IN4320 Machine Learning
Final Assignment Report

Bagas Abisena Swastanto - 4415345

August 14, 2015

1 Introduction

Standard supervised learning works on the assumption that the training and testing data come from the same distribution. In real world, that is not always the case. We sometimes encounter problem where the training distribution $P_{tr}(x)$ differs from the testing distribution $P_{te}(x)$ but the posteriors remain the same, $P_{tr}(y|x) = P_{te}(y|x) = P(y|x)$. The situation deviates from the usual supervised learning, but we still desire to overcome the problem and obtain good generalization on the test data. The problem is known as learning under *Covariate Shift*. The MiniBooNe [1] dataset used in the assignment is biased under the covariate shift context.

The influence of covariate shift can be adapted into the learning process by importance sampling [2]. Before talking about importance sampling, let us revisit the concept of Empirical Risk Minimization (ERM) [3]. IN ERM, we want to find the optimum model from model family by minimizing the following objective function

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x,y) l(x,y,\theta) \tag{1}$$

where l is the loss function, which can be log likehood, regression error, etc. The true distribution is not known, therefore we estimate it from the data. Thus, the objective function become

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \tilde{P}(x,y) l(x,y,\theta) \tag{2}$$

$$= \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} l(x_i, y_i, \theta_i) \tag{3}$$

Minimizing the risk become more complicated due to covariate shift. We want to minimize the loss over the testing distribution

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P_{te}(x,y) l(x,y,\theta) \tag{4}$$

but the training instances sampled from different distribution, P_{tr}. Because of that, the loss equation now becomes

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \frac{P_{te}(x,y)}{P_{tr}(x,y)} P_{tr}(x,y) l(x,y,\theta) \tag{5}$$

$$= \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \frac{P_{te}(x,y)}{P_{tr}(x,y)} \tilde{P}_{tr}(x,y) l(x,y,\theta) \tag{6}$$

$$= \arg\min_{\theta \in \Theta} \sum_{i=1}^{N_{tr}} \frac{P_{te}(x_i^{(tr)}, y_i^{(tr)})}{P_{tr}(x_i^{(tr)}, y_i^{(tr)})} l(x_i^{(tr)}, y_i^{(tr)}, \theta_i) \tag{7}$$

where N_{tr} is the number of the training sample. From equation above we see that the each loss from training instance is weighted by the ratio $\frac{P_{te}(x,y)}{P_{tr}(x,y)}$. That ratio is the main idea behind importance sampling. In covariate shift, since the posterior is the same between training and testing, the ratio can be simplified into

$$\frac{P_{te}(x,y)}{P_{tr}(x,y)} = \frac{P_{te}(x)P(y|x)}{P_{tr}(x)P(y|x)} \tag{8}$$

$$= \frac{P_{te}(x)}{P_{tr}(x)} \tag{9}$$

Intuitively, we want to give higher weights to the training samples which falling on the region where the testing data is dense and penalize the training samples who don't. Shimodaira [2] proofs that the ratio is the optimal choice provided that the support of P_{te} is contained in the support of P_{tr}.

The report is organized as follows: section 2 will discuss the implementation of the assignment, section 3 shows the experimental result of the implementation, and lastly the section 4 is the conclusion.

2 Implementation

The implementation is focused on two main tasks. First is how to calculate the weighing ratio $\frac{P_{te}(x)}{P_{tr}(x)}$. Second is how to use the ratio directly in the learning algorithm.

2.1 Importance Sampling Ratio

Section 1 discuss that the training samples should be weighted by the ratio $\frac{P_{te}(x)}{P_{tr}(x)}$. In the context of the assignment, the result of this phase is a vector with size the training sample (24907) which then will be use in the learning algorithm to weight the loss function. Here I implement three methods to calculate the ratio. The first two involve estimating the $P_{te}(x)$ and $P_{tr}(x)$ separately then calculating the ratio, while the third method directly computes the ratio without estimating the $P_{te}(x)$ and $P_{tr}(x)$ beforehand.

2.1.1 Parametric

This method assume that both P_{te} and P_{tr} are normally distributed $\mathcal{N}(x; \mu, \Sigma)$. The parameter of the distribution, mean μ and covariance matrix Σ, are estimated from the data using Expectation Maximization (EM) algorithm, hence the name parametric method.

It turns out after the calculation that the weight vector of this method consists of many infinite weights due to division by zero. Recall from section 1 that importance sampling works under the assumption that the support of P_{te} is contained in the support of P_{tr}. This notion prevent from getting the division by zero, because if the assumption is met, the denominator of the ratio will guarantee to have a non-zero value. This method clearly violates the assumption, therefore the weight vector will not be used in the experiment.

2.1.2 Kernel Density Estimation

Kernel Density Estimation (KDE) method directly estimate the P_{te} and P_{tr} from the data, instead of assuming that P_{te} and P_{tr} have certain distribution. The kernel density estimator is given by

$$\hat{p}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{10}$$

where n is the number of sample, $K(\cdot)$ is the kernel, and h is the width of the kernel. I use Gaussian kernel $K(u) = 1/\sqrt{2\pi}\ exp(-\frac{1}{2}u^2)$.

The width of kernel controls the accuracy of the estimation. Too small width leads to a spiky density curve while too large width gives very smooth but inaccurate estimation. I use cross validation to select the most optimum width h. Part of the data is used to fit the density function and the likelihood score of the function is computed using the remaining data. Cross validation gives width $h = 0.48$ for both the training and testing data. In contrast to parametric method, no division by zero is encountered,

suggesting that this method obeys the assumption of P_{te} support contained in the P_{tr} support.

2.1.3 KLIEP

Two previous methods rely on the density estimates of P_{te} and P_{tr}. Although the dataset have enough samples for both training and testing, the number of dimension in the data (50) is quite large. It could pose problem since estimating density can be hard and expensive for data with large dimension due to the curse of dimensionality. If the estimation is inaccurate, this could affect the learning performance under covariate shift.

To get over the problem, people come up with the way to estimate directly the ratio $\frac{P_{te}(x)}{P_{tr}(x)}$ without having to estimate the training and testing density separately. One of such method proposed by Sugiyama et al. [4] called Kullback-Leiber Importance Estimation Procedure (KLIEP).

KLIEP models the ratio with a linear model

$$\frac{P_{te}(x)}{P_{tr}(x)} = \hat{w}(x) = \sum_{l=1}^{t} \theta_l \phi_l(x) \quad \theta_l, \phi_l(x) \geq 0 \tag{11}$$

where $\phi_l(x)$ is a basis function which can be anything, but the authors suggest Gaussian kernel as basis.

Test density is approximated by

$$\hat{P}_{te}(x) = \hat{w}(x) P_{tr}(x) \tag{12}$$

The goal of KLIEP is to learn $\{\theta_l\}_{l=1}^{t}$ so that the Kullback-Leibler (KL) distance between $\hat{P}_{te}(x)$ and $P_{te}(x)$ is minimized. The KL distance is given by

$$KL[\hat{P}_{te}(x)||\hat{w}(x)P_{tr}(x)] = \int P_{te}(x) log \frac{P_{te}(x)}{P_{tr}(x)} dx - \int P_{te}(x) log \hat{w}(x) dx \tag{13}$$

The first term in equation 13 is ignored since it contains no $\hat{w}(x)$. The optimization focus on the second term, and thus the objective function becomes

$$\max_{\{\theta_l\}_{l=1}^{t}} \int P_{te}(x) log \hat{w}(x) dx \tag{14}$$

It is shown by the authors that above function is concave, so global maximum is guaranteed.

KLIEP has one parameter,the width of gaussian kernel h as the basis function in equation 11. The parameter is chosen using cross validation. The $h = 5$ parameter is the optimum according to CV.

2.2 Learning Algorithm

After ratio calculation, the next step is to incorporate those weight vector into the learning algorithm. Equation 7 shows the general training framework which will be interpreted differently by the learning algorithm. In this assignment I choose Support Vector Machine (SVM) as the classifier of choice, because embedding the importance sampling ratio is straightforward. Huang et al. [5] derives the modified version Support Vector Machine (SVM) for covariate shift. The optimization objective function of SVM becomes

$$\min_{\theta,\xi} \frac{1}{2}||\theta||^2 + C\sum_{i}^{n} \beta_i \xi_i \tag{15}$$

where β equals to the ratio

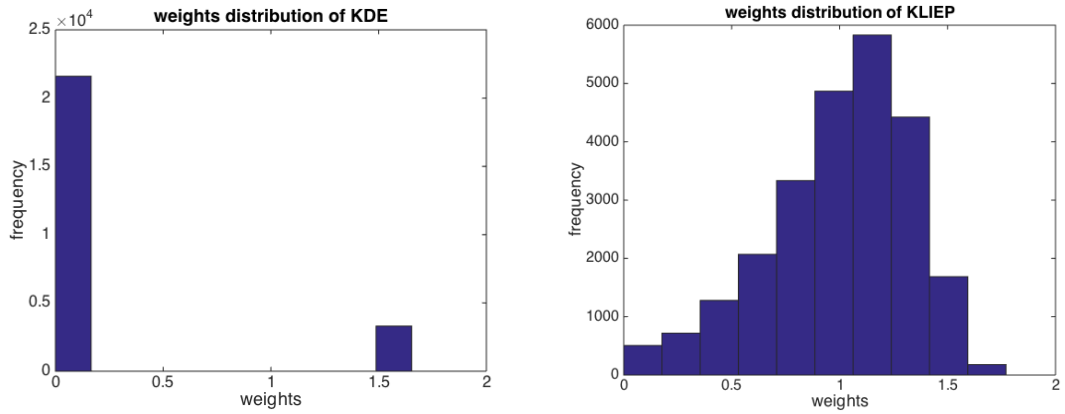
$$\beta = \frac{P_{te}(x)}{P_{tr}(x)} \tag{16}$$

which means that modified SVM only differs from the normal SVM where the slack variables ξ are multiplied by the weights. The usual optimization procedure and kernel tricks for SVM works as normal.

3 Experimental Result

In previous section I discuss the implementation detail of the project. To recap, I start by computing the importance sampling ratio $\frac{P_{te}(x)}{P_{tr}(x)}$ using KDE and KLIEP methods. Both method result in weight vector with size of 24907. After that, the weight vector is used in the weighted SVM classifier to train the training data. Lastly, the testing data is classified by the trained SVM, uploaded to Kaggle, in which the performance results are used for evaluation in the experiment.

Figure 1 shows the histogram of the weights for both KDE and KLIEP. On one hand, KLIEP weights is more nicely distributed than KDE but majority of the training samples gets around 1 weights, which means less penalty or reward are given to the training samples. On the other hand, the KDE weights heavily penalize most of the training samples.



(a) KDE

(b) KLIEP

Figure 1: The distribution of the weights given to training samples for both KDE and KLIEP. KLIEP weights are more spread out while KDE penalizes most of the training samples

The two methods give different characteristics in the weights. Comparison between the weights performance is done empirically. Table 1 shows the classification performance of SVM (RBF kernel, $\gamma = 0.02$, $C = 1$) classifier on the test set. The SVM trained on the training data weighted with both KDE and KLIEP. The result shows that KLIEP gives better performance than KDE and significantly better than the performance of non-weighted fisher classifier. Interestingly, both weighted methods give slightly worse result when compared to the same SVM trained on the non-weighted training samples.

Next step in the evaluation is to train the weighted SVM across several kernels and hyperparameters. Based on the result in table 1, I only train using weight from KLIEP method because it performs better than KDE. Figure 2 shows the results. The result reveals that the SVM with kernel RBF and parameters $C = 100$ and $\gamma = 0.002$ gives the highest performance among other parameters, which is 0.85563. Please note that the evaluated parameters are not exhaustive as I don't explore many possibility due to

Table 1: Performance of the classifiers trained with both weighted and unweighted on the test set

Classifier and Importance Sampling Method	Performance
Fisher, no weights	0.71245
SVM (RBF kernel, $\gamma = 0.02$, $C = 1$), no weights	0.85569
SVM (RBF kernel, $\gamma = 0.02$, $C = 1$), KDE	0.84852
SVM (RBF kernel, $\gamma = 0.02$, $C = 1$), KLIEP	0.85483

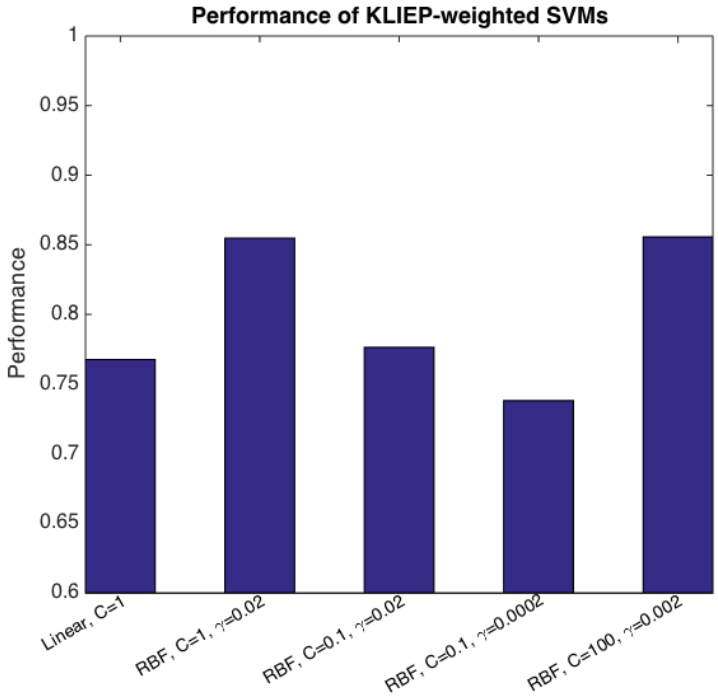


Figure 2: Performance of KLIEP-weighted SVM across several parameters and kernel

upload limitation in Kaggle.

4 Conclusion

The result in section 3 demonstrates that the implemented method, the estimation of importance sampling ratio and embedding into learning algorithm, is not yet optimum. Using the weight ratio does not show clear advantage over the non-weighted one. Over different SVM hyperparameters, the classification results on the test set variate but it cannot pass the performance barrier around 85%.

The reason for above observation, I conclude, is more due to the importance sampling method, not the learning algorithm. The current implemented importance sampling method still cannot capture the 'true' shift between training and testing sample. I believe that there exists the weights that correctly compensate the shift between training and testing. The future work should examines closely several importance estimation methods.

References

[1] Byron P. Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(23):577 – 584, 2005.

[2] 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.

[3] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[4] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V. Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1433–1440. Curran Associates, Inc., 2008.

[5] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.