IN4320 Machine Learning Exercise: Covariate Shift

Yadong Li(4608283)

May 7, 2017

1 Questions

1.1

Here is a real-life covariate shift problem. Suppose that you are asked to build a machine learning system which can identify credit card fraud. In the beginning, all the training examples you have is based on transaction details of credit card owners in the Netherlands. You spend a lot of time training this system and it works well. Now a German company wants to adopt your system to identify credit card fraud in Germany. And now this problem can be treated as covariate shift.
This system is trained on transaction details of owners in the Netherlands, hence the source domain is transaction details in the Netherlands. Since the system will be tested on test data in Germany, the target domain is transaction details in Germany. Because there are some differences between credit card owners in the Netherlands and in Germany, for example, preference of buying product, transaction time period and so on, the target domain and source domain are different.
Although the source domain and target domain are different, we have reason to believe that the class-posterior distributions are the same for both the training and test data. It is safe to assume that credit card fraud is universal. A fraud behavior in the Netherlands will have a very large chance to be detected as a fraud in another country, such as Germany.

1.2

Using a ratio of probability distributions as importance weights is problematic. The weights are defined as $p_{\mathcal{T}}(x_i)/p_S(x_i)$. They introduce a problem that source samples which lie closest to the target distribution receive large weight, while a large number of other samples receive small weights. If a large number of samples receive small weights and only a few of them have large weights, the heavily skewed weights will effectively reduce the sample size of the training set and the classifier trained will be highly variable, and often pathological.

1.3

There is a hidden property corresponding to $p_{\mathcal{T}}(x)$:

$$1 = \int_{\Omega} p_{\mathcal{T}}(x)dx = \int_{\Omega} \frac{p_{\mathcal{T}}(x)}{p_S(x)} p_S(x)dx = \int_{\Omega} w(x)p_S(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} w(x_i)$$

$$1 \approx \frac{1}{n}\sum_{i=1}^{n} w(x_i)$$

Thus, the sample average of the weights is constrained to be close to 1. Without this constraint, the sum of the weights might become arbitrarily small or large.

2 Code Assignment

Result

Hyper parameters setting: $\lambda = 5, \epsilon = 0.01$.
Figure 1 shows the histogram of the values of the importance weights that I found. Figure 2 shows a visualization of both the unweighted classifier and the weighted classifier on both the source and the target domain. The implementation details will be included in the appendix.

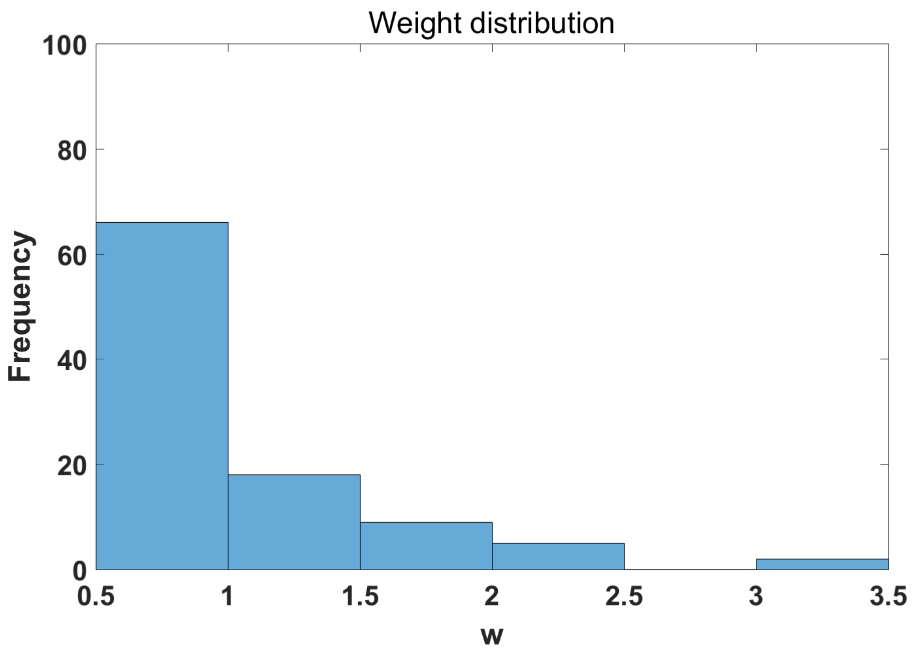

Figure 1: Histogram of the values of the importance weights

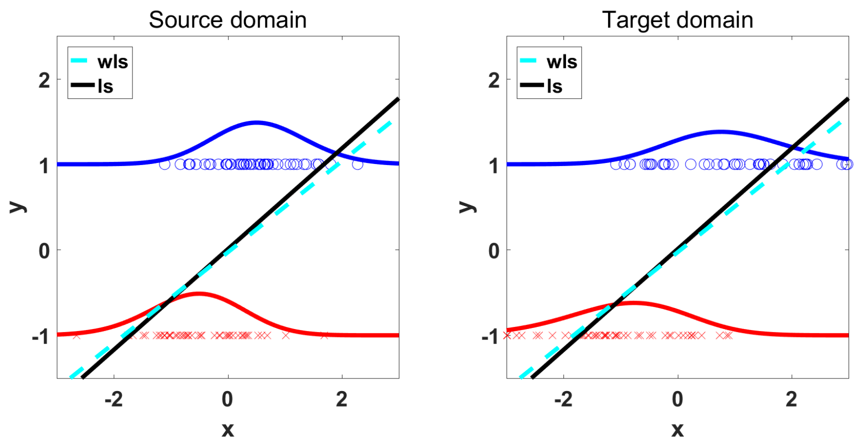

Figure 2: A visualization of both the unweighted classifier and the weighted classifier on both the source and the target domain.

Implementation for Kernel Mean Matching

```
% set hyperparameters
lambda = 5;
epsilon = 1e-2;

% set parameters for quadprog solver
H = kernel(X,X) + lambda * eye(n);
f = - n / m * sum(kernel(X,Z),2);

% set the constraint
A = 1/n * [ones(1,n); - ones(1,n)];
b = [epsilon + 1; epsilon - 1];

% quadprog solver
[w,FVAL] = quadprog(H,f,A,b,[],[],zeros(n,1),[],[],options);
```

Implementation for kernel function

```
function [ out ] = kernel( X1,X2 )
% kernel: apply the kernel to X1, X2

% get helpful variables
n = size(X1,1);
m = size(X2,1);
% initialize output
out = zeros(n,m);
% calculate kernel output
for i = 1:n
    out(i,:) = exp(-(X1(i) - X2).^2 / 2);
end

end
```