Chapter 4

To weight or not to weight

As shown in the previous chapter, it is desirable to use the dual distribution to sample the training data for a learning problem. Nevertheless, since in the supervised learning scenario it is not possible to sample from the dual distribution, or even from the test distribution in the covariate shift case, then it becomes necessary to use weights to match the training distribution to the dual (so far in the literature weighting has been used to match training to test distributions). However, when applied in practice, weighting has a mixed record and sometimes worsens the out-of-sample performance, as discussed in [25]. This raises three natural questions:

- What makes weighting work in some cases but not others?
- Is there a way to predict when it will work and when it won't?
- How accurate is the prediction when applied to real data?

In this chapter, we answer these questions. We also introduce Targeted Weighting, a novel algorithm that predicts when weighting will be beneficial. When applied to various real datasets, the algorithm achieved near-unanimous success.

4.1 What makes weighting work sometimes only?

As stated in Chapter 1, assume we have two distributions P and P' on the input space $\mathcal{X}$, where P is where training data is drawn from, and P' could be the distribution where the test data is drawn from, or some other desired distribution like the dual distribution. Let the target be $f : \mathcal{X} \to \mathcal{Y}$ which is unknown. If we are interested in finding the expected value of a loss function $\ell(g(x), f(x))$, $x \in \mathcal{X}$ is the input variable, and $g : \mathcal{X} \to \mathcal{Y}$ the hypothesis output by the learning algorithm, a standard

approach is to consider the empirical loss on a dataset $R = \{(x_i, y_i)\}_{i=1}^N$, so that we solve the problem

$$g = \arg\min_h J(h, R) = \arg\min_h \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), f(x_i)) \tag{4.1}$$

By minimizing the empirical loss, we are approximating $E_{x\sim P}[\ell(h(x), f(x))]$ with the in-sample quantity $J(h, R)$. If weights are used to match P to P', assuming that we knew both distributions and that $P(x) > 0$, we can use $w_i = P'(x_i)/P(x_i)$ such that

$$E_{x_i\sim P}[w_i \ell(h(x_i), f(x_i)] = E_{x_i\sim P}\left[\frac{P'(x_i)}{P(x_i)} \ell(h(x_i), f(x_i))\right] = E_{x_i\sim P'}[\ell(h(x_i), f(x_i))]. \tag{4.2}$$

Therefore, the use of weights allows simulating the expected value with respect to the desired distribution P'. When $P' = P_S$, we solve the mismatch problem and obtain an unbiased estimate of the loss [61]. When $P = P_R^\star$, we hope to obtain the benefits of training with the dual distribution, and therefore improve the out-of-sample performance. However, there is a side effect of using weights. From statistics, we know that the use of weights leads to an effective sample loss. We now develop an approximate expression for this sample loss based on the change of variance in the sample estimate. We also verify this expression empirically.

4.1.1 The effective sample size

In practice, the loss in Equation 4.2 is estimated empirically using a finite sample. There will be a change in the variance between the unweighted and weighted estimates, and that change is tantamount to an effective loss in sample size. Consider

$$Var\left[\frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), f(x_i))\right] = \frac{1}{N^2} \sum_{i=1}^{N} Var[\ell(h(x_i), f(x_i))] = \frac{1}{N} Var[\ell(h(x_i), f(x_i))]. \tag{4.3}$$

The variance of the estimate is reduced by N, the size of the sample. However, assume that the weights were independent of x, that is, the set $\{w_i\}_{i=1}^N$ is a set of constant weights assigned to each of the training samples. Then, the variance becomes:

$$Var\left[\frac{\sum_{i=1}^N w_i \ell(h(x_i), f(x_i))}{\sum_{i=1}^N w_i}\right] = Var[\ell(h(x_i), f(x_i)] \frac{\sum_{i=1}^N w_i^2}{\left(\sum_{i=1}^N w_i\right)^2} \tag{4.4}$$

Hence, by introducing weights, the sample size has been effectively reduced to

$$N_{\text{eff}} = \frac{\left(\sum_i w_i\right)^2}{\sum_i w_i^2}, \tag{4.5}$$

Table 4.1: RMS error using: N_{eff} training examples (R_1); using weights found with a matching algorithm but assigned randomly (R_2); and using the weights found with matching assigned correctly (R_3)

N_{eff}	Reduced training set RMS error (R_1)	Random weights RMS error (R_2)	Matched Weights RMS error (R_3)	$100 \times \lvert R_1 - R_2 \rvert / R_1$
99,072,095	0.947405	0.947407	**0.946639**	0.0002%
99,070,481	0.947425	0.947322	**0.946531**	0.011%
98,959,437	0.947487	0.947218	**0.946462**	0.028%
98,138,979	0.947823	0.946958	**0.946344**	0.091%
97,416,899	0.947937	0.947159	**0.946452**	0.082%
88,128,713	0.952993	0.951354	0.947886	0.172%
35,484,865	0.999641	0.995888	0.986235	0.375%
25,925,094	1.002576	1.002970	0.989099	0.039%
6,632,779	1.054571	1.055235	1.034925	0.063%

which is maximized when all weights are equal, making $N_{\text{eff}} = N$. This result is not exact, as in practice, the weights are a function of x, for example, if $w(x) = P'(x)/P(x)$.

Nevertheless, this measure of the effective sample size can be verified in a real dataset such as the Netflix dataset. For this set, we computed weights with the Soft Matching algorithm that is introduced in Chapter 5. To test the pure effect of weights on effective sample size reduction, without the matching effect itself coming into play, we assigned the weights *randomly* to the training set consisting of 9.91×10^7 training points. We then tested the error on an out-of-sample set consisting of 2.82×10^6 points.

The out-of-sample error obtained when weights are assigned randomly on the full training set, was compared to the out-of-sample error when weights are not used, but only N_{eff} random training points are used. N_{eff} is computed with Equation 4.5 given the set of weights. To create different instances of N_{eff}, the maximum size of the random weights as well as the matching scheme were changed in different trials. The results are summarized in Table 4.1, which shows averages over 30 runs. The table shows that the RMS errors obtained by reducing the sample size (R_1), or by using random assignment of the weights (R_2), follow each other very closely as expected, with an average difference of less than 0.1%.

If we repeat the same experiment, except that the weights computed are no longer assigned randomly but instead assigned in the order given by the matching algorithm, we obtain the RMS errors shown in the fourth column of the table ($R3$). This column captures both the sample loss and the positive effect of matching. If no weights are used, the RMS error obtained is 0.94664 and it is clear that matching leads *sometimes* to lower RMS errors with respect to this value (these cases are highlighted in the table). Yet, as expected, it always leads to lower RMS error compared to the random assignment of the same weights, thus verifying the favorable matching effect. As N_{eff} decreases, the

benefit of matching is overwhelmed by the reduction in sample size and becomes a net loss.

This notion of sample size loss was first discussed in [61], where without proof, the effective sample size was defined in entropy terms as $N_e = \exp(-\sum_{i=1}^{N} p_i \log p_i)$, where $p_i = w(x_i)/\sum_i w(x_i)$, and $w(x) = p_S(x)/p_R(x)$. In [38], the authors introduce the same expression for N_{eff} that we use, assuming $\sum_i w_i = N$, as they provide a bound for learning using the Kernel Mean Matching (KMM) method, in which N is replaced by the quantity $N_{\text{eff}} = N^2/\|w\|^2$, where the weights are found through their method.

A closer look shows that the effective sample size does not explain fully the negative effect of matching. There is a broader effect that is caused by the difference between sampling directly from a distribution and weighting points to make them look as if they were sampled from a different distribution. In the next subsection, we analyze the expected value and variance of all the moments of a weighted sample, in order to establish the difference between weighting and sampling.

4.1.2 Weighting vs sampling

We can compare analytically the difference between a sample from P' and a weighted sample from P, by looking at the expected value and variance, with respect to the data set generation, of the moments of the sample. Notice that a probability distribution is uniquely determined by the moment generating function. Recall that the moment generating function of a random variable X is given by

$$M_X(t) = \mathbb{E}[e^{Xt}] = 1 + \frac{\mathbb{E}[X]t}{1!} + \frac{(\mathbb{E}[X^2]t)^2}{2!} + \cdots . \tag{4.6}$$

Hence, if the moments are found, the moment generating function can be constructed and so the distribution can also be uniquely determined. Since the moments of an underlying probability distribution can be estimated through the moments of the sample, we compare the expected value and variance of the moments of the weighted sample coming from P and those of a an unweighted sample coming from P', with weights $w(x) = p'(x)/p(x)$. The differences we find will indicate the difference of the distribution that a weighted sample simulates and the actual distribution we match to.

Let $R = \{x_i\}_{i=1}^{N}$ be a set with points sampled from P'. The expected value of the k'th moment of the sample is given by

$$\mathbb{E}_{x_i \sim P'}\left[\frac{\sum_i x_i^k}{N}\right] = \frac{1}{N}\sum \mathbb{E}_{x\sim P'}[x_i^k] = \mathbb{E}_{x\sim P'}[x_i^k], \tag{4.7}$$

and the variance is given by

$$\begin{aligned}
\mathbb{V}ar_{x_i\sim P'}\left[\frac{\sum_i x_i^k}{N}\right] &= \frac{1}{N^2}\mathbb{E}_{x\sim P'}\left[\sum_i x_i^{2k} + \sum_{i\neq j} x_i^k x_j^k\right] - \mathbb{E}_{x\sim P'}[x_i^k] \\
&= \frac{1}{N}\mathbb{E}_{x_i\sim P'}[x_i^{2k}] + \frac{N(N-1)}{N^2}\mathbb{E}_{x_i\sim P'}[x_i^k]^2 - \mathbb{E}_{x_i\sim P'}[x_i^k]^2 \\
&= \frac{1}{N}(\mathbb{E}_{x_i\sim P'}[x_i^{2k}] - \mathbb{E}_{x_i\sim P'}[x_i^k]^2) \\
&= \frac{1}{N}\mathbb{V}ar_{x_i\sim P'}[x_i^k] \qquad (4.8)
\end{aligned}$$

Now assume the points x_i are sampled from P and we use importance weights. The expected value of the k-th moment is given by

$$\mathbb{E}_{x_i\sim P}\left[\frac{\sum_i w(x_i)x_i^k}{N}\right] = \frac{1}{N}\sum_i \mathbb{E}_{x_i\sim P}\left[w(x_i)x_i^k\right] = \frac{1}{N}\sum_i \mathbb{E}_{x_i\sim P'}\left[x_i^k\right] = \mathbb{E}_{x\sim P'}[x_i^k], \qquad (4.9)$$

where we used the fact that

$$\mathbb{E}_{x\sim P}[w(x)f(x)] = \int \frac{p'(x)}{p(x)} f(x)p(x)dx = \int f(x)p'(x)dx = \mathbb{E}_{x\sim P'}[f(x)] \qquad (4.10)$$

Hence, it is clear that the expected value of the moments is the same for a sample distributed as P' as for a sample distributed as P' if we use importance $w(x) = p'(x)/p(x)$. However, the variance of the moments does change:

$$\begin{aligned}
\mathbb{V}ar_{x_i\sim P}\left[\frac{\sum_i w(x_i)x_i^k}{N}\right] &= \frac{1}{N^2}\mathbb{E}_{x_i\sim P}\left[\sum_i w(x_i)^2 x_i^{2k} + \sum_{i\neq j} w(x_i)w(x_j)x_i^{2k}x_j^{2k}\right] - \mathbb{E}_{x\sim P'}[x_i^k]^2 \\
&= \frac{1}{N^2}(\sum_i \mathbb{E}_{x_i\sim P'}[w(x_i)x_i^{2k}] + \sum_{i\neq j}\mathbb{E}_{x_i\sim P'}[x_i^k]\mathbb{E}_{x_j\sim P'}[x_j^k]) - \mathbb{E}_{x\sim P'}[x_i^k]^2 \\
&= \frac{1}{N}(\mathbb{E}_{x\sim P'}[x_i^{2k}] - \mathbb{E}_{x\sim P'}[x_i^k]^2 + \mathbb{E}_{x\sim P'}[w(x_i)x_i^2] - \mathbb{E}_{x\sim P'}[x_i^{2k}]) \\
&= \frac{1}{N}\mathbb{V}ar_{x_i\sim P'}[x_i^k] + \frac{1}{N}(\mathbb{E}_{x\sim P'}[w(x_i)x_i^2] - \mathbb{E}_{x\sim P'}[x_i^{2k}]) \\
&= \frac{1}{N}\mathbb{V}ar_{x_i\sim P'}[x_i^k] + \frac{1}{N}\int (p'(x) - p(x))x^{2k}\frac{p'(x)}{p(x)}dx, \qquad (4.11)
\end{aligned}$$

where we use Equation 4.10 to obtain the final expression. Notice that we end up with an additional term which resembles a distance between p' and p. For the case $k = 0$, we denote the additional term by $D(p'||p)$, where

$$D(p||q) = \int (p(x) - q(x))\frac{p(x)}{q(x)} = \mathbb{E}_{x\sim P}\left[\frac{p(x)}{q(x)}\right] - 1. \qquad (4.12)$$

We first notice that this quantity is indeed a divergence, as it is non-negative and is 0 if and only if $p = q$. To show this, we first prove that

$$\mathbb{E}_{x\sim P}\left[\frac{p(x)}{q(x)}\right] \geq 1. \tag{4.13}$$

Notice that

$$\mathbb{E}_{x\sim P}\left[\frac{q(x)}{p(x)}\right] = \int \frac{q(x)}{p(x)} p(x)dx = 1. \tag{4.14}$$

Using Jensen's inequality, and the function $f(x) = 1/x$ which is convex for $x > 0$, we have

$$\mathbb{E}_{x\sim P}\left[\frac{p(x)}{q(x)}\right] = \mathbb{E}_{x\sim Q}\left[f\left(\frac{q(x)}{p(x)}\right)\right] \geq f\left(\mathbb{E}_{x\sim Q}\left[\frac{q(x)}{p(x)}\right]\right) = 1. \tag{4.15}$$

Notice also that Jensen's inequality holds with equality only when the random variable is a constant. In this case, this implies $p(x)/q(x)$ is constant. Since both numerator and denominator integrate to 1, then they must be equal. Hence

$$D(p||q) = \int (p(x) - q(x))\frac{p(x)}{q(x)} \geq 0 \tag{4.16}$$

with equality if and only if $p = q$.

In fact, this divergence falls in the class of f-divergences [29]. Let $\mathbb{R}^+$ be the set of non-negative real numbers, that is $\mathbb{R}^+ = \{x : x \in \mathbb{R}, x \geq 0\}$, and let $\mathbb{R}^{++}$ be the set of positive real numbers, that is $\mathbb{R}^{++} = \{x : x \in \mathbb{R}, x > 0\}$. These divergences are defined as $D : \mathcal{G}^{++} \times \mathcal{G}^{++} \to \mathbb{R}$, where $\mathcal{G}^{++} = \{g : \mathbb{R}^d \to \mathbb{R}^{++}\}$ and

$$D_f(p||q) = \int f\left(\frac{dQ}{dP}\right) dP = \int f\left(\frac{q(x)}{p(x)}\right) p(x)dx, \tag{4.17}$$

where f is a convex function $f : \mathbb{R} \to \mathbb{R}^+$ that satisfies $f(1) = 0$. In our case,

$$f(u) = 1/u - 1.$$

Notice that f is only convex in the set $\mathbb{R}^{++}$, which agrees with the domain of D, since D is defined only when both p and q are strictly positive. A more common f-Divergence is the KL-divergence which uses $f(u) = \log 1/u$. Notice that this is again a convex function although undefined at $u = 0$ as in our case.

This notion of distance between the distributions P and P' characterizes how the variance of the moments of the samples changes. The "further" the two distributions are, the larger the difference in this variance. An intuitive consequence of this effect, is that the support of the initial set must overlap

significantly with the support of the distribution we want to match to. Take an extreme example; assume in a matching scenario we want to match data sampled from a uniform distribution $U[0, 1]$ to a distribution given by the Gaussian distribution $\mathcal{N}(2, 0.1^2)$. The shift is so extreme that $P(x) = 0$ for x in the domain where the dual distribution is concentrated, so that the ratio $w(x) = p'(x)/p(x)$ is undefined. In practice, such scenario is uncommon as we expect that P', which is either the test distribution P_S or the dual $P_R^\star$, is close to the training distribution. In this case, we would like to think of the sample as being "diverse" enough to be able to match it to the desired distribution.

Now, the term we found that changes the variance of the moments of our weighed sample is only non-negative when we think of the 0-th moment. However, for the first moment, it is very easy to see that this term can be negative. For example, it is negative if p is a very narrow Gaussian distribution, while q is a much wider Gaussian distribution, both having the same mean. This makes the negative terms dominate, as can easily be verified numerically. Hence, the variance of the moments can actually be reduced. Such reduction would lead to lower out-of-sample error, when we consider that our random variables x_i represent the loss evaluated at the different points in the dataset. This coincides with the examples of dual distributions shown in Figure 3.2, where the dual is a slightly wider distribution than the test distribution.

We run a simple simulation to illustrate the difference between learning from a sample distributed as P', and learning from a sample distributed as P but using importance weighting $w(x) = p'(x)/p(x)$. The problem consists of a polynomial regression problem, with a model including second-order Legendre polynomials, while the target includes third-order Legendre polynomials. The constant δ corresponds to the coefficient of the deterministic noise term. Figure 4.1 summarizes the results, and we plot the out-of-sample error obtained both weighting and sampling directly, as the sample size grows. The different plots in the grid vary the KL-divergence between P' and P, with the KL-divergence growing vertically down. Horizontally, the plots change the value of σ_C, the amount of deterministic noise.

As it can be seen, as the KL-divergence grows, training with weighted samples always leads to a larger out-of-sample error. If however, the KL-divergence is small, the difference in errors is almost zero. For a medium KL-divergence, having a large number of samples diminishes the difference between both scenarios. This scenario is the most likely in practice. Notice that this effect is persistent regardless of how complex the target function is, as it is clear that the same behavior can be observed for the different values of deterministic noise.

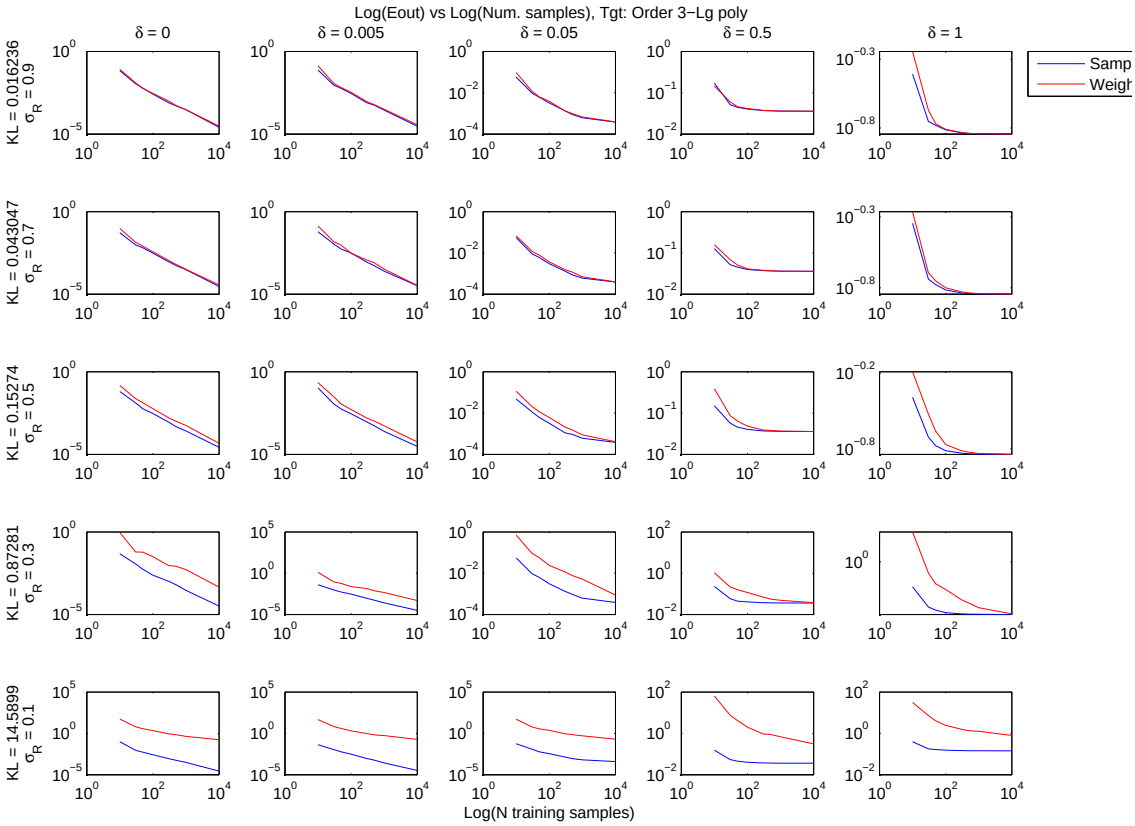


Figure 4.1: Simulations illustrating the out-of-sample error difference vs the number of training samples, when sampling data form a distribution P' and matching a sample distributed as P using importance weights $w(x) = p'(x)/p(x)$. The figure illustrates the effect as P is further from P' in KL-divergence sense (vertical), as well as the how the effect changes as deterministic noise changes (horizontal)

4.1.3 Decomposition of the weighting effect

As discussed above, using weights for matching can both help and hurt learning. We can decompose the effect of weighting into two terms. There is an ideal gain that can be achieved by training with the desired distribution (either the test or the dual distribution). Yet, since we are not able to sample directly from the desired distribution, but instead weight the points to try and achieve this, there is a difference in the moments of the distribution of a weighted versus a directly sampled dataset.

Let us introduce some notation in order to quantify these terms. Fix the training set size to N points. Let g be the hypothesis output by the learning algorithm when training with points from the original training distribution P. Now, we introduce variations of g according to the training conditions. Let $g_{P'}$ be the hypothesis obtained if we had training data distributed according to P'. Let g_w be the hypothesis obtained if the set of weights w has been used during training on points from the original distribution P. As usual, let f be the target function we are trying to learn.

The bottom line in quantifying the value of using weights is the difference in out-of-sample error with weights and without weights:

$$NetGain = \mathbb{E}_{x\sim P_S}[\ell((g(x), f(x))] - \mathbb{E}_{x\sim P_S}[\ell(g_w(x), f(x))] \tag{4.18}$$

Notice that the expected value of the out-of-sample error is taken with respect to the test distribution P_S, and that P' is not necessarily P_S, for example, we may choose $P' = P_R^\star$.

The positive effect of training with the desired distribution can be quantified as

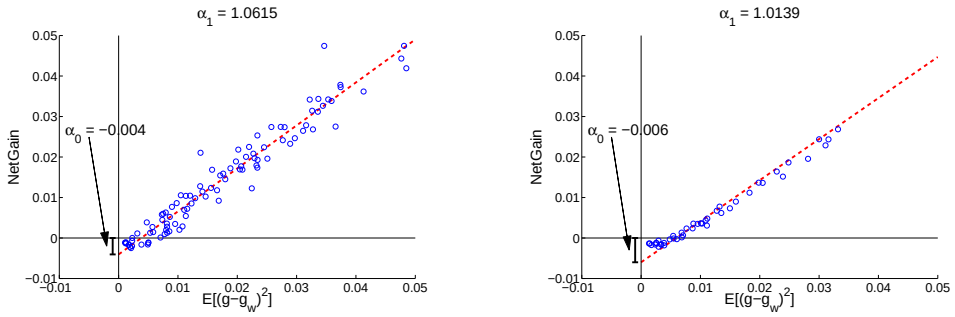
$$MatchGain = \mathbb{E}_{x\sim P_S}[\ell(g(x), f(x))] - \mathbb{E}_{x\sim P_S}[\ell(g_{P'}(x), f(x))] \tag{4.19}$$

This term corresponds to the out-of-sample gain if we had trained with points coming from P' rather than from P. As shown in Chapter 3, this quantity is maximized if P' is the dual distribution. The quantities in the right hand side cannot be directly evaluated in practice, as the setup assumes the training set is distributed according to P. Furthermore, $g_{P'}$ is unavailable, if it were, there would be no need to perform matching to the desired distribution.

Now, we have to quantify the effect of using weights rather than sampling directly. This term is given by

$$WeightLoss = \mathbb{E}_{x\sim P_S}[\ell(g_w(x), f(x))] - \mathbb{E}_{x\sim P_S}[\ell(g_{P'}(x), f(x))]. \tag{4.20}$$

Again, none of the two quantities in the right hand side can be evaluated in a practical setting as $x \sim P$.



(a) $P(x_1) = \mathcal{N}(0,1)$, $f(x_1, x_2)$: 2-D Legendre polynomials

(b) $P(x_1) = \mathcal{N}(0,1)$ $f(x_1, x_2)$: 2-D Sinusoidals

Figure 4.2: $NetGain = \alpha_0 + \alpha_1 \mathbb{E}_{x \sim P'}[(g(x) - g_w(x))^2]$

Using the terms defined, it is clear that:

$$NetGain = MatchGain - WeightLoss. \tag{4.21}$$

Therefore, the balance between the $MatchGain$ and $WeightLoss$ must be found in order to determine if there is a gain in matching P to P'. For example, it may be the case that even if P is different from P', the $MatchGain$ is so negligible that the $WeightLoss$ dominates, and matching is therefore not useful.

4.2 The algorithm: Targeted matching

The $NetGain$ is defined in terms of g, g_w, and f. It cannot be computed using the available data. Not only is the target f unknown, which is normally circumvented by validating using the labeled points available, but also the labeled points are distributed according to P, not P_S. Only unlabeled points may be distributed according to P_S, so if we validate using the labeled points, all the expected values would be computed with respect to the wrong distribution.

One way around that is to use IWCV [65], in which the labeled points are scaled by p_S/p using the same solution as the one shown in Equation 4.2. However, this assumes that P and P_S are known. Since they are not known, they need to be estimated from the samples. If we use validation to get estimates of P and P_S, this will compromise the ability of the same data to provide an unbiased estimate of the quantity we are after that depends on P and P_S. If we do not use validation, however, and the weights we get deviate from $w(x_i) = P'(x_i)/P(x_i)$, as they do in methods like KMM or KLIEP, the weighted expected value will not be a good estimate for the expected value with respect

to P_S. Indeed, as pointed out in [67], the use of validation for KMM and other weighting methods is an open research question.

An alternative approach, instead of estimating P and P_S, is to directly estimate $NetGain$ by calculating the actual weights for subsets of the data using the method at hand, be it KMM or some other method. We can then use these subsets as validation sets that capture the behavior of the weighting method. The problem with this approach is that these methods estimate the weights collectively. The algorithms return a weight for each point in the training set, but the weights returned depend on all the remaining points. Arbitrarily dividing the dataset into validation and training will change the actual weight that the algorithm intends to give to each point. Added to these issues, using a weighted estimate will also suffer an increase in the variance, by the same argument followed in Section 4.1.1, reducing the effective size of the validation set.

Hence, there is no straightforward way of validating the effect of a particular set of weights. Our contribution in this section is to provide an alternative to validation that enables us to estimate $NetGain$ in order to decide whether weighting would be beneficial or not. Our method is based on the following observation. The quantity $\mathbb{E}_{x\sim P_S}[\ell(g(x), g_w(x))]$ can always be estimated since we have unlabeled points $x \sim P_S$. Notice that if the loss ℓ is an appropriate metric, such as the squared loss, then it follows the triangle inequality. In its reverse form, it implies

$$|\mathbb{E}_{x\sim P_S}[\ell(g(x), f(x))] - \mathbb{E}_{x\sim P_S}[\ell(g_w(x), f(x))]| \leq \mathbb{E}_{x\sim P_S}[\ell(g(x), g_w(x))]. \tag{4.22}$$

Hence,

$$|NetGain| \leq \mathbb{E}_{x\sim P_S}[\ell(g(x), g_w(x))] \tag{4.23}$$

Therefore, we propose using the following regression model to estimate the $MatchGain$:

$$NetGain = \alpha_0 + \alpha_1 \mathbb{E}_{x\sim P'}[\ell(g(x), g_w(x))] \tag{4.24}$$

In fact, we can simply set $\alpha_1 = 1$, so that $\alpha_0 \in [-2\mathbb{E}_{x\sim P'}[\ell(g(x), g_w(x)], 0]$.

We first verify the validity of this model in an example with synthetic data. We generate random target functions by picking random coefficients for two dimensional Legendre polynomials, and we carry out learning by using a squared loss function and polynomial features for a non-linear transformation. We use $P' = P_S$ for simplicity, and plot the quantity $\mathbb{E}_{x\sim P_S}[\ell(g(x), g_w(x)]$ versus the $MatchGain$. Figure 4.2 shows typical regressions for the above model. We observe that $\alpha_1 \approx 1$, and that α_0 varies according to P, and P', but does not change much for f's that are somewhat similar, as the model of Equation 4.24 explains well the data points. Notice that it is precisely the quantity α_0 that can make $NetGain$ negative, as the second term is always positive for $\alpha_1 = 1$. The key

Algorithm 3 Targeted Weighting algorithm,

Input: $R = \{x_i, y_i\}_{i=1}^{N},\ S = \{x_i\}_{i=1}^{N_S},,\ w \in \mathbb{R}^N, R \sim P, S \sim P'$
Learn g and g_w using R and w
Compute $e = \frac{1}{N_S}\sum_{i \in S}(\ell(g(x_i), g_w(x_i)) \approx \mathbb{E}_{x \sim P'}[\ell(g(x), g(w))]$
Set $Count = 0$
for $k \leftarrow\ 1 : K$ **do**
  Generate surrogate target f_k
  $R^{(k)} = \{x_i, f_k(x_i)\}_{i=1}^{N}$
  Learn g_k and g_{kw} using $R^{(k)}$ and w
  Compute $e_1 = \frac{1}{N_S}\sum_{i \in S}(\ell(g_k(x_i), f_k(x_i)) \approx \mathbb{E}_{x \sim P_S}[\ell(g_k(x), f_k(x))]$
  Compute $e_2 = \frac{1}{N_S}\sum_{i \in S}(\ell(g_{kw}(x_i), f_k(x_i)) \approx \mathbb{E}_{x \sim P_S}[\ell(g_{kw}(x), f_k(x))]$
  Compute $e_3 = \frac{1}{N_S}\sum_{i \in S}(\ell(g_k(x_i), g_{kw}(x_i)) \approx \mathbb{E}_{x \sim P_S}[\ell(g_k(x), g_{kw}(x))]$
  $\alpha_{k0} = e_1 - e_2 - e_3$
  if $\alpha_{k0} + e > 0$ **then**
    $Count++$
  end if
end for
if Count $> K/2$ **then**
  Use weights
else
  Do not use weights
end if

observation is that the problem reduces to estimating α_0.

We propose the following procedure to estimate the sign of $NetGain$. Since the simulations on synthetic data show α_0 changes with every P and P', but is fairly constant for f's that have comparable complexity and numerical range, we can try to estimate α_0 using surrogate target functions that are similar to the actual target in these two aspects. To match the complexity, we generate the surrogate functions using a parametric model and adjust the number of parameters and level of added noise to make the error when learning surrogates about the same as the error when learning the actual target. To match the numerical range, we normalize the outputs of the surrogates according to the range of values of the target.

The algorithm, which we call Targeted Weighting (TW), is described in Algorithm 3. The idea is to find α_0 for K surrogate functions. Let f_k, $k = 1, \ldots, K$, be surrogate target functions. Let g_k be the hypotheses learned when training the algorithm with points sampled from P. Let g_{kw} be the resulting hypotheses when training with the set of weights w. The idea is to find the values α_{0k} for each of these surrogates. According to the model of Equation 4.24, this value is given by

$$\alpha_{0k} = MatchGain_k - \mathbb{E}_{x \sim P_S}[\ell(g_k, g_{kw}] \tag{4.25}$$

$$= \mathbb{E}_{x \sim P_S}[\ell(g_k(x), f_k(x)] - \mathbb{E}_{x \sim P_S}[\ell(g_{kw}(x), f_k(x))] - \mathbb{E}_{x \sim P_S}[\ell(g_k, g_{kw})]. \tag{4.26}$$

We then compute a proxy for the $NetGain$ for each of the surrogate functions:

$$NetGain \approx \alpha_{0k} + \mathbb{E}_{x\sim P_S}[\ell(g(x), g_w(x))]. \tag{4.27}$$

With the K proxies for $NetGain$, we decide if weighting is beneficial or not by taking a majority vote on the sign of the proxies obtained. We test our procedure on real datasets to verify its validity.

4.3 Experimental results

We applied this algorithm on various real datasets. One of these datasets is the Netflix Prize dataset. The distribution of the training points in this set is quite different from that of the test points used for evaluating the performance of the solutions. After the competition was over, the labels of the test points were made available, which makes it possible to verify if indeed our algorithm improves the ultimate out-of-sample performance. We also applied TW on 17 benchmark datasets that were used in [38] to evaluate the matching effect. The original datasets have $P_R = P_S$, but we artificially introduced a mismatch between P and P_S as was done in that paper. We report the results in the next two subsections.

4.3.1 Results on the Netflix dataset

We ran an elaborate experiment with TW on the Netflix data. In each run, we used $K = 100$ surrogate functions. We generated the surrogate targets using a factorization model similar to SVD, so that the surrogate function is $r_{ij}^{(k)} = \text{round}(\sigma(p_{ik}^T q_{jk} + \mu + \epsilon))$ for user i and movie j, where $p_{ik}, q_{jk} \in \mathbb{R}^\kappa$ are generated randomly, ϵ is added noise, and μ and σ adjust the mean and standard deviation of the labels to those in the Netflix training set. The distributions that generate p_{ik} and q_{jk} were chosen to make the resulting ratings lie with high probability in the normal range $[1, 5]$ of movie ratings. If a particular rating was outside this range, its value was truncated to 1/5. The dimension of the movie and user features, κ, was varied between 10 and 50 for the different surrogate functions, a range compatible with the complexity of the original set.

We chose the training algorithm known as SVD++ [44], implemented with the speedup proposed in [24]. This learning algorithm provided the best solutions during the Netflix competition. We trained on the raw training set whose distribution is different from that of the validation and test sets (known as 'Probe' and 'Qual'). We only used the inputs in 'Probe' without their labels, as well as the provided inputs in 'Qual' which are similarly distributed, in order to perform matching.

Since the training set has about 100 million points, and we matched along 6 coordinates, we needed an efficient matching algorithm for our experiments, which we describe in Chapter 5 and call

Table 4.2: TW with $K = 100$ in the Netflix dataset. *NetGain* is computed with the labels of the original Netflix competition test set.

Coord.	λ	$\%f_k$ with $NetGain > 0$	*NetGain* improvement (basis points)	λ	$\%f_k$ with $NetGain > 0$	*NetGain* improvement (basis points)
t		**0**%	−71		37%	9
dt		**0**%	−14		**58**%	0.1
us	1000	**0**%	−121	100	**60**%	5
ms		**0**%	−147		**28**%	−5
ut		**0**%	−41		**0**%	−20
un		**0**%	−40		**0**%	−17
t		**55**%	7		**60**%	2
dt		**64**%	0.1		**77**%	0.1
us	50	**51**%	9	10	**69**%	3
ms		**5**%	−0.3		**54**%	0.4
ut		**0**%	−14		**0**%	−5
un		**0**%	−13		**0**%	−5
t		**65**%	1		**76**%	0.3
dt		**84**%	0.1		**83**%	0.01
us	5	**71**%	2	1	**87**%	0.4
ms		**61**%	0.2		**76**%	0.01
ut		**0**%	−2		**0**%	−0.01
un		**0**%	−2		**0**%	−0.01
t, us, ms	100, 100, 10	**60**%	10	100, 50, 10	**60**%	15

Soft Matching. Each point in the dataset consists only of a user id, a movie id and the time of the rating, $R = \{u_i, m_i, t_i\}_{i=1}^N$. We represented each point by 6 characteristic coordinates that represent the mismatch between the training and test points of this dataset. The coordinates are the absolute time of rating (t); time since first rating of movie (dt); number of ratings per user or 'user support' (us); number of ratings for a movie or 'movie support' (ms); time since first rating of user (ut); and order of rating among the user's ratings (un). The distribution along these coordinates was distinctly different between the training and test sets. Soft Matching matches the distributions along these coordinates by binning the values along each coordinates and minimizing the discrepancy between corresponding bins while maximizing N_{eff}. A parameter λ controls the extent of matching, where $\lambda = 0$ implies all weights are 1, and $\lambda = \infty$ yields importance weighting. The details of this algorithm will be explained further in Chapter 5.

Table 4.2 shows different choices for λ and the coordinate to be matched. It also shows what percentage of surrogate functions had $\alpha_{k0} + \mathbb{E}[\ell(g(x), g_w(x))] > 0$ (the '$\%f_k$ with $NetGain > 0$' column), and then the actual improvement or worsening of out-of-sample error in the test set. Notice that in the results shown, if the weighting mechanism worsens results, it is always the case that less than 50% of the surrogate functions yield improvement. On the other hand, only for one case of the weighting mechanism parameters, there is a positive *NetGain* while less than 50% of the surrogate

Table 4.3: Out-of-sample performance when not using weights, always using weights, and using TW to decide if weights should be used. The number of points in the training set (N_R) and the test set (N_S) are shown for reference. After sampling, only $Nsub_R$ points were used for training

Dataset (Classif.)	N_R	Avg. $Nsub_R$	N_S	Test error		
				No weights	With weights	With TW
Br. Cancer(1)	546	173	137	0.055 ± 0.0006	0.055 ± 0.0006	$\mathbf{0.053 \pm 0.0006}$
Br. Cancer(2)	546	109	137	0.317 ± 0.003	0.313 ± 0.003	$\mathbf{0.300 \pm 0.004}$
Br. Cancer(3)	546	208	137	0.0681 ± 0.0007	0.0651 ± 0.0007	$\mathbf{0.0650 \pm 0.0007}$
Br. Cancer	546	228	137	0.045 ± 0.0005	0.046 ± 0.0005	$\mathbf{0.044 \pm 0.0005}$
German cred.	614	216	154	0.298 ± 0.0009	0.298 ± 0.0009	$\mathbf{0.296 \pm 0.0009}$
Haberman	245	97	61	0.257 ± 0.0016	0.259 ± 0.0016	$\mathbf{0.248 \pm 0.0016}$
India diabetes	614	217	154	0.269 ± 0.0011	0.271 ± 0.0011	$\mathbf{0.262 \pm 0.0011}$
Ionosphere	281	153	70	0.066 ± 0.0009	0.067 ± 0.0009	$\mathbf{0.061 \pm 0.0009}$
USPS 3vs9	1042	443	260	0.5031 ± 0.0009	0.4896 ± 0.0008	$\mathbf{0.4883 \pm 0.0008}$
(Regression)				NMSE error		
Abalone	3342	2319	835	0.4850 ± 0.0010	0.4840 ± 0.0010	$\mathbf{0.4828 \pm 0.0010}$
Ailerons	11000	3637	2750	0.1847 ± 0.0002	0.1850 ± 0.0002	$\mathbf{0.1846 \pm 0.0002}$
Bank8FM	6554	3393	1638	0.0657 ± 0.0001	$\mathbf{0.0653 \pm 0.0001}$	$\mathbf{0.0653 \pm 0.0001}$
Bank32nh	6554	3353	1638	0.4733 ± 0.0006	0.4740 ± 0.0006	$\mathbf{0.4731 \pm 0.0006}$
Bos. Housing	405	160	101	0.3197 ± 0.0028	0.3164 ± 0.0029	$\mathbf{0.3061 \pm 0.0026}$
CA Housing	16512	5250	4128	0.3688 ± 0.0004	0.3686 ± 0.0004	$\mathbf{0.3679 \pm 0.0004}$
Cpu-act	6554	6325	1638	0.2767 ± 0.0007	0.2778 ± 0.0010	$\mathbf{0.2719 \pm 0.0008}$
Cpu-small	6554	6331	1638	0.2891 ± 0.0007	0.2881 ± 0.0009	$\mathbf{0.2846 \pm 0.0007}$
Delta Ailerons	5703	5691	1426	0.4585 ± 0.0004	0.4603 ± 0.0004	$\mathbf{0.4580 \pm 0.0004}$
Kin8nm	6554	4109	1638	0.5881 ± 0.0005	0.5881 ± 0.0005	$\mathbf{0.5877 \pm 0.0005}$
Puma8nh	6554	3993	1638	0.632 ± 0.0006	0.632 ± 0.0006	$\mathbf{0.631 \pm 0.0006}$

functions yielded improvement. The table highlights the runs where the majority of the surrogates agreed with the ultimate out-of-sample performance.

4.3.2 Results on further benchmark datasets

We also tested our TW algorithm on UCI classification dataset [7], as well as on LIACC Regression datasets [68]. Since these datasets have $P_R = P_S$, a sampling bias is further introduced as described in the experiments in [38]. We began with detailed experiments in the Breast Cancer dataset where three types of sampling biases are introduced to the training set: bias using a single input feature (1), bias using all features (2), and label-dependent bias (3). We then ran experiments in both classification and regression datasets where sampling bias is introduced along the first PCA component, and we matched along all original coordinates. All biased sampling parameters are set as in [38].

Experiments were run 1,000 times for each type of sampling bias scheme. We used a Gaussian kernel SVM as the learning algorithm for the classification datasets. The SVM package `libsvm` [23] with weights was used to carry out the experiments, and the size of the kernel for each dataset was taken from [38]. For the regression datasets, we used regularized least squares (LS). For each run, a

different split between training and test sets was done, where 20% of the data was saved for testing. The remaining 80% was sub-sampled for each of the sampling bias schemes. TW was run using $K = 100$ surrogate functions. For the classification tasks, the surrogate functions were generated by randomly choosing support vectors and coefficients. The only constraint in this random selection was to use a similar number of support vectors to the ones obtained when training with the original data, in order to have comparable complexity. Specifically, surrogate target k had $sv_k \in [0.9sv, 1.1sv]$, where sv is the number of support vectors obtained with the original data. For the regression tasks, a random parameter vector $\theta_k \sim \mathcal{N}(\theta, (0.1\theta)^2)$ was generated, where θ were the parameters of the LS output on the unweighted training data. The labels for the artificial target were then computed as $y_{ik} = x_i^T \theta_k + \epsilon_i$, and $\epsilon_i \sim \mathcal{N}(0, 0.1^2)$.

Table 4.3 summarizes the expected out-of-sample error obtained when no weights are used, when weights are always used, and when TW is used to decide if weights should be used. For classification tasks, the test classification error is shown, while for regression the normalized mean-squared error ($NMSE = \frac{1}{NVar[y]} \sum_i (y_i - g(x_i))^2$) is shown. As it can be seen in the table, for *every* dataset and *every* sampling scheme used, the use of TW improves performance.

Hence, the unanimous success of Targeted Weighting in these datasets shows that it is a reliable method that can be used to determine if a particular weighting scheme will yield out-of-sample improvements or not. With this algorithm it is possible to answer the second fundamental question we had pose in Chapter 1. Now we move on to answer the final question: how to match a sample coming from distribution P, to a desired distribution P'?