Importance Reweighting Using Adversarial-Collaborative Training

Yifan Wu
yw4@andrew.cmu.edu

Tianshu Ren
tren@andrew.cmu.edu

Lidan Mu
lmu@andrew.cmu.edu

Abstract

We consider the problem of reweighting a source dataset D_S to match a target dataset D_T, which plays an important role in dealing with the covariate shift problem. One of the common approaches to reweight the source data to match the distribution of target data is to use kernel mean matching, which tries to learn the likelihood ratios by minimizing the kernel mean discrepancy. In this work, we first drive a counterpart for the kernel mean matching technique by replacing the kernel mean discrepancy with the adversarial training objective. Then we argue that likelihood ratio based reweighting may not be the best choice for the covariate shift problem in terms of low effective sample size. To balance between the distribution matching and the effective sample size, we further propose another learning objective that contains a "collaborator" in addition to the adversary. The effectiveness of our approach is shown by experiment results on both synthetic data and real data.

1 Introduction

Covariate shift is a common problem when dealing with real world machine learning problems. It is quite often that the training data and test data come from different distributions. Existing approaches [1, 2, 3, 4] often contain two stages: The first stage is a "data reweighting" process, where the training data are reweighted such that its distribution is matched to the test data (in this stage only the feature vectors are considered, without any labels). And the second stage is to train the model with "reweighted" loss functions. Here we only consider the first stage.

One way of reweighting the data is called kernel mean matching [2], where the weights over the training data are optimized to minimize the kernel mean discrepancy. In kernel meaning matching, the kernel mean discrepancy can be viewed as a divergence measure of two sample sets. [5] proposed another way to measure the divergence between two sample sets by introducing an "adversarial discriminator". The intuition is that the two sample sets are more likely to be from the similar distributions if they are harder to be differentiated by a classifier.

In this work, we first investigate the possibility of replacing the kernel mean matching objective with adversarial training. We empirically show that, on simple synthetic datasets, the importance weights can be learned by the adversarial training process. Then we argue that, in covariate shift [1] or other potential applications, importance reweighting based on matching the likelihood ratios may not be the most effective choice due to the possibly low effective sample size. So we further propose an approach that is able to balance between distribution matching and effective sample size by adding a "collaborator". The idea is that, first we can interpret the importance weights as the acceptance rate in doing subsampling from the source data. Then we can push up the effective sample size by simultaneously maximizing the discrepancy between the rejected samples and the target dataset. In

[1] here we only consider the reweighting technique, excluding the discussion of transfer learning in domain adaptation as in [6]

other words, we will be able to sample as much relevant data as possible from the source dataset in order to perform further tasks with the target dataset.

The learning outcome of our adversarial-collaborative training can be also viewed as a classifier between population A and B. The classifier is learned from a sample set of population A and a mixed unlabelled sample set $A \cup B$, which is a semi-supervised classification problem with labelled data only from one class and unlabelled data. For example, we want to learn a classifier to classify cat and dog but what we only know is a labelled set of dogs only and an unlabelled mixture set of cat and dogs. To the best of our knowledge, we are not aware of any algorithm that can directly learn the classifier in this setting.

The structure of this report is as follows. In Section 2 we first formally define our problem of importance reweighting and then provide another view of our problem as semi-supervised classification. In Section 3 we introduce how the idea of adversarial training can be applied to our problem and further propose our adversarial-collaborative training algorithm. In Section 4 we empirically investigate the effectiveness of our algorithm on both synthetic dataset and MNIST dataset.

2 Problem Statement

2.1 Importance reweighting under covariate shift

Suppose we have a source data distribution $p_S(\cdot)$, a target data distribution $p_T(\cdot)$, and we are doing rejection sampling from $p_S(\cdot)$ to match $p_T(\cdot)$ with acceptance probability $\beta(\cdot) \in [0, 1]$. We define the distribution of the sample outcomes as $p_{\beta(S)}(\cdot)$ where $p_{\beta(S)}(x) = \frac{p_S(x)\beta(x)}{\int \beta(x)p_S(x)dx}$. The goal is to learn $\beta(\cdot)$ from finite samples $D_S \sim p_S(\cdot)$ and $D_T \sim p_T(\cdot)$ to minimize $\mathbb{D}\left(p_{\beta(S)}, p_T\right)$ for some divergence measure $\mathbb{D}(p, q)$. It can be shown that this problem is the equivalent to kernel mean matching [2] when $\mathbb{D}(p, q) = \|\mathbb{E}_{x\sim p}[\Phi(x)] - \mathbb{E}_{x\sim q}[\Phi(x)]\|_{\mathcal{H}}$ for some feature map $\Phi(\cdot)$ implicitly defined by a kernel function. The optimal solution is that β is proportional to the likelihood ratio, i.e.

$$\beta(x) \propto \frac{p_T(x)}{p_S(x)}$$

for all x, such that $p_{\beta(S)}$ and p_T are the same distribution.

Note that here we interpret β as "acceptance probability" in rejection sampling instead of "importance reweighting" because it brings convenience to define a distribution $p_{\beta(S)}(\cdot)$ over x and use the term "divergence measure". Since under both interpretation the goal is just to learn the likelihood ratio we can use learned β as either acceptance probabilities or importance weights.

Different from kernel mean matching where an individual weight β_x is learned for all data point $x \in D_S$, our goal here is to learn a function mapping $\beta(\cdot)$ that maps the feature space to $[0, 1]$, which provides (i) smoothing on the weights such that similar points will have similar weights and (ii) the ability of generalization to unseen data and may have potential more applications other than the simple covariate shift setting.

2.2 Semi-supervised classification with one-class labelled data

We can also view the learned β as a binary classifier between "from target data distribution" and "not from target data distribution". If these two classes have well separated meanings (e.g. "target distribution" = "dogs", "source distribution" = "cats and dogs" and thus "not in target distribution" = "cats") then this becomes a semi-supervised classification problem with one-class labelled data.

More formally, there is a joint data distribution $p(x, y)$ where $x \in R^d$ and $y \in \{0, 1\}$. The source data distribution D_S contains i.i.d. samples drawn from the marginal distribution $p_S(x) = p(x, y = 0) + p(x, y = 1)$, while the target data distribution D_T contains i.i.d. samples drawn from the conditional distribution $p_T(x) = p(x|y = 1)$. The goal is to learn a classifier $\beta(\cdot)$ such that

$$\beta(x) = p(y = 1|x)\,.$$

By Bayes' theorem we can show that

$$\beta(x) = \frac{p(x|y = 1)p(y = 1)}{p(x)} = \frac{p_T(x)p(y = 1)}{p_S(x)} \propto \frac{p_T(x)}{p_S(x)}\,,$$

which means that the view of semi-supervised classification is equivalent to the view of importance reweighting.

3 Algorithms

In this section we introduce two approaches to learn $\beta(\cdot)$. The first one is trying to learn the likelihood ratio as in kernel mean matching. After discussing some potential issues with the first approach, we introduce another algorithm by adding a collaborative classifier, which pushes $\beta(\cdot)$ to be a more "sample efficient" importance reweighter.

3.1 Importance reweighting with adversarial training

In the work of Generative Adversarial Nets [5], the goal is to learn a generative model that indirectly models the data distribution as a multi-layer perceptron by using finite data samples. It learns the generative model f by introducing an adversary g (a binary classifier) that tries to differentiate between samples generated from f and samples from the true data distribution p_{data}. Specifically, g is trained to make the best classification while f is trained to minimize the power of g. There training objective can be written as $\min_f \mathbb{D}(f, p_{data})$ where $\mathbb{D}(p, q) = \max_g \mathbb{E}_{x\sim p}[\log g(x)] + \mathbb{E}_{x\sim q}[\log(1 - g(x))]$.

Here our goal is to use this divergence measure to learn $\beta(\cdot)$, that is

$$\min_{\beta} \max_{g} \mathbb{E}_{x\sim p_{\beta(S)}}[\log g(x)] + \mathbb{E}_{x\sim p_T}[\log(1 - g(x))]\,. \tag{1}$$

To learn from finite number of samples for both source and target datasets, we derive the empirical optimization formulation for (1): Define

$$f_1(\theta, \lambda) = \frac{1}{\sum_{x\in D_S} \beta_\theta(x)} \sum_{x\in D_S} \beta_\theta(x) \log g_\lambda(x) + \frac{1}{|D_T|} \sum_{x\in D_T} \log(1 - g_\lambda(x))\,,$$

and the goal is to optimize

$$\min_{\theta} \max_{\lambda} f_1(\theta, \lambda)\,. \tag{2}$$

Optimizing (2) can be done by performing gradient descent/ascent on θ and λ alternatively [2]. Note that, for large datasets, optimizing over λ can be done by stochastic gradient descent but optimizing θ cannot. This is because the gradient has $\sum_{x\in D_S} \beta(x)$ in the denominator. One possible approach to improve the computational efficiency is to use the mini-batch optimization to get an estimation of $\sum_{x\in D_S} \beta(x)$. However, a small batch size leads to high computational efficiency but poor estimate of $\sum_{x\in D_S} \beta(x)$ while a large batch size leads a good estimate of $\sum_{x\in D_S} \beta(x)$ but poor computational efficiency. Bad estimate of $\sum_{x\in D_S} \beta(x)$ negatively affects the convergence performance of the optimization. In practice, we found another approach that performs better which uses delayed update of $\sum_{x\in D_S} \beta(x)$. Updating $\sum_{x\in D_S} \beta(x)$ only once per-epoch while using stochastic gradient descent gives both good computation efficiency and convergence performance.

One problem with this algorithm is that $\beta_\theta(x)$ can be arbitrarily small (as long as it is proportional to the likelihood ratio) if we optimize the above objective (2). It is fine to use the small weights for sample reweighting in the covariate shift scenario but if the goal of learning β is to do rejection sampling (as described in Section 2) with a finite number of available source samples D_S then small $\beta_\theta(x)$ will lead to low acceptance rate, which means that we ignore a lot of useful samples. So attempting to accept as many samples that are likely from the target distribution as possible is one of the motivations that we propose the second algorithm.

Another motivation is a general issue with sample reweighting in the covariate shift scenario: the bias variance trade-off (as discussed in [3]). Using the likelihood ratio as importance weights will give an unbiased estimate of the Bayes risk on the test data distribution. However, the importance weights may lead to high variance due to the possibly low effective sample size $\frac{\|\beta\|_1^2}{\|\beta\|_2^2}$. Moreover, in learning tasks without model misspecification, it can be shown that, even if there is a mismatch between

[2] Here we will not discuss which optimization technique is good for optimizing the objective since how to optimize the adversarial training objective with good convergence guarantee is still an open problem.

the training and test distributions, doing empirical risk minimization **without** any reweighting is the optimal thing to do as discussed in [7]. Since assuming no model misspecification is kind of too strong, a good reweighting should balance between the distribution matching (no bias) and the highest effective sample size (no reweighting). Here what we are trying to achieve is to push $\beta(x)$ to be 1 for all x that is a point of interest according to the target dataset while pushing $\beta(x)$ to be 0 if x is not a point of interest, which will lead to a balance between the distribution matching and high effective sample size. [3]

3.2 Sample-efficient reweighting with adversarial-collaborative training

Motivated by the issues discussed above, here we propose an algorithm that learn a $\beta(\cdot)$ that can both (i) sample as many points of interest in the source dataset as possible (have high effective sample size) (ii) reject points that are not likely from the target distribution. The idea is to simultaneously maximize the divergence between the rejected samples and the target samples using a "collaborator" h. That is, by defining

$$f_2(\theta, \lambda) = \frac{1}{\sum_{x \in D_S} 1 - \beta_\theta(x)} \sum_{x \in D_S} (1 - \beta_\theta(x)) \log h_\lambda(x) + \frac{1}{|D_T|} \sum_{x \in D_T} \log(1 - h_\lambda(x)) \,,$$

we want to optimize

$$\min_\theta \left(\max_{\lambda_1} f_1(\theta, \lambda_1) - \gamma \max_{\lambda_2} f_2(\theta, \lambda_2) \right) . \tag{3}$$

The parameter $\gamma > 0$ controls the balance between distribution matching and effective sample size. Small γ will make learned β closer to the likelihood ratio while large γ will make learned β have higher effective sample size. Note that when $\gamma = 0$ the objective is the same as (2). We further conjecture that, even with $\gamma = 1$, under certain assumptions on p_S and p_T, our algorithm can still consistently learn β in proportion to the likelihood ratio while using fewer samples. This conjecture is supported by our experiments on synthetic datasets introduced in Section 4.

Algorithm 1 shows the empirical optimization process we use in practice. One thing to be mentioned is that (4) is a biased gradient with respect to θ. As we discussed in 3.1, it is impossible to do pure SGD to update θ because the normalizers $\sum_{x \in D_S} \beta_\theta(x)$ and $\sum_{x \in D_S} 1 - \beta_\theta(x)$ also depend on θ and computing them in each update is costly. Alternatively, we use delayed terms A_t, B_t, C_t, D_t and only update them once in each epoch. In practice we found that these delayed updates does not hurt performance (increasing the update frequency does not lead to faster convergence but results in slow computation).

4 Experiments

In this section, we empirically show that our proposed algorithms can achieve certain properties.[4]

4.1 Synthetic dataset

We generate a one-dimensional synthetic dataset by sampling source dataset $D_S \sim \text{Uniform}(-1.5, 1.5)$ and target dataset $D_T \sim \text{Uniform}(-0.5, 0.5)$. We further generate labels $y = x^3 - x + 1 + \epsilon$, where $\epsilon \sim N(0, 0.1^2)$ to show why we may need reweighting to do regression (e.g. if we use linear regression here it is better to use data only from $[-0.5, 0.5]$). In our experiment, the number of source samples is 300 and the number of target samples is 100.

4.1.1 Visualizing learned weights

We compare the learned weights from our algorithms with kernel mean matching, where we use a rbf kernel with radius 1. In our experiments all β, g and h are $1 \to 16 \to 6 \to 1$ fully connected neural networks with relu activation functions.

[3] We are not arguing that this will lead to the optimal balance.

[4] Code can be downloaded at `https://github.com/Coco-o/CSAN`.

Algorithm 1 Empirical optimization for adversarial-collaborative training

Inputs: D_S, D_T function classes β, g, h, learning rate η.
Output: function β_θ.
for $t = 1, 2, ...$ **do**
  Compute $\beta_{\theta^t}(x)$ for all $x \in D_S$.
  Using SGD with η to update g, h for one epoch and get λ_1^t, λ_2^t.
  Compute $g_{\lambda_1^t}(x)$ and $h_{\lambda_2^t}(x)$ for all $x \in D_S$.
  $A_t = \frac{1}{|D_S|} \sum_{x \in D_S} \beta_{\theta^t}(x)$,
  $B_t = 1 - A_t$,
  $C_t = \frac{1}{A_t^2 |D_S|} \sum_{x \in D_S} \log g_{\lambda_1^t}(x) \beta_{\theta^t}(x)$.
  $D_t = \frac{1}{B_t^2 |D_S|} \sum_{x \in D_S} \log h_{\lambda_2^t}(x)(1 - \beta_{\theta^t}(x))$.
  Set $\theta = \theta^t$.
  for $s = 1, 2, ..., |D_S|$ **do**

$$\theta \leftarrow \theta - \eta \left(\frac{\log g_{\lambda_1}(x_s)}{A_t} + \frac{\log h_{\lambda_2}(x_s)}{B_t} - C_t - D_t \right) \frac{\partial \beta_\theta(x_s)}{\partial \theta} . \tag{4}$$

  end for
  Set $\theta^{t+1} = \theta$.
end for

Figure. 1 shows the learned β by different algorithms. We can see that the weights learned by either kernel mean matching or the naive adversarial training in section 3.1 suffer from low effective sample size. But the adversarial-collaborative training algorithm we proposed do have expected properties (reweighting all relevant data samples uniformly while reject irrelevant ones).

More interestingly, since we generate points from uniform distributions the likelihood ratios in $[-0.5, 0.5]$ should be same, which means that our second algorithm performs even better on learning β to be the likelihood ratio although its objective is sort of different. This evidence supports our previously mentioned conjecture that, under certain assumptions, our adversarial-collaborative algorithm can consistently learn the likelihood ratio while using fewer samples.

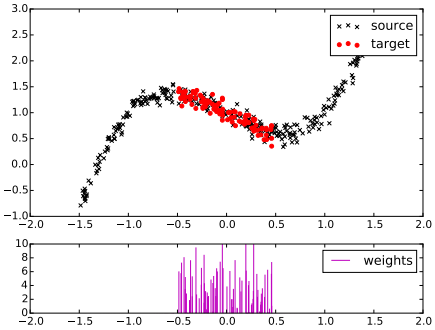
4.1.2 Convergence of the learning objective

Figure 2 shows the convergence of the crossentropy loss for the adversary g_{λ_1} and the collaborator h_{λ_2} ($-f_1$ and $-f_2$ respectively), from which we can see that, as expected, the loss of g_{λ_1} converges to a higher level while the loss of h_{λ_2} converges to a lower level.

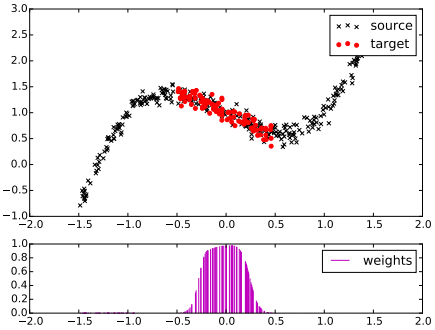
4.2 MNIST dataset

We also experiment on the MNIST dataset where we sample 5000 images with equal number of observations from each label as the source dataset. For the target dataset, we include only certain digits and sample 500 images for each digit while making sure there is no overlap with the source dataset. For each of the target dataset, we use both our model and kernel mean matching as a sampler/classifier and compare the quality of the samples picked out by these two algorithms from the source dataset. Specifically, for our model, we accept a sample if the acceptance probability $\beta(x) > 1/2$. For kernel mean matching, we accept a sample if the weight is greater than average. Although this choice may seem arbitrary, we could not find a simple way [5] to decide the cut-off threshold without knowing about the proportion of data in D_S that are similar to D_T. (In the semi-supervised classification view, this means we do not know $p(y = 1)$).

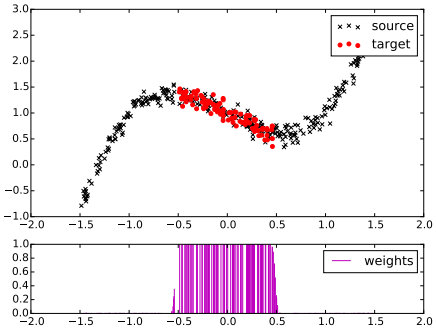
[5] There might be better but more complicated ways to decide the cut-off threshold than the average but we did not look into them.



(a) Kernel mean matching



(b) Adversarial training ($\gamma = 0$)



(c) Adversarial-collaborative training ($\gamma = 1$)

Figure 1: Visualizing learned β on 1-D synthetic dataset

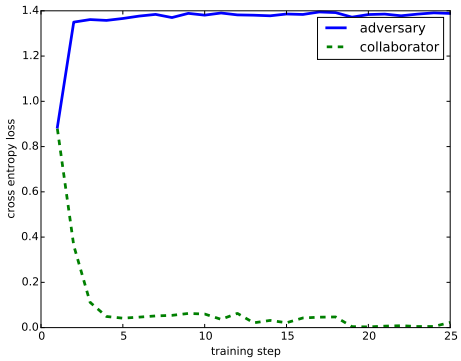
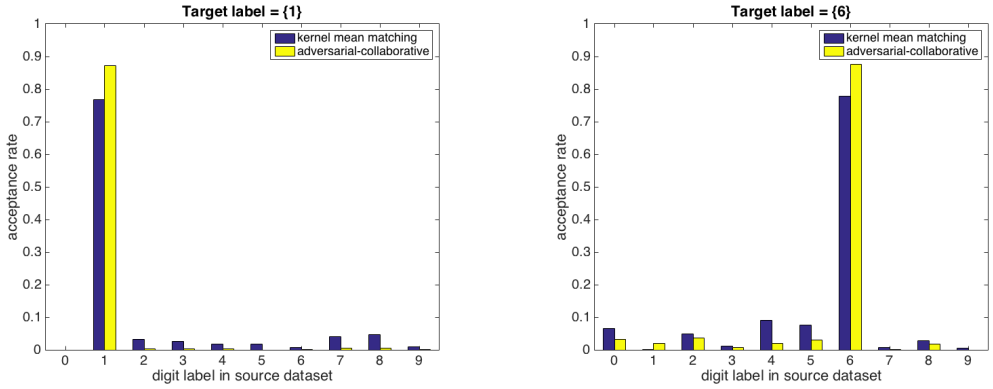


Figure 2: Convergence of the loss for adversary g and collaborator h.

In our experiments all β, g and h are $784 \to 200 \to 100 \to 1$ fully connected neural networks with relu activation functions. Typically using 10 to 20 epochs in training adversarial-collaborative network has the best performance. For kernel mean matching we use the degree-2 polynomial kernel.[6].

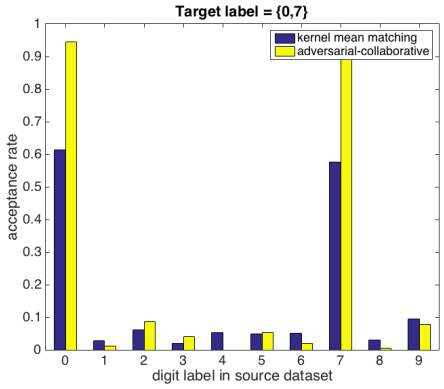
Figure 3 shows the sampling results of both our adversarial-collaborative model and the KMM model under several target distributions. As the result shows, our learned sampler has a clear advantage over the kernel mean matching sampler in almost all target distributions. In general, our sampler is able to achieve high acceptance rate on correct labels and low acceptance rate on incorrect labels,

[6]Theoretically one should use a kernel with infinite dimensional feature space such as the rbf kernel but we found that in high dimension the rbf kernel always performs poorly and hard to tune

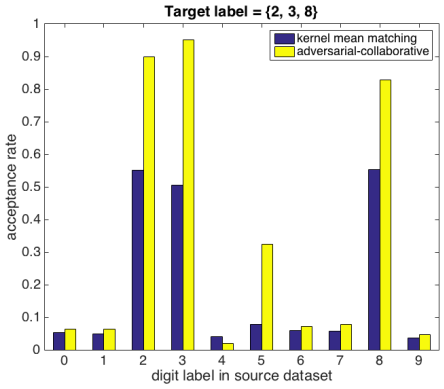


(a) Target label = {1}

(b) Target label = {6}



(c) Target label = {0, 7}



(d) Target label = {2, 3, 8}

Figure 3: Acceptance rate for our method and kernel mean matching under target dataset label {1}, {6}, {0, 7} and {2, 3, 8}

irrespective of the complexity of the target distribution. As we argued in Section 3.1, kernel mean matching suffers from low effective sample size (lower acceptance rate at target labels) although it can reject most irrelevant samples.

According to our motivation, a good sampler should be able to retain as many samples that has the same label as the target dataset as possible, and reject as many samples that does not match with the target dataset as possible. This is in spirit identical to doing classification on samples in the source dataset. Thus for the purpose of a more comprehensive comparison, we also evaluate the models using precision and recall on a number of target datasets. Table 1 shows a comparison of the quality of samples from our adversarial-collaborative algorithm and kernel mean matching.

One detail to emphasize is that although labels of the digits are used for evaluation, the training process is completely unsupervised. Our model has no notion of label, but only utilizes the knowledge of whether an observation is from the target dataset. Thus it is fully expandable to other types of data, as long as a source and a target dataset is provided.

5 Conclusions

In this work we propose an alternative approach to perform importance reweighting in the covariate shift scenario using adversarial training instead of kernel mean matching. We also propose an adversarial-collaborative training objective to learn importance weights that are balanced between the effective sample size and the distribution matching (bias). Experimental results show that our approach is able to achieve good performance in both importance reweighting and semi-supervised classification scenarios. One potential advantage is the flexibility of our model: (i) We do not explicitly model any probability density functions (KMM also has this property but some others

Target Set	AC Pre	AC Rec	AC Acc	KMM Pre	KMM Rec	KMM Acc
{0}	93.19%	87.60%	**98.12%**	73.56%	76.80%	94.92%
{1}	96.89%	87.20%	**98.44%**	79.50%	76.80%	95.70%
{2}	84.99%	83.80%	**96.90%**	61.79%	69.20%	92.64%
{3}	81.25%	75.40%	**95.80%**	65.82%	72.40%	93.48%
{4}	72.21%	95.60%	**95.88%**	67.41%	78.20%	94.04%
{5}	85.29%	80.00%	**96.62%**	65.85%	74.80%	93.60%
{6}	84.07%	87.60%	**97.10%**	69.84%	77.80%	94.42%
{7}	89.53%	90.60%	**98.00%**	68.65%	78.40%	94.26%
{8}	72.60%	72.60%	**94.52%**	53.05%	69.60%	90.80%
{9}	31.59%	96.60%	78.74%	58.96%	72.40%	**92.20%**
{0, 7}	86.55%	95.20%	**96.08%**	75.32%	59.50%	88.00%
{2,3,8}	80.04%	89.27%	**90.10%**	81.17%	53.73%	82.38%

Table 1: Quality of samples under different target dataset. The bold entries indicate higher accuracy among the two models. Meaning of acronyms: AC = Adversarial-Collaborative Training; KMM = Kernel Mean Matching; Pre = Precision; Rec = Recall; Acc = Accuracy

that we do not mentioned here do not have this property). (ii) Any binary classifiers can be used in the divergence measure while in KMM we need to explicitly specify a kernel. (iii) Our model can be efficiently trained by stochastic gradient descent and more applicable to large scale complex problems.

There are several interesting remaining tasks to be done in the future: (i) In the semi-supervised learning scenario, we only test the performance on classifying the given source dataset D_S. Testing the generalization performance on unseen data samples will also be of interest. (ii) Comparing a learned function $\beta(\cdot)$ with individual βs for each sample seems an unfair approach since learning a parametrized function will naturally add smoothing to the weights. A better comparison might be implementing KMM with a function mapping from data to weights, which also provides the ability of generalization. With limited time we were not able to find a computationally efficient way to perform optimization so we leave this as future work. (iii) We also plan to experiment our algorithm on more complex dataset than MNIST to see whether our approach have more advantage than the others. (iv) There might be many other approaches that can be used to tackle this problem but we did not have chance to implement them. Several interesting theoretical questions could be studied, including (v) why delayed update of the normalizer in empirical optimization does not hurt the convergence and (vi) why adding a collaborator helps learning the likelihood ratio (and what assumptions do we need to validate this).

References

[1] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning under covariate shift. *Journal of Machine Learning Research*, 10(Sep):2137–2155, 2009.

[2] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4):5, 2009.

[3] Sashank J Reddi, Barnabas Poczos, and Alex Smola. Doubly robust covariate shift correction. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2949–2955. AAAI Press, 2015.

[4] Junfeng Wen, Russell Greiner, and Dale Schuurmans. Correcting covariate shift with the frank-wolfe algorithm. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1010–1016. AAAI Press, 2015.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[6] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1180–1189, 2015.

[7] Junfeng Wen, Chun-Nam Yu, and Russell Greiner. Robust learning under uncertain test distributions: Relating covariate shift to model misspecification. In *ICML*, pages 631–639, 2014.