R)(N_{\mathcal{Z}}+N_{\mathcal{X}})^2)$. If we consider D and R as constant and by multiplying out we obtain $\mathcal{O}(N_{\mathcal{Z}}^3 + N_{\mathcal{X}}^2)$ for the PCTKVM with a fixed theta.

4.3.1 Theta Estimation

Another, more deterministic but heuristic, approach to obtain a theta is presented by Kitayama in [?] 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 4.3.[?] In this the black dots representing the data, the dashed lines are the Gaussian kernel, and the solid lines represent the regression. In 4.3a the width is set to 0.5 and in 4.3b it is set to 1.
They are claiming in general that solving the theta via an optimisation problem is very time-consuming. The reason for this is because the optimisation problem is depending on the data function, which can give a very large amount of variables and purposed a simpler algorithm to find a good θ.[?]
First, they use a scaling technique to scale every dimension to an equal range. Consider $N = N_{\mathcal{Z}} + N_{\mathcal{X}}$ as the number of all samples with M dimensions. In the approach, they using a *Min-Max* Normalization to do it.[?]

$$x_j = \frac{x_j - x_j^L}{x_j^U - x_j^L} \times s, \quad i = 1, \ldots, M \tag{4.20}$$

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 criterium, but not needed in our approach. The author's

recommendation is to set alpha to 1.1. Moreover, therefore 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:[?].

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

Where $\bar{x}_j$ is the mean and σ_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.[?] 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.[?]

$$\theta_i = \frac{d_{i,max}}{\sqrt{N}\sqrt[N]{M-1}}, i = 1, \dots, N \tag{4.22}$$

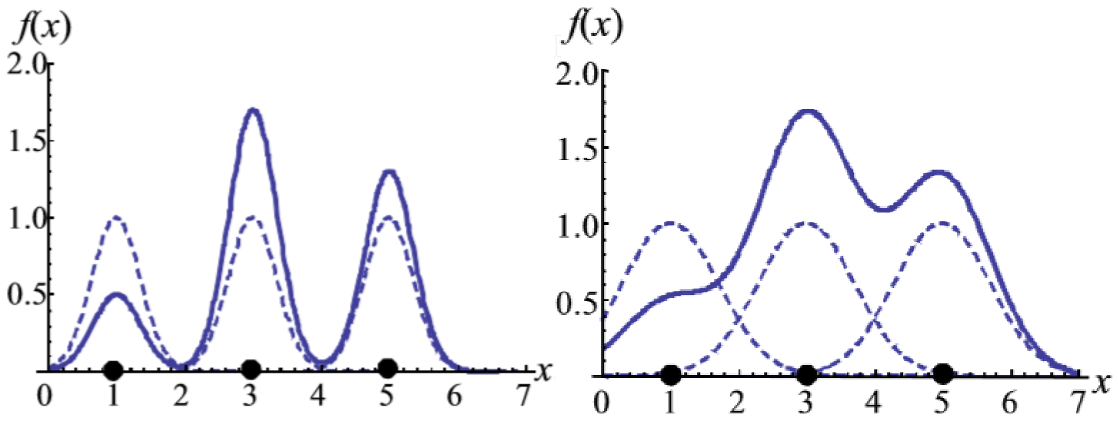
Where $d_{j,max}$ is the maximum distance between the i-th data point and another data point. And furthermore, θ_i is the width for the i-th Gaussian kernel in every row.
This is the main improvement in comparison with other approaches to determine the width because for example $\theta = \frac{d_{max}}{\sqrt[N]{NM}}$ can only be applied to uniform distributed data. However with (4.22) the distance can be applied to non-uniform distributed data too, which is mostly the case in real-world problems.[?]
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 [?]:

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

Step 2: Calculate the distance matrix

Step 3: Find maximum distance for N data points using (4.22)

Step 4: Find the minimum distance for every training data with $\theta_{min} = \min_{1 \le i \le N} \theta_i$ and return θ_{min}



(a) Small Value

(b) Large Value

Figure 4.3: Effect to regression of the width in the Gaussian kernel[**?**]

To calculate the distance in practice, we are using a dissimilarity matrix D. This matrix employs the euclidean distance with $d_{i,j} = ||x_i - x_j||^2, \forall i, j \in N$ as dissimilarity measure. This matrix is symmetric and is zero on the diagonal.[**?**, p. 22,299] Because the Gaussian kernel is used in this approach, the dissimilarity matrix is already involved in the computational complexity. Furthermore, 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. In the tests, we are using the whole dataset including training and testing.
Although we can reduce the complexity with a clever implementation, the complexity discussion is based on the worst case involving all steps to compute. The computational complexity of calculating the dissimilarly matrix is $\mathcal{O}(N^2)$.[**?**] Furthermore, finding a maximum in N rows with M values can be solved in $\mathcal{O}(NM)$. Because the dissimilarity matrix is symmetric the complexity changes to $\mathcal{O}(N^2)$ and multiplying a scalar to N elements costs $\mathcal{O}(N)$, which can be summarised in $\mathcal{O}(N^2 + N) = \mathcal{O}(N^2)$. Again finding a minimum within the theta array from step 4 takes $\mathcal{O}(N)$. Putting it together we can obtain the complexity of the theta estimation with $\mathcal{O}(N^2)$. Note that in practice, we have no insight into the built-in Matlab function, which algorithm or implementation is used. Because of this, the performance may be better for finding a minimum. Furthermore, 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, PCTKVM is presented in algorithm 3. Note that the complete algorithm which involves every detail is found in appendix

Algorithm 3 Probabilistic Classification Transfer Kernel Vector Machine

```
Input: Input Data $\mathbf{K} = [\mathbf{Z};\mathbf{X}]$ as $M$ sized training and $N$ sized text set; $\mathbf{Y}$ as
    $N$ sized training label vector; kernel(-type) $ker$; eigenspectrum dumping factor
    $\zeta$; $\theta$ as kernel paramater; $niter$ as maximal number of iterations; $threshold$ as
    convergence criteria; InitVector as $M$-sized initialization vector.
Output: Weight Vector $\mathbf{w}$; bias $b$, kernel parameter $\theta$; transfer kernel $\bar{\mathbf{K}}_{\mathcal{A}}$.
1: $\mathbf{D}$ = calculate_Dissimilarity_Matrix($\mathbf{K}$);
2: $\theta$ = theta_Estimation($\mathbf{D}$);    ▷ According to section (4.3.1) (optional)
3: $\bar{\mathbf{K}}_{\mathcal{A}}$ = transfer_Kernel_Learning($\mathbf{D}$,$ker$,$\theta$,$\zeta$);    ▷ According to (4.2)
4: [$\mathbf{w}$,$b$] = pcvm_Training($\bar{\mathbf{K}}_{\mathcal{Z}}$,InitVector,$niter$,$threshold$);    ▷ Algorithm (1)
```

C. We observed in practice that the kernel is for some θ not PSD anymore, although he is in theory. We assume that there is some issue in the floating point multiplication for small numbers and therefore added the regularisation term $eps\mathbf{I}_{N\times N}$ to the kernel. This solved the problem.
As it can be seen the PCTKVM the number of free parameters remains the same because the θ is estimated, only the eigenspectrum dumping factor ζ is left. Furthermore, we obtain by combining the complexity of the theta estimation with PCVM and TKL the same complexity as already discussed, which is $\mathcal{O}(N_{\mathcal{Z}}^3 + N_{\mathcal{X}}^2)$. The reason for this is that the dissimilarity matrix is already involved and can be validated in table 5.5.

4.3.3 Prediction

Additionally, we can make predictions based on the provided prediction function of the PCVM with the use of $\bar{\mathbf{K}}_{\mathcal{X}\mathcal{Z}} = \mathbf{U}_{\mathcal{X}}\mathbf{\Lambda}\bar{\mathbf{U}}_{\mathcal{Z}}^T$ as kernel. When it comes to the SVM the kernel has size $N_{\mathcal{Z}} \times N_{\mathcal{X}}$ is used for the prediction according to the support vectors. The suggested prediction function for the SVM has the form $\mathbf{y} = \bar{\mathbf{K}}_{\mathcal{X}\mathcal{Z}}(\alpha \odot \mathbf{y}_{\mathbf{Z}}) + b$, where α are the Lagrange multipliers.[?]
But 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 K non zero weight vectors with $K \ll N$ and because the PCVM uses only kernel rows/columns corresponding to the non zero weight vector index, then our final kernel $\bar{\mathbf{K}}_{\mathcal{K}\mathcal{X}}$ for prediction has size $(K \times N_{\mathcal{X}})$. Therefore the prediction function of the PCTKVM has the form:

$$\mathbf{y} = \bar{\mathbf{K}}_{\mathcal{K}\mathcal{X}}\mathbf{w} + b \tag{4.23}$$

Finally, the class label is obtained from the sign of the elements y_i of the test label vector $\mathbf{y}_{\mathcal{X}}$ which has size $N_{\mathcal{X}}$. The probabilistic output is calculated with the probit link function as the PCVM does it in equation 2.14.

4.3.4 Extensions

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

To solve the multi-class problem algorithmically, there are two main approaches. 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 each class. The label assignment for one class is done in iteration only. However, because this leads to large unclassifiable regions in the feature space.[**?**, 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 $K = C(C-1)/2$. Furthermore, the unclassifiable region in comparison with the one vs rest approach is smaller. At the predication we do again K iteration and predict the whole dataset with two classes.[**?**, p. 127-128]

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

We choose then the final label for the returning 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 lower, the smaller class label is assigned regarding whole numbers, according to the Matlab *mode*[2] function. This is rather unconventional because it is likely to be a random selection, but in our tests, the results are the best. Another approach would be to assign the class label according to the biggest probability given from $\mathbf{P}$.

To determine a final probability output $\mathbf{p}$ for a final label decision with respect to the data point, the algorithm calculates the mean of the probabilistic output for the runs where the label is equal to the final label is assigned.

The number of relevance vectors is determined by using all unique vectors, which are used during the K iterations.

[2]https://de.mathworks.com/help/matlab/ref/mode.html

4.3.5 Further Extensions

As suggested from [?], 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 [?] for it.
Another suggested extension is to adopt the algorithm to 'big data'. Because kernel machine can have at least $\mathcal{O}(N^2)$ or even worse, they may not be applied well to very large 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.[?]
Consider a large test kernel matrix $\mathbf{K}_{\mathcal{X}} \in \mathbb{R}^{N_{\mathcal{X}} \times N_{\mathcal{X}}}$. Then sample from this kernel in a IID manner to form a subset kernel $\mathbf{K}_{\mathcal{X}\hat{\mathcal{X}}} \in \mathbb{R}^{N_{\mathcal{X}} \times \hat{N}_{\mathcal{X}}}$ with $\hat{N}_{\mathcal{X}} \ll N_{\mathcal{X}}$, 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:

$$[?]\mathbf{U}_{\mathcal{X}} = \mathbf{K}_{\mathcal{X}\hat{\mathcal{X}}}\mathbf{U}_{\hat{\mathcal{X}}}\mathbf{\Lambda}_{\mathcal{X}}^{-1} \tag{4.24}$$

Furthermore, we can approximate the source kernel $\mathbf{K}_{\mathcal{Z}}$ with the cross domain kernel $\mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}} \in \mathbb{R}^{N_{\mathcal{Z}} \times \hat{N}_{\mathcal{X}}}$ and the subset kernel $\mathbf{K}_{\hat{\mathcal{Z}}}$. The procedure is similar to (4.24) or with the use of the kernels only, shown in (4.12) or (4.11), by replacing the target kernel with $\mathbf{K}_{\hat{\mathcal{Z}}}$.

$$\mathbf{K}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}}\mathbf{K}_{\hat{\mathcal{Z}}}^{-1}\mathbf{K}_{\hat{\mathcal{Z}}\mathcal{Z}} \tag{4.25}$$

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

$$\bar{\mathbf{U}}_{\hat{\mathcal{Z}}} \simeq \mathbf{K}_{\hat{\mathcal{Z}}\mathcal{X}}\mathbf{U}_{\mathcal{X}}\mathbf{\Lambda}_{\hat{\mathcal{X}}}^{-1} \tag{4.26}$$

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

$$\bar{\mathbf{U}}_{\mathcal{Z}} \simeq \mathbf{K}_{\mathcal{Z}\hat{\mathcal{Z}}}\bar{\mathbf{U}}_{\hat{\mathcal{Z}}}\mathbf{\Lambda}_{\hat{\mathcal{X}}}^{-1} \tag{4.27}$$

With this, we have the required information to solve the Quadratic Program and are making the TKL algorithm large scale and applicable for big data.[?]
Because it is a Nyström approximation, the quality of the approximation depends hardly on the quality of the taken sample, as mentioned in 4.1.1.

4.4 Conclusion

In this chapter, we successfully integrated transfer learning in the PCVM. We successfully integrated a proposed transfer learning method in the PCVM, which forms the PCTKVM. Furthermore, we solved the problem that the algorithm will be slow through the theta optimisation by adopting a simple theta estimation and adjust it to the needs of the PCVM. Because it uses unlabelled target data, the algorithm is transductive and solves the transfer problem with an 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$_{\theta \mathrm{Est}}$ can be obtained from Github[3]. Furthermore, the reader can find the complete code written for this thesis, including all tests, in the repository.

[3]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, the used datasets to determine the performance are defined, and the set of transfer learning classifiers with the corresponding parameters are appointed.
Furthermore, the used statistical methods in the study are introduced, and the statistics are shown and discussed. This study is split up in three main statistics. The first part is a basic descriptive statistic 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 this all datasets described in this thesis are tested.
The procedure and selection of statistics are mainly adopted from [?] and [?]. 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. Note that in this chapter for clarity only the summarised tables are shown. The complete results are shown in appendixB.

5.1 Dataset Description

The study consists of 18 benchmark datasets. Three of them are text-based from Reuters-21578[1] and the remaining twelve are images from Caltech-256[2] and Office. Note that the datasets are not extracted from the raw data, but rather used from others with a pre-processing state. The corresponding articles, which are using this version of datasets are [?] and [?].
This 18 sets are well known in the area of domain adaptation and transfer learning and for example used in [?][?][?][?][?][?] and discussed in [?].
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

[1]http://www.daviddlewis.com/resources/testcollections/reuters21578
[2]http://www.vision.caltech.edu/Image_Datasets/Caltech256/

the same top category or source. The classes itself are subcategories or subsets.[?] The relation will be clear in the separate sections for the datasets respectively.

5.1.1 Text Dataset

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.[?]

It is a frequently used text collection to evaluate the performance of text categorisation techniques. The collection is segmented in five top categories and many subcategories. For testing classifiers, the task is to evaluate on the three big categories *orgs*, *people* and *places*. In the last subcategory, all documents about the 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 done at the collection.[?]

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.[?] Furthermore, stopwords are removed. With the Document Frequency (DF) Thresholding, the numbers of features are cut down. To apply the DF Thresholding the number of documents in which a certain term occurs has to be count. Each term which is less than a previously set threshold is removed from the feature space.[?] 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 document times the inverse document frequency which is the number of occurrences of this term in all documents. [?, 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.[?]

In table 5.1 the dataset with its features, observations, labels and the corresponding divergences, KLD and MMD, are shown. The divergences are defined int 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.

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

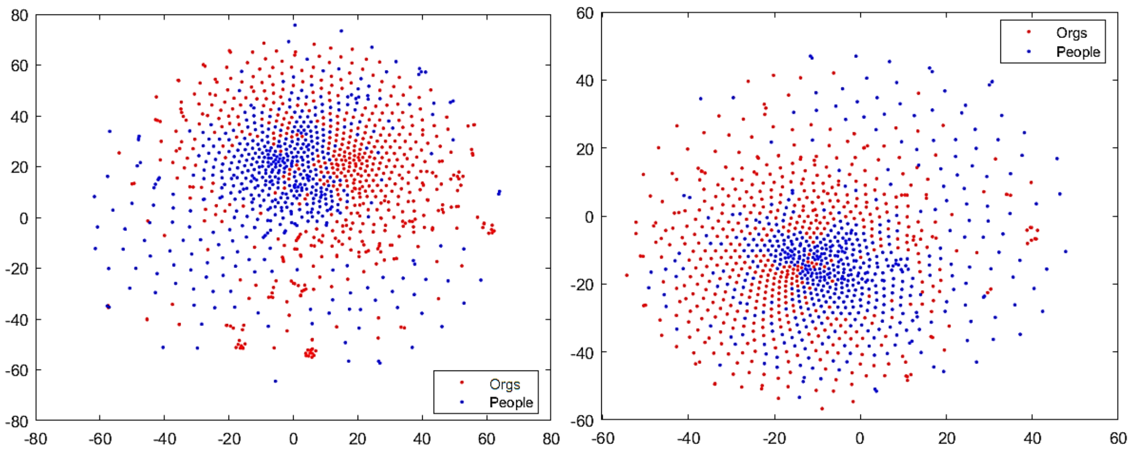


Figure 5.1: The plot of the *Orgs vs People* dataset. One the left hand is the training set, and on the right, the test set is shown. The dimension is reduced with t-SNE to two dimensions. The KLD to the original Dataset is 2.233 and 2.464

5.1.2 Image Dataset

The image dataset has two main Parts. The first one which is named *office* in this thesis and is collected and created from [?]. This contains various images from three different sources which are the different datasets respectively. One are images taken from the web downloaded from the online merchants *amazon* (Name in Dataset). Originally 31 categories, which are different types of products, 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 recognise the object. [?]
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 within five different objects are taken. An object contains on average three images, taken from various viewpoints. This makes a total of 423 images.
The image dataset has two main Parts. The first one which is named *office* in this thesis and is collected and created from [?]. This contains various images from three different sources which are the different datasets respectively. One are images taken from the web downloaded from the online merchants *amazon* (Name in Dataset).

Originally 31 categories, which are different types of products, 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 recognise the object. [?]
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 within five different objects are taken. An object contains on average three images, taken from various viewpoints. This makes a total of 423 images. The last set of images are from a *webcam*. Images are taken from this, with a low resolution with a 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 [?]. In comparison with the other image collections, the *Caltech-256* dataset is much larger. It has an average of 119 images per category in 257 categories with a total of 30607 images. They are crawled from Google and PicSearch. Duplicates are deleted. The Scale Invariant Feature Transform (SIFT) approach transforms images into scale-invariant coordinates relative to local features which are 'summarised' in key point descriptors.[?] 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 this 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.[?, 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 are ten similar categories 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 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.[?]
The SURF Algorithm uses integral images for images convolutions and uses existing detectors and descriptors and reduced the used methods to the essentials. [?] In figure 5.2 are some example images taken from the four datasets of the category computer monitor. As in 5.1.1, the 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. The last set of

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

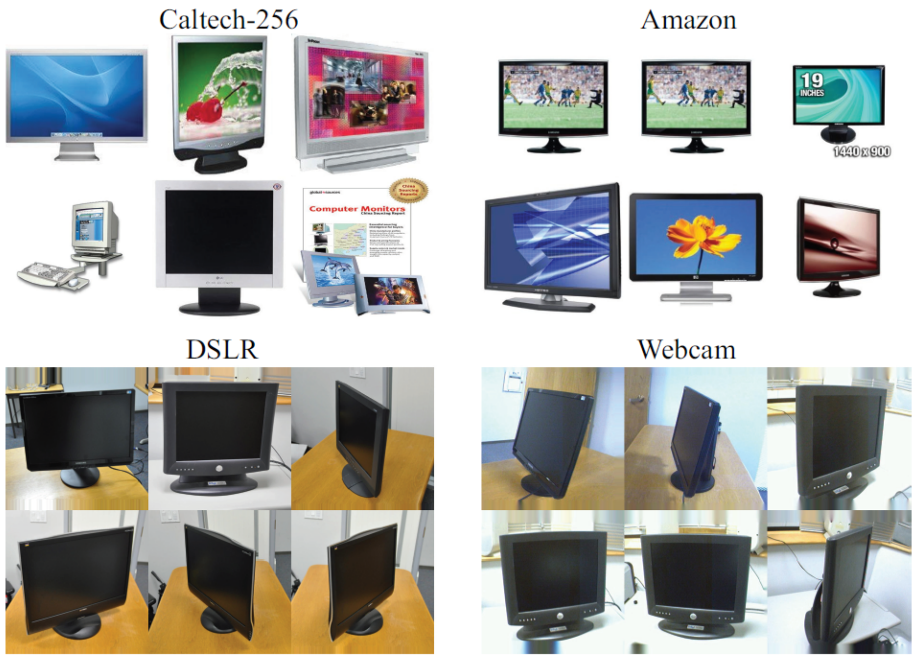


Figure 5.2: Examples Images from the four Image Datasets Caltech10, Amazon, DSLR, Webcam. [?]

images are from a *webcam*. Images are taken from this, with a low resolution with a 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 [?]. In comparison with the other image collections, the *Caltech-256* dataset is much larger. It has an average of 119 images per category in 257 categories with a total of 30607 images.

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 from the PCTKVM, 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. The PCVM is described in section2.

5.2.1 Support Vector Machine

The SVM is a popular solution for the classification or regression task. The latter and the following description is from [?, p. 325-337] A key property of the SVM is that the model parameters are determined with a convex optimisation problem. With this, any local solution is also global. In the optimisation problem, the Lagrange multipliers are used. The SVM is a maximum margin classifier with the goal to determine a margin to separate two classes of samples which should be maximal. Therefore the margin is the smallest distance between the decision boundary and any sample. The classifier in his simplest form and besides the choice of the kernel has two model parameters $\mathbf{w}$ and b. The first one is modelling 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. If a sample has a minimal distance to the hyperplane, then it is considered as support vector. The support vectors are on both sides of the hyperplane for the two classes. The prediction is made by determining the side of a data point corresponding to the decision boundary. If a new sample point is one side of the hyperplane, the point belongs to the corresponding class and vice versa. To be more formal:

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b = \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}) + b \tag{5.1}$$

Where $\mathbf{\Phi}(\mathbf{x}) = (\phi(\mathbf{x})_1, \dots, \phi(\mathbf{x}_N))$ a vector 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 $y(\mathbf{x}) > 0$ it belongs to class 1. Furthermore, when it comes to $y(\mathbf{x}) < 0)$ it belongs to class -1. Note that if a point in the trainings state has the function value of $y(\mathbf{x}) = 1 \vee y(\mathbf{x}) = -1$, then it is considered as support vector. The SVM has a complexity of $\mathcal{O}(M^3)$, with M basis functions, which is smaller than the actual N data points. To get a visual idea of how the SVM works the process is summarised in figure 5.3.

Note that this is a very brief explanation of the SVM, which takes some assumptions. Beside some concepts, for example, errors, are not defined. This is just to get an idea of how it works. The SVM is the classifier of every transfer learning method which can be treated as wrapper algorithms, except PCVM, PCTKVM, PCTKVM$_{\theta\text{Est}}$ and GFK. The latter is using an own implementation of k nearest neighbour. 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 there university

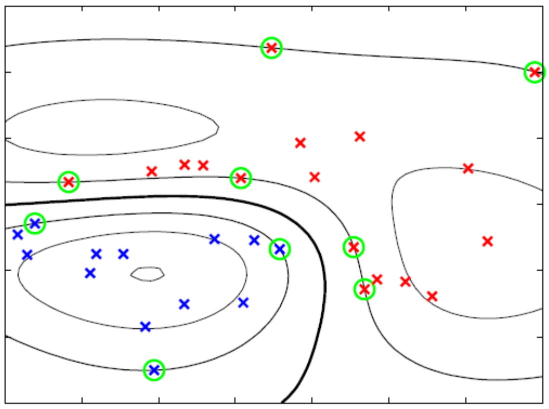


Figure 5.3: Example of the SVM trained with the synthetic dataset with two classes in two dimensions. The green surrounded data points are the support vectors. [?, p. 331]

page[3].

5.2.2 Data Sampling

The samples for testing the classifiers are drawn with 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 draw from one only for training or testing. This means for example, two image sets *amazon* and *webcam* are not merged together. Furthermore, the comparability of this study with another study would be lost. The training set is sampled from *amazon* and the test is from *webcam*. This is a suggested standard method for cross-validation in transfer learning.[?] Besides, the datasets are normalised via z-scored. This can be obtained from equation 4.21 in section 4.3.1.

5.2.3 Kernel

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

$$\mathbf{K}(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') \tag{5.2}$$

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 refereed as basis- instead of feature mapping functions.[?, p. 291-296, 329]

[3]https://www.csie.ntu.edu.tw/ cjlin/libsvm/

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 [?, p. 17] and has the form:

$$k(\mathbf{x}, \mathbf{x}') = \frac{d(\mathbf{x}, \mathbf{x}')^2}{2\sigma^2} \tag{5.3}$$

Where d is the Euclidean distance and sigma is the width of the Gaussian kernel. The used one is suggested from [?] and 'replaces' the σ parameter with $\frac{1}{A}$ and A as median over the samples:

$$k(x, x') = \exp(-\frac{1}{A} \left\|x - x'\right\|^2) \tag{5.4}$$

Note $\gamma = \frac{1}{A}$ is a used abbreviation in [?]. There are two classifiers with different kernels. First, the GFK uses the implementation of an own supervised k nearest neighbour approach which can be obtained from the website of the author 3.5.2. Second, the PCVM, because in the cross-validation state for the parameter determination the performance drops dramatically with the use of (5.4). Therefore the PCVM uses the proposed RBF-Kernel from [?].

$$k(x, x') = \exp(-\frac{\left\|x - x'\right\|^2}{\sigma^2}) \tag{5.5}$$

The rest is trained with the suggested one from (5.4), although they are used to train with (5.4), 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.

5.2.4 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, is set to 10. [?, p. 421-422] Finally, the LibSVM implementation which can work with pre-calculated kernels is used. The kernel is described above in section5.2.3.

The PCVM as baseline Classifiers has only one parameter θ, 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 ξ which is set to two for the text datasets and 1.1 for the image datasets.[?] Furthermore, the PCTKVM uses the new kernel with $\gamma = \frac{\sigma}{A}$ in (5.4). Because of that, there is another parameter σ which is set to two for the text sets and one for the image datasets. When it comes to the sigma estimation for PCTKVM$_{\theta\mathrm{Est}}$ described in section 4.3.1 the σ value is determined for

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 of the Text and Image Datasets

every dataset respectively. The estimated sigmas 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 regularisation parameter λ which is set to one for both.[**?**]
For the GFK solution the parameter, number of subspace dimensions are evaluated with cross-validation from $k = \{1, 5, 10, 20, ..., 100, 200\}$ 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 $\mu = \{1, 5, 10, 20, ..., 100, 200\}$ and is set to $\mu = 50$ for both datasets. 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.[**?**, p. 216-217] First, the mean which gives the average value of samples.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5.6}$$

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.

$$s^2 = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})} \tag{5.7}$$

In both (5.6) and (5.7), 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.[?] To determine if the null hypothesis can be rejected, 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 are four key figures used. For clarity, the key figures are represented in the result table in percent. This differs from the origin calculation presented here. The selection of the key figures are influenced by [?].

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.[?, p. 13]
A Receiver Operating Characteristics (ROC) Curve is a technique for visualising and organising classifiers based on their performance. This curve sometimes also called 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.[?, p. 4]
In this thesis, when it comes to two classes, it 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."[?, 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. For this thesis the build-in Matlab function *[X,Y, ,AUC] = 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:[?]

$$Accuracy = \frac{|x : x \in \mathcal{X} \land f(x) = y(x)|}{|x : x \in \mathcal{X}|} \tag{5.8}$$

With $f(x)$ as the given label from the classifier and $y(x)$ as the ground truth label. In other words, 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 normalised.[?, 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.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n} e_i^2} \tag{5.9}$$

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 penalises 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. [?]

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.[?]
With that, the different key figures over the datasets are summarised. The performance, in general, is divided into datasets. The full result can be considered in appendix B.

5.4.1 Metric Results

In this subsection, the performance over the metrics, which are descried earlier, is shown. The results are summarized in table 5.4. Note that this is 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 in the two dataset categories 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 classifier, which is created in the course of this work are italics. The performance of the best classifier is bold. The GFK implementation supports no probabilistic output and therefore has no AUC value in general. Not that through the cross-validation of the 18 datasets, the means metrics are combined from 180 performance results.
It can be observed that the TKL with the SVM 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 solutions. But we can note that the PCTKVM *overall outperforms* the PCVM in this test.
The SVM is still better in the image category than the transfer PCVM's. This might comes from the fact that the latter has some datasets, where the performance drops dramatically. Referring to table B.1 in appendix B, we can observe that if the dataset *dslr* and *webcam* are involved, then the performance drop happens. We assume that this happens because the datasets are very small, with 295 and 157 samples respectively, because we can rather obtain a good performance on the larger text datasets and usually the larger image datasets in comparison with the SVM. This assumption may be more valid when one takes a look at the comparison, which involves the whole dataset, which is shown in table B.5. Therefore 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.
Additional, 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 that because of the differences in the distribution caused by the dataset; it might be a better idea for transfer learning to find a hyperplane, which separates the classes best, rather than modelling it via probabilistic. In fact, the performance is of the TKL is always better than the performance of PCTKVM.
Furthermore we can observe that the PCTKVM$_{\theta \mathrm{Est}}$ is in general worse than the PCTKVM. Because these two approaches only differ in the θ value, we can say that the θ estimation approach described in section 4.3.1, works out not very well. In fact, the estimation was originally designed for regression and classification [**?**]. However, it seems that the value of the estimation is too 'smooth' and the prob-

ERR	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Reuters	31.82(2.02)	34.19(3.65)	*30.03*(6.63)	*26.12*(2.97)	30.87(3.24)	32.95(2.54)	34.53(2.74)	**22.70**(2.17)
Image	61.51 (2.75)	68.13(3.11)	*63.53* (3.48)	*62.64* (3.35)	59.56(3.33)	59.39(3.32)	64.24(2.96)	**58.25**(7.63)
RMSE	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Reuters	31.88 (8.43)	34.48(5.93)	*30.77* (10.35)	*26.28* (9.32)	31.03(9.00)	33.05(8.81)	34.63(6.28)	**22.81**(7.63)
Image	61.59(14.71)	68.23(6.88)	*63.64* (9.15)	*62.76*(9.35)	59.67(14.76)	59.50(14.81)	64.33(15.39)	**58.36**(13.63)
AUC	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Reuters	69.00 (2.33)	70.96(3.99)	*76.47* (8.63)	*79.36*(4.82)	72.32(3.78)	69.11 (3.13)	-	**83.46** (2.26)

Table 5.4: The results of the transfer learning classifiers with cross-validation under the four metrics on the 18 datasets.

abilistic 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 [?], 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 with respect to 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.1. As pointed out in the results of Chen et al. in [?] and with our results again, the range of standard deviation for 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 random initialised is not critical because we can observe that the standard deviation are not greatly higher in comparison with the other classifiers form table B.1.

5.4.2 Time Results

The next comparison is over the mean computational time, which a classifier needed to train and test the datasets. 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. Note that that the computational time is summarised in ranks. The first rank is assigned for the fastest and the last for the slowest. Average ranks are assigned for ties with respect to ranks. With this, there are the means of the datasets combined, and the ranks are the result of it. The table 5.5 shows the mean of these ranks. The second row of the table shows the mean time over all 18 datasets in seconds.
The result of the time ranking is unambiguous, with the SVM as the fastest classifier.

However, this is no surprise, because it is beside the PCVM the baseline classifier, and has because of this the lowest computational complexity, which results in lower training and testing time. Obviously, because the other solution are trying to do the transfer learning, which results in higher complexity.
Although the complexity of PCVM and SVM are comparable, in practice, more precisely, datasets with two domains, we observed that the θ optimisation takes forever. It seems that in the transfer learning setting, the local optimum for the next theta is hard to obtain. Another point to mention is that the PCVM does not converge very well and uses the full 500 iterations. This, combined with the θ optimisation in every iteration, leads to the high computational time.
Although the ranks from PCTKVM$_{\theta\text{Est}}$ and PCTKVM are bad in comparison with the other classifiers, we have, by integration of the new kernel, greatly improved the speed. This has two reasons: The first is that the θ optimisation is replaced with a fixed value or the estimation. The second is that the new PCVM's are converge much faster and therefore may need fewer iterations to finish. The theta estimation in comparison with the fixed value makes a mean difference from 11.52 seconds.
The computer which is used to determining the needed time was an Acer Aspire V5-573G, containing an Intel Core i5 4200U Prozessor 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$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
Rank	**1.08**	8.00	*4.46*	*5.46*	4.10	5.23	4.96	2.71
Sec	**3.99**	353.75	*17.56*	*17.19*	5.88	6.70	12.52	4.40

Table 5.5: The mean computational time rank of the classifiers.

5.4.3 Number of Support Vectors

The number of support vectors are an indicator of 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 in general just for simplicity.
May 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 generalisation abilities of the model.[**?**, p. 81]
Another explanation for a large number of support vectors is that the dataset is complicated. For example, the dataset tends to be inseparable.[**?**, 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 as 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.[**?**, p. 349]
This is the strength of the PCTKVM$_{\theta\text{Est}}$ and PCTKVM, although the sparse model of the PCVM has a worse performance, our solutions are obtaining relative sparse model by maintaining a competitive performance on difficult datasets. The mean number of support vectors needed by a classifier is split up dataset categories and is summarised in table 5.6. The mean number of the datasets are shown on the left-hand side of the name in brackets.

N. SV.	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	TKL
Reuters(1153.66)	482.35	46.93	**3.02**	**3.02**	182.70	220.28	190.73
Image(633.25)	309.90	68.40	**50.02**	*54.88*	253.30	287.433	382.53

Table 5.6: The mean of number support vectors of a classifier for Reuters and Image datasets.

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. [**?**] The first approach, the 5x2 t-test, was designed by Dietterich in 1998. [**?**] The improved 5x2 cv F test should have less Type I error (wrongly reject null hypothesis) and a higher power (probability of detecting a difference if it exists) than the proper.[**?**] It is in general designed to compare two classifiers. As mentioned in 5.2.2 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 $i = 1, ..5$. 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:

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} (p_i^{(i)})^2}{2 \sum_{i=1}^{5} s_i^2} \tag{5.10}$$

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 [?, p. 338-340] Critical values from this distribution can be obtained from a statistic table, e. g. [?, 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).[?]

5.5.1 Result and discussion

The result of the 5x2 cv F test is shown in table 5.7. In this table the cells are not showing the value of error in percent, but rather the summary of the wins, loses and ties against the PCTKVM as competitor algorithm. The row Significant is showing the amount of the significant wins loses or ties of the total. Note that because of this setup, one has to read the table the other way around. A lose for a classifier shown in this table is a win for the PCTKVM and vice versa. A win is determined by comparing the performance of the two folds (10 runs). If there are more wins out of the 10 for one classifier, then it is a win for it. If the wins and losses are equal, then is a tie. That means a tie does not represent a real equal performance, but rather that the wins and losses are even.
With the descriptive statistic table from 5.4, we can not make a valid statistical evidence if the difference of performance is significant, i. g. we can reject the null hypothesis.[?, p. 9] We can only observe trends and assume significance.
The results are showing, that the PCTKVM does a decent job in the Reuters dataset. In fact, it wins every time against the baseliners and the transfer learning solutions, except against the TKL, partly even significant.
The PCTKVM can outperform the baseline PCVM four times significant out of 6 wins with the text dataset. The performance gap is not that large at images, but sill our approach wins every time out of twelve datasets with one significant win.
Compared to the PCTKVM$_{\theta\text{Est}}$, it wins five out of 6 times with one significant and one tie. 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 θ for the image is not well selected, but it is selected with a cross-validation process to determine the best performance. Another reason could be that the θ estimation does a better job when it comes to images or takes values that are not considered in the parameter determination.
Considering the performance with the image dataset with respect to the remaining classifiers, the baseline SVM can win six times against the PCTKVM, with one significant, while the latter only wins four times. The same 'worse' result happens with the transfer learning classifiers, with a similar result. As already discussed in section 5.4.1, a reason for this might be the small datasets. Note that this statistic

Win-Lose-Tie	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	TCA	JDA	GFK	TKL
Reuters	0-6-0	0-6-0	*4-1-1*	0-6-0	0-6-0	0-6-0	6-0-0
Significant	0-3-0	0-4-0	*0-0-0*	0-2-0	0-3-0	0-4-0	0-0-0
Image	6-4-2	0-12-0	*4-3-5*	9-3-0	6-3-3	6-6-0	10-1-1
Significant	1-0-0	0-1-0	*1-0-0*	2-1-0	3-0-0	2-0-0	4-0-0

Table 5.7: 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.

has ten and five degrees of freedom.

5.6 Comparison over multiple Datasets

Testing algorithm over only one dataset has a certain risk. The problem is with a high amount of tests like it is done in 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 is used as a statistic to compare supervised classification learning algorithms over multiple Datasets. It is first introduced, as the name says, by Friedman in 1937. It is a non-parametric equivalent of the repeated-measures of Analysis of Variance (ANOVA).[**?**, 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 is significantly larger than the error variability, the null hypothesis can be rejected with the conclusion that there are differences between the classifier. 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.[**?**, p. 10]
Sphericity is the condition that the difference between samples is homogenate. If not the ANOVA F value will be too large.[**?**, 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. For r_i^j as the j-th rank of k algorithm on the i-th of N datasets. And 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 Rj should be equal. In equation (5.11) the formula for the F value is shown.

$$\mathcal{X}_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{5.11}$$

It is distributed according to X_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 $k > 5$. [?, p. 11]
We are about the 'solid' numbers with eight classifiers and 15 datasets. However, Iman and Davenport showed in [?] that the Friedman $\mathcal{X}_F^2$ value is needless conservative and proposed a better statistic which is shown in equation (5.12).[?, p. 11]

$$F_f = \frac{(N-1)\mathcal{X}_F^2}{N(k-1)-\mathcal{X}_F^2} \tag{5.12}$$

This is statistic is F distributed with $k-1$ and $(k-1)(N-1)$ degrees of freedom. The Friedman test only determines is there any significant difference at all. If the null hypothesis can be rejected (Significant difference), we can proceed with the post-hoc test to determine which classifiers are significantly different. Significance exists, if the F value from (5.12) is equal or above the critical values, for a confidence level of 0.95, from the F distribution.[?, p.11]
In this article, the Bonferroni-Dunn test is used for the post-hoc test as it is used in [?]. 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_α and is set to 2.690, because of eight classifiers and the confidence level of $\alpha = 0.5$. So the confidence levels of Friedman and 5x2 cv F test are the same. The critical difference is shown in equation (5.13).[?, p. 11-12]

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{5.13}$$

Note that the Friedman test and the post-hoc test is implemented on our own, because it seems that Matlab[4] provides only the first part of the test (Original Friedman), which is the Friedman test without the described improvements from Iman and Davenport.

[4]https://de.mathworks.com/help/stats/friedman.html

5.6.3 Result and Discussion

The in the following discussed results are covering the 18 datasets. The result of the Friedman test is separated into two tables. The first 5.8 is showing the mean ranks under the performance metrics. This is done by ranking the mean of the performance according to one dataset and results in 18 ranks per classifier. A tie is assigned if the mean of two classifiers are equal.
The second 5.9 shows the Friedman test result and the corresponding post hoc test. It shows the results of Friedman test with the corresponding post hoc test. The critical difference is computed according to equation (5.13). The 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 the 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, RMSE a classifier is better if the rank difference is negative and worse if it is positive. For AUC and ACC it is vice versa. The bold cells are indicating any significance. Note that one may draw further results out of the results by just taking the mean ranks of two classifiers from table 5.8, subtract them and compare the difference against the critical difference.
Although we have omitted some metrics in the previous discussion, it seems reasonable to show all four metrics, because they differ more in this test in comparison with the previous statistics. Finally we can provide a 7×119 degrees of freeman for error, RMSE and accuracy and 6×102 degrees of freedom for AUC.
With the results of table 5.9, we can say that the PCTKVM can outperform the PCVM significant in three out of four metric categories. The same is valid when it comes to GFK.
In comparison with the PCTKVM$_{\theta\text{Est}}$, we can say that the PCTKVM is in general better but not significant. In fact the SVM is a little bit better than the PCTKVM$_{\theta\text{Est}}$ and always better as the PCVM in the overall comparison.
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 GFK are 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.[?][?][?]
Another important fact which arises from the result is that the SVM by incorporating the kernel from section 5.2.3, very competitive to the transfer learning solutions. Note that we do not say, that the kernel causes the good performance, but we want

Mean Ranks	SVM	PCVM	PCTKVM{θEst}	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.

Metrics	Friedman test	$CD_{0.05}$	SVM	PCVM	$PCTKVM_{\theta Est}$	TCA	JDA	GFK	TKL
ERR	0.00	2.20	0.72	**3.16**	*1.16*	-0.39	0.33	**2.78**	-2
RMSE	0.00	2.20	0.44	**3.05**	*1.05*	-0.5	0.33	**2.61**	-2.11
AUC	0.01	2.20	-1.55	-2	*-0.61*	-1.72	-0.94	-	0.23
ACC	0.00	2.20	-0.72	**-3.16**	*-1.16*	0.39	-0.33	**-2.78**	2

Table 5.9: The result of the Friedman Test with Bonferroni-Dunn as Post-Hoc Test to compare classifiers over multiple Datasets.

to point out the difference to the suggested setting of the LibSVM[5]. Only the TKL can significantly outperform the SVM. Furthermore, the differences of $\text{PCTKVM}_{\theta\text{Est}}$ and the TKL are actual significant.

5.7 Conclusion

Summarising, the study is made of the three main parts of the statistic. 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 analytical 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 corresponding kind of dataset. With this listing of the 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 viewpoint, the comparison over all used datasets and the results in performance giving the most useful insights.
Summarizing, we proofed the success of the integration of transfer learning into the PCVM by showing significant differences in the performance between these two. Fur-

[5]http://www.csie.ntu.edu.tw/˜cjlin/libsvm/faq.htm#f506

thermore, our approach can compete with current transfer learning solutions and is just significantly outperformed by one method.
We proofed it with an extensive study over 18 dataset compositions of two data categories, which are common in testing of transfer learning, and by finally incorporating two statistical procedures to determine significance.

Chapter 6

Evaluation and Future Work

This chapter is devoted to critical review the characteristics of the study and future work.
The study classifier used in this thesis are primarily based on the kernel from (5.4). According to the corresponding articles, they using the standard Gaussian kernel from (5.3).[?][?][?] 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. Additional, 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 comparable 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 more datasets, to give a more general evidence, whether a transfer learning solution is better or not. For example the 20-Newsgroup[1] dataset, as a text based dataset, as suggested in [?]. 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 a 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, and it would be better to include more transfer learning solutions than datasets.
Other examples for image transfer learning datasets *USPS*[2], *MNIST*[3], *COIL20*[4].[?][?] These are not included in the study. Again for a more general assumption on the per-

[1]http://qwone.com/ jason/20Newsgroups/
[2]http://www-i6.informatik.rwth-aachen.de/ keysers/usps.html
[3]http://yann.lecun.com/exdb/mnist/
[4]http://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php

formance, it would be useful. As well as real-world scenarios like WiFi-localisation or human activity classification.[?, p. 32] At least we tested our transfer learning solution with the myoelectric prosthesis as real-world scenario.
Furthermore, as described in 5.4.1, a more analytical 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' of it. A possible approach would be to incorporate another estimation for finding the theta between the separated classes in the training set and combining them with the overall theta. 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, the biggest problem is, that the performance drops dramatically for certain datasets. For the image datasets, we assume that the reason for this is only a few training examples per domain. However, we can also observe a drop of performance by the *people vs places* dataset, which has more than 1000 samples. In future work, the reason for the dips could be determined.
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. 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 a 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 [?].

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 feature of this PCVM is to deal with the uncertainty or latent state of 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 behaviour, the potential and limits of transfer learning methods We gained knowledge of the efficiency of the 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 Classifcation Transfer Kernel Vector Machine which is comparable to many currently available transfer learning solutions.
Summarizing, in general transfer learning aims to improve the 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 great amount of transfer learning methods.[**?**, p. 33] All of them are solving the problem in their way with an individual set of requirements, which results naturally in different performances, concerning tasks or domains in general. Based on different sub problems in transfer learning, the current state is that domain adaptations will lower the differences in marginal distributions. Furthermore, to address the problem of different conditional probability distributions the pseudo labelling technique will be applied. When it comes to different

tasks, an inductive approach with a small amount of labelled target data is also 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 in order 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.[**?**] This challenge can be addressed with transfer learning to make machine learning in general more practical for real-world problems. Especially when it the lab conditions are left behind. Our example real-world problem is the re-calibration of a myoelectric prosthesis, which classifies motion sequences in movements. This problem can be addressed with transfer learning to improve quality of the recognition rate, which can finally improve the quality of the patient's life. The range of type from real-world problems is very broad. Some examples are sentiment classification from very different types of text document, difficulty image classification, the detection of defect prone software.[**?**] However, more important the task of human activity recognition, cancer perdition or other healthcare related tasks.[**?**][**?**]
The incorporating of machine learning in this research sections, not only our prosthesis case, has the potential to improve the health or life of patients greatly. 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 notations are explained. This notation is used through out the whole thesis. In particular the matrix notations are not consistent defined, which is observable for example in the following literature: [**?**], [**?**] and [**?**]. First 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 simple $\mathbf{w}$ and $(w_1 \dots w_N)^T$ is a column vector two. 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 $N \times M$. In the context of pattern recognition, the rows and columns are the numbers of observations or data points and dimensions. The M $\times$ M identity matrix is denoted as $\mathbf{I}_{\mathrm{M}}$. If the number of dimensionalities is obvious the matrix is simply referred as $\mathbf{I}$. The elements of the identity matrix $\mathbf{I}_{\mathrm{i,j}}$ 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 it is obvious which variable has been used, the subscript is for simplicity omitted, e.g. $E[x]$. If the distribution of X is conditional on another variable Y, then conditional expectation will be written as $E_x(f(x) \mid Y)$. In other words, the conditional expectations of X given Y.
The variance of 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 [**?**, 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 $P(y \mid x)$. If g is a

distribution of a random variable X, then $p_i = g(x_i)$ is the proability that $X = x_i$. If a probability distribution f follows the normal distribution N, then this is denoted 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 ≥ 0. With this, the value of the i-th row and j-th column is addressed.

A.0.1 Terminology

In this thesis may various definitions of the Gaussian distribution 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 centred at the origin. Furthermore, the zero-mean Gaussian distribution has his mean also in origin with $N(0, \sigma^2)$, with σ^2 as variance. If $\sigma^2 = 1$, then it is the Standard Gaussian Distribution.
The terms source and training or target and testing, referring to feature spaces, distribution or labels are synonyms. A similar case is when it comes to labels. A class refers to a portion of labels, which represents them. When it comes to multi class problems, if it is obvious, then we may also refer 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 the chapter 5 for clarity. The detailed metrics for the single datasets can be looked up her. Furthermore in this chapter are test results which are not discussed 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 [?] 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 table is showing the mean over the 5x2 cv F test, which is a total of ten repetitions. Every repetition has his unique sampling from the dataset. Note that in table B.1 shows not only the mean but the standard deviation in brackets. Furthermore in table B.2 the AUC values are shown. The mean number of support vectors used can be viewed in table B.3. The last table B.4 shows the needed time in seconds.

Error Image	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	PCTKVM	TCA	JDA	GFK	TKL
C vs A	**52.38** (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)	**56.69** (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)	**63.52**(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)	**58.70**(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)	**60.90** (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)	**64.29** (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)	**70.94** (2.52)
D vs A	73.63 (1.61)	77.06 (2.52)	72.44 (2.07)	74.03 (2.47)	**68.93** (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)	**32.47**(4.55)
W vs C	73.86 (1.52)	76.49 (1.76)	73.70 (2.1)	74.14 (2.79)	**68.44** (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)	**66.28** (1.53)	72.13 (1.55)	71.27 (2.24)	68.08 (1.45)
W vs D	**25.10** (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)	**10.65**
Error Reuters	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	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)	**19.29** (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)	**12.76** (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)	**22.84** (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)	**18.33** (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)	**29.55** (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)	**33.42** (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)	**22.81**(7.63)

Table B.1: Cross-validation comparison of the tested algorithms on 18 domain adaptation datasets by the error and RMSE metrics. It shows the mean of the ten runs of cross-validation per dataset with the standard deviation.

AUC Image	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
AUC Reuters	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.2: Cross-validation comparison of the tested algorithms on 18 domain adaptation datasets by the AUC value. It shows the mean of the ten runs of cross-validation per dataset. The positive class is one, and the negative class are the remaining classes.

N. SV. Image	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
N. SV. Reuters	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.3: 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.

Time Image	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	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
Time Reuters	SVM	PCVM	PCTKVM$_{\theta\text{Est}}$	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.4: 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.5 are the error and RMSE values shown. The AUC values are shown in table B.6. For a complete overview over the used support vectors, see table B.7. And finally the computation time in second as mean over the 5 runs is shown in B.8.

Error Image	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
Error Reuters	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.5: The test result of the classifiers by using the whole 18 datasets. It shows the mean error over five runs.

AUC Image	SVM	PCVM	PCTKVM$_{\theta \mathrm{Est}}$	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
AUC Reuters	SVM	PCVM	PCTKVM$_{\theta \mathrm{Est}}$	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.6: 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 are the remaining classes.

N. SV. Image	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
N. SV. Reuters	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.7: Comparison of the number of used support vectors for the whole 18 datasets. It shows the mean number over five runs.

Time Image	SVM	PCVM	PCTKVM$_{\theta \mathrm{Est}}$	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
Time Reuters	SVM	PCVM	PCTKVM$_{\theta \mathrm{Est}}$	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.8: 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 to Algorithms

In this appendix, the detailed steps for the PCTKVM algorithm are presented. We omit it the steps, which are already described in PCVM and TKL for clarity. The complete algorithm procedure 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 the θ or proceed with the estimation. If one decides to choose a θ on his own, then the model has two parameters: The width of the Gaussian kernel and the ζ eigenspectrum dumping factor. Moreover, PCTKVM algorithm has following algorithm parameter: The PCTKVM supports the standard Gaussian kernel, the new Gaussian kernel from (5.4) and the Laplacian kernel. But we strongly recommend the use of the new Gaussian kernel.

Furthermore the number of iterations *niter* for the PCVM can be specified and the convergence criteria as *threshold*.

Algorithm 4 Probabilistic Classification Transfer Kernel Vector Machine

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

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

```
 1: $\mathbf{D}$ = calculate_Dissimilarity_Matrix($\mathbf{K}$);
 2: if $\theta == -1$ then
 3:     $\theta$ = theta_Estimation($\mathbf{D}$);        ▷ According to section (4.3.1)
 4: end if
 5: [$\mathbf{K}_{\mathcal{Z}}$, $\mathbf{K}_{\mathcal{X}}$,$\mathbf{K}_{\mathcal{Z}\mathcal{X}}$]=compute_Gaussian_Kernel($\mathbf{D}$,$ker$,$\theta$),
 6: [$\{\mathbf{\Lambda}_{\mathcal{X}}, \mathbf{U}_{\mathcal{X}}\}$]= eigenDecompose($\mathbf{K}_{\mathcal{X}}$)        ▷ Eq. (4.5)
 7: [$\bar{\mathbf{U}}_{\mathcal{Z}}$] =extrapolate_Eigensystem($\mathbf{K}_{\mathcal{Z}\mathcal{X}}, \mathbf{U}_{\mathcal{X}}, \mathbf{\Lambda}_{\mathcal{X}}^{-1}$)        ▷ Eq. (4.13)
 8: [$\mathbf{\Lambda}$]=quadratic_Programming($\mathbf{Q}, \mathbf{C}, r$)        ▷ Eq. (4.16) and constrains (4.17)
 9: [$\bar{\mathbf{K}}_{\mathcal{A}}$]=compute_Extrapolated_Kernel($\bar{\mathbf{U}}_{\mathcal{Z}}, \mathbf{\Lambda}, \mathbf{U}_{\mathcal{X}}$)        ▷ Eq. (4.18)
10: [$\mathbf{w}, b$] = initialize(initVector);
11: nonZero = determine_nonZero_Vector($\mathbf{w}$);
12: for $i = 1$ to $niter$ do
13:     $\mathbf{w}^{new}$ = weight_Update($\bar{\mathbf{K}}_{\mathcal{Z}}, \mathbf{w}, \mathbf{Y}$, nonZero);        ▷ Eq. (2.17)
14:     $b^{new}$ = bias_update($\bar{\mathbf{K}}_{\mathcal{Z}}, b, \mathbf{Y}$, nonZero);        ▷ Eq. (2.18)
15:     $\theta^{new}$ = parameter_update($\bar{\mathbf{K}}_{\mathcal{Z}}, \mathbf{Y}, ker, \theta, \mathbf{w}^{new}, b^{new}$, nonZero);  ▷ Eq. (2.19)
16:     nonZero$^{new}$ = determine_nonZero_Vector($\mathbf{w}^{new}$);
17:     if max(abs($\mathbf{w}^{new} - \mathbf{w}$)) < $threshold$ then
18:         break;
19:     else
20:         continue;
21:     end if
22: end for
```