Correcting Covariate Shift with the Frank-Wolfe Algorithm

Junfeng Wen and **Russell Greiner** and **Dale Schuurmans**
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{junfeng.wen, rgreiner, daes}@ualberta.ca

Abstract

Covariate shift is a fundamental problem for learning in non-stationary environments where the conditional distribution $p(y|x)$ is the same between training and test data while their marginal distributions $p_{\text{tr}}(x)$ and $p_{\text{te}}(x)$ are different. Although many covariate shift correction techniques remain effective for real world problems, most do not scale well in practice. In this paper, using inspiration from recent optimization techniques, we apply the Frank-Wolfe algorithm to two well-known covariate shift correction techniques, Kernel Mean Matching (KMM) and Kullback-Leibler Importance Estimation Procedure (KLIEP), and identify an important connection between kernel herding and KMM. Our complexity analysis shows the benefits of the Frank-Wolfe approach over projected gradient methods in solving KMM and KLIEP. An empirical study then demonstrates the effectiveness and efficiency of the Frank-Wolfe algorithm for correcting covariate shift in practice.

1 Introduction

Many machine learning algorithms assume that training and test data come from the same distribution, which is often violated in practical applications. Researchers have thus endeavoured to resolve the distribution shift problem under varied assumptions [Zadrozny, 2004; Bickel *et al.*, 2007; Quionero-Candela *et al.*, 2009]. In this work, we focus on the *covariate shift* scenario [Shimodaira, 2000] in which the marginal data distributions are different between training and test domains ($p_{\text{tr}}(x) \neq p_{\text{te}}(x)$) while their conditional distributions remain the same ($p_{\text{tr}}(y|x) = p_{\text{te}}(y|x)$). Correcting covariate shift has a wide range of applications, such as in natural language processing [Jiang and Zhai, 2007], off-policy reinforcement learning [Hachiya *et al.*, 2009], computer vision [Yamada *et al.*, 2012] and signal processing [Yamada *et al.*, 2010].

A common approach to correcting covariate shift is *importance reweighting*: each individual training point is assigned a positive weight intended to diminish the discrepancy between training and test marginals by some criterion [Sugiyama *et al.*, 2012]. If a reweighting function is modelled properly, covariate shift can be effectively corrected in learning. For example, for a given prediction function $f(x)$ and a loss function $l(f(x), y)$, reweighting training points with weight $w(x) = p_{\text{te}}(x)/p_{\text{tr}}(x)$ can minimize the loss over the test distribution:

$$\mathbb{E}_{x\sim p_{te}}\mathbb{E}_{y|x}[l(f(x), y)] = \mathbb{E}_{x\sim p_{tr}}\mathbb{E}_{y|x}[w(x)\, l(f(x), y)],$$

which suggests that one should seek the function $f^\star$ that minimizes this expected loss. While these correction techniques remain effective for many real world applications, most of them do not exploit the structure of the solution and as a result, do not scale well in practice.

Recently, the Frank-Wolfe (FW) algorithm has begun to gain popularity in the machine learning community [Zhang *et al.*, 2012; Bach, 2015]. It has been proven to be an efficient algorithm for many optimization problems, particularly when the solution has sparse structure [Jaggi, 2013], and has also been shown effective and efficient in many applications [Joulin *et al.*, 2014; Salamatian *et al.*, 2014]. However, whether Frank-Wolfe can be applied to the covariate shift problem has remained unexplored.

In this work, we show that herding [Welling, 2009; Chen *et al.*, 2010] (as a Frank-Wolfe algorithm) can be applied to the covariate shift scenario, and point out a connection between kernel herding and Kernel Mean Matching (KMM) [Gretton *et al.*, 2009]. Moreover, we exploit the structure of another commonly used covariate shift correction technique, Kullback-Leibler Importance Estimation Procedure (KLIEP) [Sugiyama *et al.*, 2008], to speed up the algorithm using Frank-Wolfe. We also analyse the convergence rate and complexity of KMM(FW) and KLIEP(FW), and present an empirical study that demonstrates the efficiency of Frank-Wolfe in solving KMM and KLIEP over the traditional projected gradient approach.

2 Related Work

This work is closely related to kernel herding [Chen *et al.*, 2010], which is a sampling technique for moment approximation. In the original setting, data points are iteratively generated to approximate the population mean (first moment), either from a population when the distribution p is known, or selected from existing dataset when p is unknown. This work is different in that we apply herding to the covariate

shift scenario, where the goal is to approximate the *test* mean by sub-sampling existing training data.

Even though there are many ways to correct covariate shift, few methods scale well, which hampers their usage in practice. Tsuboi *et al.* [2009] modifies KLIEP with a different parametric form of the weight function. Although the computation time of the new model is independent of the number of test points, the parametric form is less interpretable than the original KLIEP. Here we apply Frank-Wolfe to the original KLIEP, where the weight function is parametrized by a mixture of Gaussians. Another attempt has been made to solve KMM and KLIEP via online learning [Agarwal *et al.*, 2011]. However, the convergence rate of that approach, $O(1/\sqrt{t})$, is slower than our proposed method, $O(1/t)$, and assumes the availability of explicit feature representation of $\phi(\boldsymbol{x})$, contrary to the idea of Hilbert space estimation.

There are also several other techniques that correct covariate shift. For instance, RuLSIF [Yamada *et al.*, 2011] minimizes the α-relative Pearson divergence using least squares, while RCSA [Wen *et al.*, 2014] corrects covariate shift in an adversarial setting. We do not extensively investigate the possibility of applying Frank-Wolfe to these techniques, since Frank-Wolfe is efficient primarily when a convex problem has sparse solution structure and the corresponding linearised problem can be solve easily.

3 A Brief Review of Herding

Kernel herding [Chen *et al.*, 2010] is a greedy algorithm that sequentially generates instances, $\phi(\boldsymbol{x}_t)$ in the tth iteration, to minimize a squared error $\mathcal{E}_T^2$:

$$\mathcal{E}_T^2 = \left\| \frac{1}{T} \sum_{t=1}^{T} \phi(\boldsymbol{x}_t) - \mu_p \right\|_{\mathcal{H}}^2,$$

where $\phi(\cdot)$ is a feature map to a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ and $\mu_p = \mathbb{E}_{x\sim p}[\phi(\boldsymbol{x})]$ is the population mean. Algorithmically, it generates instances according to:

$$\begin{aligned} \boldsymbol{x}_{t+1} &\in \operatorname*{argmax}_{\boldsymbol{x}\in\mathcal{X}} \langle u_t, \phi(\boldsymbol{x})\rangle, \\ u_{t+1} &= u_t + \mu_p - \phi(\boldsymbol{x}_{t+1}), \end{aligned} \tag{1}$$

for some properly chosen u_0, where $\mathcal{X}$ is the sampling space. In the original kernel herding algorithm, the expectation is taken with respect to the training population, i.e., $p = p_{\text{tr}}$.

Bach *et al.* [2012] showed that herding is equivalent to a Frank-Wolfe algorithm (also known as conditional gradient descent) for the objective

$$\min_{\widehat{\mu}\in\mathcal{M}} \frac{1}{2} \|\widehat{\mu} - \mu_p\|_{\mathcal{H}}^2,$$

where $\mathcal{M}$ is the marginal polytope (the convex hull of all $\phi(x)$ for $x \in \mathcal{X}$). Specifically, it employs the following updates

$$\begin{aligned} s_{t+1} &\in \operatorname*{argmin}_{s\in\mathcal{M}} \langle \widehat{\mu}_t - \mu_p, s\rangle, \\ \widehat{\mu}_{t+1} &= (1-\rho_t)\widehat{\mu}_t + \rho_t s_{t+1}, \end{aligned} \tag{2}$$

where $\rho_t \in [0, 1]$ is step size. They pointed out that herding corresponds to using a step size $\rho_t = 1/(t+1)$, while other choices (e.g., via line search) are also valid. With $\rho_t = 1/(t+1)$, s_t can be seen as $\phi(\boldsymbol{x}_t)$ while $\widehat{\mu}_t = \mu - u_t/t$ as in (1).

4 Frank-Wolfe for Covariate Shift

4.1 Kernel Herding and KMM

In this work, we apply kernel herding to the covariate shift scenario, where the training and test marginal distributions are different ($p_{\text{tr}}(\boldsymbol{x}) \neq p_{\text{te}}(\boldsymbol{x})$) while their conditional distributions are the same ($p_{\text{tr}}(y|\boldsymbol{x}) = p_{\text{te}}(y|\boldsymbol{x})$). We use herding to generate instances from the training pool in order to approximate the *test* mean, i.e., where $\mathcal{X}$ is the training set but $p = p_{\text{te}}$ is the test marginal distribution. Intuitively, herding actively selects representative training points to approximate the test population mean. With proper substitution of empirical estimates, our objective is

$$\min_{\widehat{\mu}\in\widehat{\mathcal{M}}} \frac{1}{2} \left\| \widehat{\mu} - \frac{1}{m} \sum_{j=1}^{m} \phi(\boldsymbol{x}_j) \right\|_{\mathcal{H}}^2,$$

where $\widehat{\mathcal{M}}$ is marginal polytope of training dataset and m is the number of (unlabelled) test points. According to (2), in each update step, a representative training point $\phi(\boldsymbol{x}_t)$ will be chosen from $\widehat{\mathcal{M}}$ and $\widehat{\mu}_t$ will be moved toward that direction. This objective is very similar to that of Kernel Mean Matching (KMM) [Gretton *et al.*, 2009]: [1]

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathcal{W}} \quad & \left\| \sum_{i=1}^{n} w_i \phi(\boldsymbol{x}_i) - \frac{1}{m} \sum_{j=1}^{m} \phi(\boldsymbol{x}_j) \right\|_{\mathcal{H}}^2 \\ \mathcal{W} &= \left\{ \boldsymbol{w} \middle| \sum_{i=1}^{n} w_i = 1, w_i \geqslant 0 \right\} \subset \mathbb{R}^n, \end{aligned} \tag{3}$$

where n training points and m test points are given. Problem (3) (a linearly constrained quadratic program) is generally solved by standard solver with gradient descent and feasibility projection. However, according to the interpretation in (2), herding for KMM is projection-free: with a proper step size $\rho_t \in [0, 1]$, the estimated mean $\widehat{\mu}_t$ is a convex combination of previous chosen training points, and thus the constraint $\mathcal{W}$ in (3) is automatically satisfied.

4.2 Frank-Wolfe for KLIEP

Once herding is viewed as a Frank-Wolfe algorithm, it can be applied to other covariate shift correction procedures. For instance, the Kullback-Leibler Importance Estimation Procedure (KLIEP) [Sugiyama *et al.*, 2008], a popular choice for covariate shift correction, can be efficiently solved by Frank-Wolfe. The objective of KLIEP is to minimize the Kullback-Leibler divergence from $p_{\text{te}}(\boldsymbol{x})$ to $\widehat{p}_{\text{te}}(\boldsymbol{x})$, where $\widehat{p}_{\text{te}}(\boldsymbol{x})$ is attained by reweighting the training marginal:

$$\widehat{p}_{\text{te}}(\boldsymbol{x}) = \widehat{w}(\boldsymbol{x})\, p_{\text{tr}}(\boldsymbol{x}).$$

The reweighting function is further parametrized as a mixture of Gaussians:

$$\widehat{w}(\boldsymbol{x}) = \sum_{l=1}^{b} \alpha_l \varphi_l(\boldsymbol{x}) = \sum_{l=1}^{b} \alpha_l \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_l\|^2}{2\sigma^2} \right),$$

[1] Here the predefined bound on the weight ($w_i \leqslant B$) and the derivation tolerance ($|\sum_{i=1}^{n} w_i - 1| \leqslant \epsilon$) are ignored for the sake of clarity and simplicity. However, they can be easily incorporated into the herding scheme.

where α_l are the mixing coefficients, $\boldsymbol{c}_l$ are predefined centres (usually test points) and σ is a parameter selected by some criterion. After some derivation and substitution of empirical marginals, the final objective becomes

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & F(\boldsymbol{\alpha}) = \sum_{j=1}^{m} \log \left(\sum_{l=1}^{b} \alpha_l \varphi_l(\boldsymbol{x}_j) \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \sum_{l=1}^{b} \alpha_l \varphi_l(\boldsymbol{x}_i) = n;\ \alpha_1, \alpha_2, \cdots, \alpha_b \geqslant 0. \end{aligned} \tag{4}$$

This is a convex optimization problem. To apply Frank-Wolfe, we first derive the gradient of the objective

$$\frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha_l} = \sum_{j=1}^{m} \frac{\varphi_l(\boldsymbol{x}_j)}{\sum_{l'=1}^{b} \alpha_{l'} \varphi_{l'}(\boldsymbol{x}_j)}. \tag{5}$$

Considering the constraint, one can observe that the possible range for α_l is $[0, n/\sum_{i=1}^{n} \varphi_l(\boldsymbol{x}_i)]$. For the tth iteration, we choose l_t such that

$$l_t = \operatorname*{argmax}_l \left\langle \frac{\partial F(\boldsymbol{\alpha})}{\partial \alpha_l}, \frac{n}{\sum_{i=1}^{n} \varphi_l(\boldsymbol{x}_i)} \right\rangle,$$

then update the current $\boldsymbol{\alpha}_t$ by

$$\boldsymbol{\alpha}_{t+1} = (1 - \rho_t) \cdot \boldsymbol{\alpha}_t + \rho_t \sum_{i=1}^{n} \varphi_{l_t}(\boldsymbol{x}_i) \cdot \boldsymbol{e}_{l_t}, \tag{6}$$

where ρ_t is the step size and $\boldsymbol{e}_l = [0, \cdots, 0, 1, 0, \cdots, 0]^T$ (all zeros except for a one in the lth entry). Compared to KMM(FW), this method chooses representative Gaussians to match test marginal, instead of choosing individual points.

The chosen Gaussian centres (i.e., $\{\boldsymbol{c}_l\}$) are crucial for KLIEP. In the original paper, the $\{\boldsymbol{c}_l\}$ are set to be the test points $\{\boldsymbol{x}_j\}$, which can be computationally intensive when one has many test points. Therefore, for large test data, the original authors proposed to randomly select $b = 100$ test points as Gaussian centres. However, such an approach can significantly reduce the performance of the algorithm, since the 100 chosen centres might not appropriately model the distribution difference, a phenomenon we observe in our experiments (see Section 6.2). If Frank-Wolfe is applied, only one Gaussian is amplified per iteration and the optimal solution can be efficiently attained [Jaggi, 2013], since it tends to be sparse [Sugiyama *et al.*, 2008]. More importantly, we can use *all* the test points as Gaussian centres in order to utilize their information to capture the distribution difference.

5 Convergence and Complexity

5.1 Convergence Rate

In this section, we compare the convergence and complexity of projected gradient (PG) and Frank-Wolfe methods. For convergence, since both KMM and KLIEP are convex problems, it is well known that the convergence rate of PG is $O(1/t)$ [Bertsekas, 1999]. The convergence rate of Frank-Wolfe for KMM and KLIEP is also $O(1/t)$ without further assumptions [Jaggi, 2013]. However, under some circumstances, $O(1/t^2)$ and even $O(e^{-t})$ are possible for KMM(FW) [Chen *et al.*, 2010; Beck and Teboulle, 2004]. Moreover, the solutions to KMM and KLIEP tend to be sparse in practice (for example, see Section 6.1), which is a very suitable scenario for the Frank-Wolfe algorithm.

5.2 Complexity Analysis

Next we analyze the time and space complexities of the algorithms. Since both KMM and KLIEP are convex, the computation of the objective is not necessary in each iteration. Instead, one can simply check the gradient of the objective and stop when the norm of the gradient is sufficiently small.

We first focus on KMM. In this case, one can rewrite the objective in (3) in matrix form as

$$\min_{\boldsymbol{w} \in \mathcal{W}} \quad \frac{1}{2} \boldsymbol{w}^T K_{\text{tr}} \boldsymbol{w} - \boldsymbol{k}_{\text{te}}^T \boldsymbol{w}, \tag{7}$$

where

$$(K_{\text{tr}})_{ii'} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_{i'}) \rangle,\ (\boldsymbol{k}_{\text{te}})_i = \frac{1}{m} \sum_{j=1}^{m} \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle,$$

and the constant term is ignored. For the projected gradient method, the bottleneck is the computation of the gradient. From (7) it is obvious that $O(n^2)$ multiplications are required to compute the gradient in each iteration. This cannot be simplified because the projection to $\mathcal{W}$ creates a nonlinear operation on $\boldsymbol{w}_t$ in each iteration. Projected gradient requires additional $O(n)$ space to store the gradient. For Frank-Wolfe using a $1/(t+1)$ step size, we can see from (2) that the time complexity is $O(n)$ per iteration. As s_{t+1} must be a single $\phi(\boldsymbol{x}_{i^\star})$ for some $i^\star$, we only need to maintain a score list of size n to store $\{\langle \widehat{\mu}_t - \mu_p, \phi(\boldsymbol{x}_i) \rangle\}$ for all i. After decomposition, $\langle \mu_p, \phi(\boldsymbol{x}_i) \rangle$ is simply $(\boldsymbol{k}_{\text{te}})_i$ and can be precomputed, while $\langle \widehat{\mu}_t, \phi(\boldsymbol{x}_i) \rangle$ can be easily maintained for each t as

$$\langle \widehat{\mu}_{t+1}, \phi(\boldsymbol{x}_i) \rangle = (1 - \rho_t) \langle \widehat{\mu}_t, \phi(\boldsymbol{x}_i) \rangle + \rho_t (K_{\text{tr}})_{ii^\star}.$$

This is faster than the projected gradient approach because there is no projection step in Frank-Wolfe, and the score list can be maintained efficiently with only $O(n)$ additional space. Finally, when line search is applied to ρ_t, the time complexity is still $O(n)$, since the line search step size is attained in closed form $\rho_t = \frac{\langle \mu_p - \widehat{\mu}_t,\ s_{t+1} - \widehat{\mu}_t \rangle}{\| s_{t+1} - \widehat{\mu}_t \|^2}$. We can decompose the inner products as before and see that all operations can be performed in $O(n)$ time. While the computation of $\| \widehat{\mu}_{t+1} \|^2 = \boldsymbol{w}_{t+1}^T K_{\text{tr}} \boldsymbol{w}_{t+1}$ seems to require $O(n^2)$ multiplications, we can further decompose it, using the fact that $\boldsymbol{w}_{t+1} = (1 - \rho_t) \boldsymbol{w}_t + \rho_t \delta \boldsymbol{w}_t$, as

$$\begin{aligned} &(1-\rho_t)^2 \boldsymbol{w}_t^T K_{\text{tr}} \boldsymbol{w}_t + 2(1-\rho_t) \rho_t \boldsymbol{w}_t^T K_{\text{tr}} \delta \boldsymbol{w}_t + \rho_t^2 \delta \boldsymbol{w}_t^T K_{\text{tr}} \delta \boldsymbol{w}_t \\ &= (1 - \rho_t)^2 \| \widehat{\mu}_t \|^2 + 2(1 - \rho_t) \rho_t \boldsymbol{w}_t^T (K_{\text{tr}})_{:i^\star} + \rho_t^2 (K_{\text{tr}})_{i^\star i^\star}, \end{aligned}$$

where $(K_{\text{tr}})_{:i^\star}$ means the $i^\star$th column of K_{tr}. Therefore, only $O(n)$ multiplications are required since the first term can be used recursively and $\delta \boldsymbol{w}_t$ only has one singular non-zero entry. Similar tricks cannot be applied to projected gradient because $\delta \boldsymbol{w}_t$ is not sparse in that case, and $\delta \boldsymbol{w}_t^T K_{\text{tr}} \delta \boldsymbol{w}_t$ still requires $O(n^2)$ computation in general. An additional $O(n)$ storage is still needed for the score list in the line search case.

Table 1: Complexities of projected gradient (PG) and Frank-Wolfe with $1/(t+1)$ or line search step size. Time complexity is the complexity of one iteration. The additional required space is shared across iterations.

		PG	$1/(t+1)$	Line
KMM	Time	$O(n^2)$	$O(n)$	$O(n)$
	Space	$O(n)$	$O(n)$	$O(n)$
KLIEP	Time	$O(mb)$	$O(mb)$	$O(mb)$
	Space	$O(m+b)$	$O(m+b)$	$O(m+b)$

Next we investigate the complexity of KLIEP. Similar to KMM, the bottleneck is the computation of the gradient (5). The projected gradient method requires $2mb$ multiplications to compute (5) for all l, while Frank-Wolfe only requires $2m+mb$ multiplications, since the denominator is linear in $\boldsymbol{\alpha}$ and can be updated efficiently due to (6). This can lead to different run time in practice (see Section 6). Line search is less efficient for KLIEP(FW) compared to KMM(FW) as there is no closed form solution to ρ_t. Note that $F(\boldsymbol{\alpha}_{t+1}) = G(\rho_t)$ is a concave function of ρ_t. Moreover, the gradient $G'(\cdot)$ is decreasing in (0,1) with $G'(0) > 0$. If $G'(1) \geqslant 0$, a step size of 1 is used, otherwise $G'(1) < 0$ and we use binary search on the interval $[0, 1]$ to find ρ_t such that $G'(\rho_t) = 0$. Such a binary search requires $O(m)$ multiplications. All three versions require $O(m+b)$ space to compute and store the gradient. Table 1 summarizes the results of this section.

6 Experiments

In this section, we demonstrate the results of the proposed approach on both synthetic and some benchmark datasets. In particular, we compare the proposed Frank-Wolfe method on KMM and KLIEP with the projected gradient (PG) method. In the experiments, a Gaussian kernel is applied to KMM where the kernel width is chosen to be the median of the pairwise distances over the training set. For KLIEP, the width is chosen according to the criterion of Sugiyama *et al.* [2008].

6.1 Synthetic Datasets

To investigate the efficiency of the different methods, we compare their runtime to reach the same accuracy. Synthetic data is generated from $y = x^3 - x + 1 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.1^2)$. Training points are generated from $\mathcal{N}(0.5, 0.5^2)$ while test points are generated from $\mathcal{N}(0, 0.3^2)$ [Shimodaira, 2000]. In the experiment, $n = m$. We first run the projected gradient method to convergence, then run the Frank-Wolfe methods (with two different step sizes: $\rho_t = 1/(t+1)$ and line search) until it reaches the same objective. Figure 1 shows the runtime in log scale with varied sample sizes. The Frank-Wolfe methods are consistently faster than the projected gradient method. In the case of KLIEP, although Frank-Wolfe and projected gradient have the same theoretical complexity and convergence as shown in Section 5, their actual runtime can differ in practice, as the constant factor in the big-O notation can be different. Also, note that line search has better efficiency in the case of KLIEP, since the final solution is usually extremely sparse, meaning that line search can find it in a few iterations while $1/(t+1)$ takes some more time.

We next check the solution quality of the different methods on the same problems, to determine the effectiveness of Frank-Wolfe. We generated $n = m = 128$ data points and show the final training weights in Figures 2 and 3. On one hand, the weights of the three KMM methods differ from each other significantly, but their objective values remain very close. This implies that although the objective of KMM (3) is convex, there are many "optimal" solutions (in terms of objective values) that minimize the discrepancy between the means in the RKHS. This also leads to different behaviours in the underling task (regression or classification, etc). No further criteria is proposed in the literature to distinguish which solution is preferable for KMM. For example, we include a red line and a blue line in each graph, showing the linear models learned by least squares from the unweighted (i.e., with uniform weights) and reweighted training set, respectively. The blue lines of the three KMM methods are slightly different from each other. More results can be found on benchmark datasets in Section 6.2. On the other hand, although KLIEP also demonstrates multiple solutions with close objective values, the final weights produced by the three KLIEP methods are very similar, as shown in Figure 3. Note that Frank-Wolfe tends to find sparser solutions than projected gradient in general, which suggests the effectiveness of Frank-Wolfe in finding sparse solutions if they exist.

6.2 Benchmark Datasets

Next we applied the reweighting methods on some benchmark datasets from the libsvm[2] and delve[3] libraries to show their performance in correcting covariate shift on reweighted learning. All methods are run until convergence.

For each dataset, we first normalize the input features and the output to $[-1, 1]$ if it is significantly out of scale. We introduce a covariate shift by the following scheme. First we compute the first (with largest eigenvalue) principle component of the dataset. Let $z_{\max}$ and $z_{\min}$ be the maximum and minimum projected values along the principle direction and σ_z be the standard deviation of the projected values. In each trial, we first draw $m = 2000$ test points without replacement from the pool, with probability proportional to $\mathcal{N}(z_{\max}, \sigma_z^2/4)$. After eliminating these test points from the dataset, we then draw $n = 5000$ training points without replacement from the pool, with probability proportional to $\mathcal{N}(z_{\min}, \sigma_z^2/4)$. After computing the weights for the training points, we learn a linear model from the reweighted set and evaluate the model performance on the test set, using hinge loss for classification and squared loss for regression. We compare the performance of the reweighting methods with the unweighted method (i.e., with uniform weights for training points). For KMM(FW) we use $1/(t+1)$ as the step size, while for KLIEP(FW) we use the line search step size, since these choices are faster in practice. In addition to the projected gradient solver KLIEP(PG), we also compare the KLIEP(FW) method with the implementation of the original authors KLIEP(100), where the "$b = 100$" trick is used for projected gradient when there are too many test instances

[2] www.csie.ntu.edu.tw/∼cjlin/libsvm

[3] www.cs.toronto.edu/∼delve/data/datasets.html

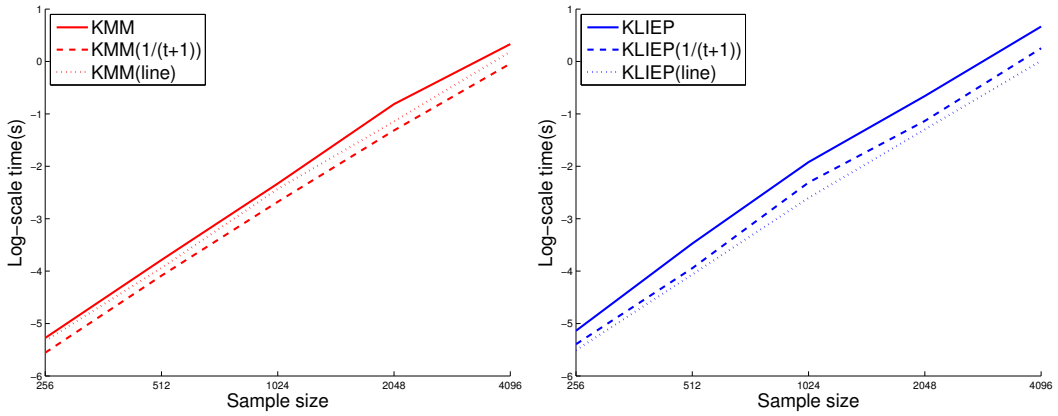
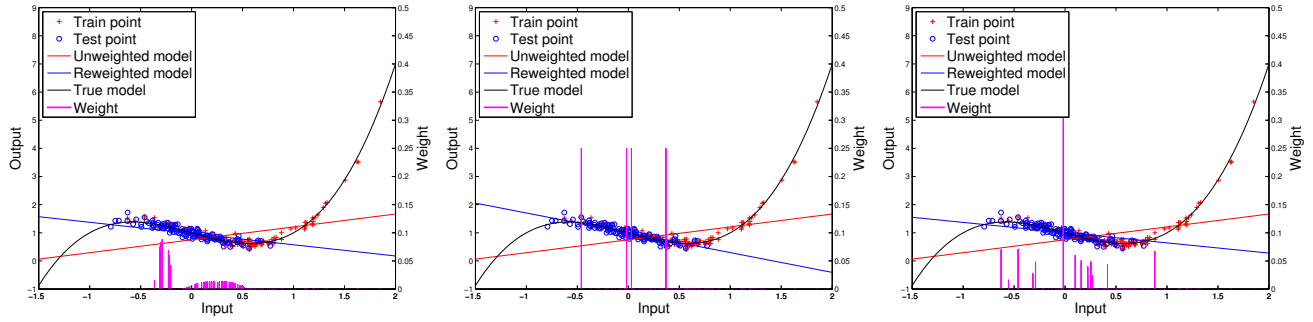


Figure 1: Runtime plot over different sample sizes. Time is measured in seconds.



(a) Projected gradient method. (b) Frank-Wolfe with $1/(t+1)$ step size. (c) Frank-Wolfe with line search step size.

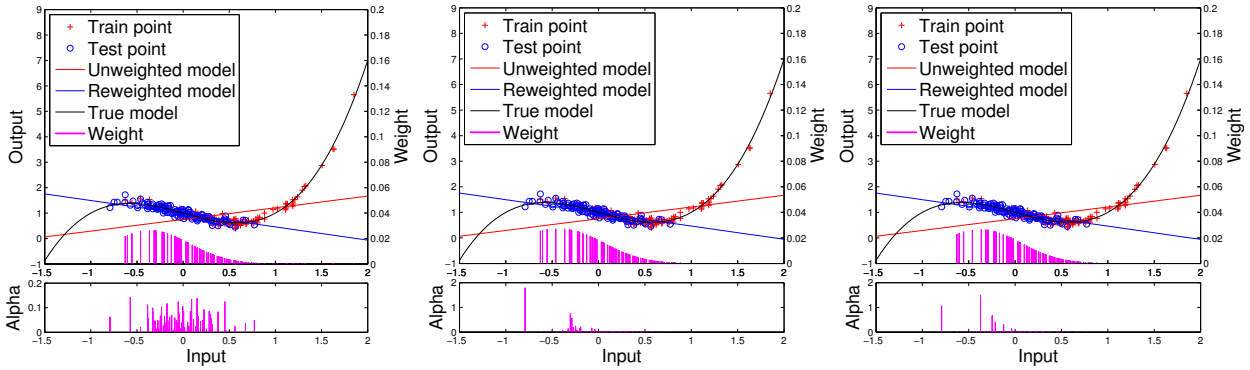
Figure 2: Illustrating the training point weights produced by different KMM optimization methods. Without reweighting, least squares outputs the red line as the best linear fit. By adjusting the weights of the training points (as shown by the purple vertical bars), a new linear fit (blue line) can be found for better test set prediction. Notice that the three methods produce significantly different training weights (purple vertical bars), but their final objective values are very close.

(only randomly select 100 Gaussian centres; see Section 4.2 for more details).

Table 2 shows the test loss and Table 3 shows the runtime over 50 trials. First, note that the performance of KMM is not very reliable because 1) the kernel width used in the experiment is difficult to tune, as observed by others [Cortes *et al.*, 2008] and 2) the solution to KMM tends to be sensitive to random initialization – i.e., different solutions may have similar objective values – which leads to unstable behaviour on the underlying task. Next, we can see that KLIEP(FW) is effective in correcting covariate shift. In most cases its test loss is smaller than unweighted learning as well as KLIEP(PG) on the test set. One possible reason is that KLIEP(PG) may fail to improve the objective on a step and stop prematurely. Comparing KLIEP(PG) and KLIEP(100), we can see that using all information from the test set will probably reduce test loss, although it may suffer from increased runtime. Finally, judging from Table 3, Frank-Wolfe is a very efficient algorithm compared to projected gradient in the current setting where sparse solutions are possible. KLIEP(100) is extremely fast but its task performance is so restrictive that it becomes impractical. We may apply values other than $b = 100$, but the best value of b is dataset-dependent and could be hard to identify. Instead, FW does not require parameter tuning and can utilize all Gaussians without computational issues.

7 Conclusion

We have proposed an efficient method that can be incorporated into two well-known covariate shift correction techniques, Kernel Mean Matching (KMM) and Kullback-Leibler Importance Estimation Procedure (KLIEP). Our approach is inspired by noticing a connection between kernel herding and KMM. By exploiting sparse solution structure, we apply the Frank-Wolfe algorithm to KMM and KLIEP. Since Frank-Wolfe is projection-free, its time complexity per iteration is in general smaller than the traditional projected gradient method. Our convergence and complexity analysis show that KMM(FW) has an advantage over the projected gradient method, while KLIEP(FW) has runtime performance comparable to the projected gradient method. Our empirical studies show that Frank-Wolfe is very suitable and practical for finding optimal solutions and correcting covariate shift as the sample size grows. Whether Frank-Wolfe can be applied to other covariate shift correction techniques remains an open question. We are also exploring ways to incorporate covariate shift correction to the underlying task for better performance.



(a) Projected gradient method. (b) Frank-Wolfe with $1/(t+1)$ step size. (c) Frank-Wolfe with line search step size.

Figure 3: Illustrating the training point weights produced by different KLIEP optimization methods. These methods choose very different Gaussian centres (the values shown in the $\boldsymbol{\alpha}$ graph at the bottom), but their training weights (purple vertical bars in the top graphs) are very similar to each other, obtaining very close final objective values. Note that the mixing coefficients $\boldsymbol{\alpha}$ in (a) have a different scale for better visualization. The solutions obtained by Frank-Wolfe are sparser than that of projected gradient. The final solution using line search in (c) has only 7 Gaussian components, which is extremely sparse compared to all 128 possible Gaussian candidates. This shows the capability of Frank-Wolfe method in finding sparse solutions.

Table 2: Test losses for the different covariate shift correction methods over 50 trials (mean with standard error in parentheses). Weights are computed using either Frank-Wolfe (FW) or projected gradient (PG). Losses are normalized so that the mean loss of the unweighted method is 1.0. The upper half of the table considers classification datasets (hinge loss), while the lower half considers regression datasets (squared loss). The best method(s) according to the Wilcoxon signed-rank test at the 5% significance level is(are) shown in bold for each dataset.

Dataset	Unweighted	KLIEP(FW)	KLIEP(PG)	KLIEP(100)	KMM(FW)	KMM(PG)
a7a	1.000 (0.006)	0.978 (0.007)	0.998 (0.015)	0.999 (0.009)	0.968 (0.024)	**0.955** (0.023)
a8a	1.000 (0.006)	**0.971** (0.008)	1.002 (0.020)	0.997 (0.015)	**0.969** (0.027)	**0.959** (0.025)
a9a	1.000 (0.005)	**0.968** (0.010)	0.983 (0.023)	0.998 (0.012)	0.963 (0.025)	0.965 (0.023)
mushrooms	1.000 (0.077)	1.010 (0.078)	0.981 (0.072)	1.078 (0.109)	**0.800** (0.080)	0.923 (0.060)
cpusmall	1.000 (0.001)	**0.893** (0.023)	0.943 (0.010)	0.936 (0.016)	1.034 (0.034)	0.978 (0.023)
kin-8fh	**1.000** (0.003)	**1.014** (0.043)	4.462 (0.511)	5.872 (1.212)	7.686 (2.566)	3.594 (1.059)
kin-8fm	1.000 (0.004)	**0.738** (0.048)	5.060 (0.690)	5.329 (0.909)	3.028 (1.070)	1.786 (0.522)
kin-8nh	**1.000** (0.002)	**1.018** (0.040)	2.032 (0.329)	2.659 (0.454)	2.752 (0.398)	1.913 (0.370)
kin-8nm	1.000 (0.004)	**0.803** (0.017)	1.172 (0.112)	1.396 (0.219)	1.031 (0.093)	0.899 (0.055)

Table 3: Average runtime in seconds over 50 trials (mean with standard error in parenthesis). KLIEP(FW) is generally faster than KLIEP(PG). Although KLIEP(100) is extremely fast, its corresponding performance on the learning task is very restrictive. KMM(FW) is faster than KMM(PG) for every dataset.

Dataset	KLIEP(FW)	KLIEP(PG)	KLIEP(100)	KMM(FW)	KMM(PG)
a7a	6.22 (0.03)	5.29 (2.59)	0.07 (0.04)	1.32 (0.02)	1.76 (0.05)
a8a	6.25 (0.04)	7.17 (2.92)	0.04 (0.00)	1.31 (0.02)	1.78 (0.04)
a9a	6.28 (0.04)	4.39 (2.38)	0.06 (0.04)	1.30 (0.02)	1.77 (0.05)
mushrooms	4.19 (0.21)	23.90 (0.02)	0.89 (0.10)	1.18 (0.02)	1.97 (0.10)
cpusmall	1.24 (0.02)	3.18 (1.13)	0.08 (0.03)	1.12 (0.03)	3.32 (0.21)
kin-8fh	5.61 (0.21)	23.79 (0.02)	0.76 (0.13)	1.06 (0.02)	1.56 (0.06)
kin-8fm	5.60 (0.21)	23.78 (0.03)	0.60 (0.14)	1.07 (0.02)	1.53 (0.06)
kin-8nh	5.46 (0.30)	23.85 (0.02)	0.72 (0.13)	1.11 (0.02)	1.53 (0.06)
kin-8nm	5.69 (0.03)	23.76 (0.02)	0.73 (0.13)	1.08 (0.02)	1.55 (0.05)

References

[Agarwal *et al.*, 2011] Deepak Agarwal, Lihong Li, and Alex J Smola. Linear-time estimators for propensity scores. In *International Conference on Artificial Intelligence and Statistics*, pages 93–100, 2011.

[Bach *et al.*, 2012] Francis Bach, Simon Lacoste-Julien, and Guillaume Obozinski. On the equivalence between herding and conditional gradient algorithms. In *International Conference on Machine Learning*, pages 1359–1366, 2012.

[Bach, 2015] Francis Bach. Duality between subgradient and conditional gradient methods. *SIAM Journal on Optimization*, 25(1):115–129, 2015.

[Beck and Teboulle, 2004] Amir Beck and Marc Teboulle. A conditional gradient method with linear rate of convergence for solving convex linear systems. *Mathematical Methods of Operations Research*, 59(2):235–247, 2004.

[Bertsekas, 1999] Dimitri. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 2nd edition, 1999.

[Bickel *et al.*, 2007] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *International Conference on Machine Learning*, pages 81–88, 2007.

[Chen *et al.*, 2010] Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Uncertainty in Artificial Intelligence*, pages 109–116, 2010.

[Cortes *et al.*, 2008] Corinna Cortes, Mehryar Mohri, Michael Riley, and Afshin Rostamizadeh. Sample selection bias correction theory. *Algorithmic Learning Theory*, 5254:38–53, 2008.

[Gretton *et al.*, 2009] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. Covariate shift by kernel mean matching. *Dataset Shift in Machine Learning*, pages 131–160, 2009.

[Hachiya *et al.*, 2009] Hirotaka Hachiya, Takayuki Akiyama, Masashi Sugiayma, and Jan Peters. Adaptive importance sampling for value function approximation in off-policy reinforcement learning. *Neural Networks*, 22(10):1399–1410, 2009.

[Jaggi, 2013] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pages 427–435, 2013.

[Jiang and Zhai, 2007] Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 264, 2007.

[Joulin *et al.*, 2014] Armand Joulin, Kevin Tang, and Li Fei-Fei. Efficient image and video co-localization with frank-wolfe algorithm. In *Computer Vision–ECCV 2014*, pages 253–268. Springer, 2014.

[Quionero-Candela *et al.*, 2009] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[Salamatian *et al.*, 2014] Salman Salamatian, Nadia Fawaz, Branislav Kveton, and Nina Taft. Sppm: Sparse privacy preserving mappings. In *Conference on Uncertainty in Artificial Intelilgence*, 2014.

[Shimodaira, 2000] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

[Sugiyama *et al.*, 2008] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul Von Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in Neural Information Processing Systems*, 2008.

[Sugiyama *et al.*, 2012] Masashi Sugiyama, Taiji Suzuki, and Takafumi Kanamori. *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[Tsuboi *et al.*, 2009] Yuta Tsuboi, Hisashi Kashima, Shohei Hido, Steffen Bickel, and Masashi Sugiyama. Direct density ratio estimation for large-scale covariate shift adaptation. *Information and Media Technologies*, 4(2):529–546, 2009.

[Welling, 2009] Max Welling. Herding dynamical weights to learn. In *International Conference on Machine Learning*, pages 1121–1128, 2009.

[Wen *et al.*, 2014] Junfeng Wen, Chun-Nam Yu, and Russell Greiner. Robust learning under uncertain test distributions: Relating covariate shift to model misspecification. In *International Conference on Machine Learning*, pages 631–639, 2014.

[Yamada *et al.*, 2010] Makoto Yamada, Masashi Sugiyama, and Tomoko Matsui. Semi-supervised speaker identification under covariate shift. *Signal Processing*, 90(8):2353–2361, 2010.

[Yamada *et al.*, 2011] Makoto Yamada, Taiji Suzuki, Takafumi Kanamori, Hirotaka Hachiya, and Masashi Sugiyama. Relative density-ratio estimation for robust distribution comparison. In *Advances in Neural Information Processing Systems*, pages 594–602, 2011.

[Yamada *et al.*, 2012] Makoto Yamada, Leonid Sigal, and Michalis Raptis. No bias left behind: Covariate shift adaptation for discriminative 3d pose estimation. In *Computer Vision–ECCV 2012*, pages 674–687. Springer, 2012.

[Zadrozny, 2004] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *International Conference on Machine Learning*, page 114. ACM, 2004.

[Zhang *et al.*, 2012] Xinhua Zhang, Dale Schuurmans, and Yao-liang Yu. Accelerated training for matrix-norm regularization: A boosting approach. In *Advances in Neural Information Processing Systems*, pages 2906–2914, 2012.