The Matrix Multiplicative Weights Algorithm for Domain Adaptation

David Alvarez Melis

New York University, Courant Institute of Mathematical Sciences

251 Mercer Street

New York, NY 10012

May 2013

A thesis submitted in partial fulfillment
of the requirements for the degree of
master's of science
Department of Mathematics
New York University
May 2013

Advisor: Mehryar Mohri

Abstract

In this thesis we propose an alternative algorithm for the problem of domain adaptation in regression. In the framework of (Mohri and Cortes, 2011), this problem is approached by defining a *discrepancy distance* between source and target distributions and then casting its minimization as a semidefinite programming problem. For this purpose, we adapt the primal-dual algorithm proposed in (Arora et al. 2007) which is a particular instance of their general Multiplicative Weights Algorithm (Arora et al. 2005).

After reviewing some results from semidefinite programming and learning theory, we show how this algorithm can be tailored for the context of domain adaptaiton. We provide details of an explicit implementation, including the ORACLE, which handles dual feasibility. In addition, by exploiting the structure of the matrices involved in the problem we propose an efficient way to carry out the computations required for this algorithm, avoiding storing and operating with full matrices. Finally, we compare the performance of our algorithm with the smooth approximation method proposed by Cortes and Mohri, in both an artificial problem and a real-life adaptation task from natural language processing.

Contents

List of Figures

Chapter 1

Introduction

The typical setting of supervised machine learning consists of inferring rules from labeled *training data* by means of a learning algorithm and then using these to perform a task on new "unseen" data. The performance of the method obtained in this way is evaluated on a separate set of labeled data, called the *test data*, by using the learned rules to predict its labels and comparing these with the true values.

In the classical setting, it is assumed that the method will be used on data arising from the same source as the training examples, so the training and testing data are always assumed to be drawn from the same distribution. The early pioneering results on learning theory, such as Vapnik and Chervonekis's work [32] and the PAC (Probably Approximately Correct) learning model by Valiant [29], are built upon this assumption.

However, it might be the case that the training examples are drawn from some *source* domain that differs from the *target* domain. This *domain adaptation* scenario violates the assumptions of the classical learning models, and thus theoretical estimates of generalization error provided by these no longer hold. Consequently, the standard learning algorithms which are based on this theory are deprived of their performance guarantees when confronted with this scenario.

The framework of domain adaptation, as it turns out, occurs naturally in various applications, particularly in natural language processing, computer vision and speech recognition. In these fields, the reason to train on instances stemming from a domain different from that of interest is often related to availability and cost, such as scarcity of labeled data from the target domain, but wide availability from another similar source domain. For example, one might wish to use a language model on microblogging feeds, but due to abundance of labeled entries, it might be more convenient to train it on journalistic texts, for which there are immense corpora available with various linguistic annotations (such as the famous *Penn Treebank* Wall Street Journal dataset[1]). Naturally, this is an adaptation task, since the language used in

[1]`http://www.cis.upenn.edu/~treebank/`

these two types of writing is significantly discrepant, that is, they can be thought of as being drawn from statistical language models with different underlying distributions.

The problem of adaptation started to draw attention among the Machine Learning community in the early 1990's, particularly in the context of the applications mentioned above (see for example [9] or [13]). In this early stage, most authors presented techniques to deal with domain adaptation that despite achieving varying degrees of success, were mostly context-specific and lacked formal guarantees.

Theoretical analysis of this problem is much more recent, starting with work by Ben-David et al.[4] for the context of classification, in which they provide VC-dimension-based generalization bounds for this case, followed by work by Blitzer et al.[5] and Mansour et al.[18]. In a subsequent paper by the latter [19], the authors introduce a novel distance between distributions, the *discrepancy distance*, which they use to provide domain adaptation generalization bounds for various loss functions. For the case of regression and the L_2 loss, they show that a discrepancy minimization problem can be cast as a semidefinite program.

Building upon the work by Mansour et al.[19] and equipped with the discrepancy distance, Cortes and Mohri [7] revisit domain adaptation in the regression-L_2 setting, providing pointwise loss guarantees and an efficient algorithm for this context, based on Nesterov's smooth approximation techniques. Here we propose an alternative to this algorithm, by making use of the Multiplicative Weights Algorithm [1], recently adapted by Arora and Kale [2] to semidefinite programming problems.

In order to present a coherent articulation between domain adaptation, semidefinite programming and the Multiplicative Weights algorithm, we provide a brief albeit comprehensive review of the main concepts behind these, which occupies the first three chapters. Chapter 5 is devoted to showing how the multiplicative weights algorithm can be tailored to domain adaptation, along with all the practical hindrances that this implies. At the end of that chapter we present our algorithm, provide guarantees for it and compare it to the smooth approximation method used by Cortes and Mohri. In Chapter 6 we present results for two practical experiments: an artificial toy adaptation problem, and a real problem from natural language processing, followed by a concluding section summarizing the main results of this thesis.

The purpose of this work is twofold. It intends to provide the reader with a succinct, consistent overview of three somewhat separated topics (domain adaptation, semidefinite programming and the Multiplicative Weights algorithm) and then showing how these can be brought together in an interesting manner over an elegant - yet artful - optimization problem.

Chapter 2

Domain Adaptation

In this section we formalize the learning scenario of Domain Adaptation, define the notion of discrepancy distance between distributions and use it to derive the optimization problem that sits at the core of this thesis. This problem will then motivate the content of the remaining sections.

2.1 Background

As usual for regression, let us consider two measurable subsets of $\mathbb{R}$, the input and output spaces, which we will denote by $\mathcal{X}$ and $\mathcal{Y}$, respectively. The former contains the explanatory variables and the latter contains response variables, also referred to as *labels*. Thus, a labeled *example* consists of a pair $(x, y) \in (\mathcal{X} \times \mathcal{Y})$. In the standard supervised learning setting, the elements in $\mathcal{X}$ are $\mathcal{Y}$ are assumed to be related by a target labeling function $f : \mathcal{X} \to \mathcal{Y}$, and the usual task consists of estimating this function.

For the domain adaptation framework we define domains by means of probability distributions over $\mathcal{X}$. So, let Q be the distribution for the *source* domain, and P the distribution over $\mathcal{X}$ for the *target* domain. Naturally, the idea is to assume that P and Q are not equal in general. Consequently, their corresponding *labeling* functions f_P and f_Q might differ.

In the problem of regression in domain adaptation the *learner* is given a labeled sample of m points $\mathcal{S} = \big\{(x_1, y_1), \dots, (x_m, y_m)\big\} \in (\mathcal{X} \times \mathcal{Y})^m$, where each x_i is drawn i.i.d according to Q and $y_i = f_Q(x_i)$. We will denote by $\hat{Q}$ the empirical distribution corresponding to $x_1, \dots, x_m$. On the other hand, he is also provided with a set $\mathcal{T}$ of unlabeled test points from the target domain (that is, drawn according to P), with a corresponding empirical distribution $\hat{P}$.

Intuitively, the task of the learner is infer a suitable labelling function which is *similar* to f_P. To set up this task more formally, let us consider a hypothesis set $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\}$ and a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ that is symmetric and convex with respect to each argument. L is frequently taken to be the squared loss (as usual in regression), but can be more general. This leads to the following definition from statistical decision theory.

Definition 2.1.1. *Suppose we have two functions $f, g : \mathcal{X} \to \mathcal{Y}$, a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ and a distribution D over $\mathcal{X}$. The **expected loss** of f and g with respect to L is given by*

$$\mathcal{L}_D(f, g) = E_{x\sim D}[L(f, g)]$$

In light of this, it is clear what the objective of the learner is. He must select a hypothesis $h \in \mathcal{H}$ such that

$$\mathcal{L}_D(h, h') = E_{x\sim D}[L(h(x), h'(x)]$$

That is, the problem consists of minimizing the expected loss of choosing h' to approximate f_P.

By this point we notice the inherent difficulty of this learning task. The learner has no direct information about f_P, but only about f_Q, through the labeled examples in $\mathcal{S}$. A naive strategy would be to select a hypothesis h based solely based on information about f_Q, and hope that P and Q are sufficiently similar. This optimistic approach will not only be devoid of theoretical learning guarantees, but will also most likely fail if the source and target domains are even slightly different.

2.2 Discrepancy Distance

Based on the analysis above, it is clear that the crucial aspect behind domain adaptation is to be able to quantify the disparity between the source and target distributions P and Q. For this purpose, Mansour, Mohri and Rostamizadeh [19] introduce such a notion of similarity, the *discrepancy distance*, which is tailored to adaptation problems and turns out to facilitate several results on learning bounds. It is this very notion of similarity that will prove crucial in the formulation of the optimization problem in Section 2.4.

Definition 2.2.1. *Given a hypothesis set H and loss function $\mathcal{L}$, the* **discrepancy distance** *between two distributions P and Q over X is defined by*

$$\mathrm{disc}(P, Q) = \max_{h,h'\in H} |\mathcal{L}_P(h', h) - \mathcal{L}_Q(h', h)| \tag{2.1}$$

This definition follows naturally from the way we have set up the learner's task above, for it measures the difference in expected losses incurred when choosing a fixed hypothesis h in presence of a target labeling function f_P, over both domains.

Other alternatives to this notion of dissimilarity have been proposed before (the l_1-distance or the d_A distance, for example), but the discrepancy distance[1] is advantageous over those in various ways. First, as pointed out by the authors, the discrepancy can be used to compare distributions for general loss functions. In addition, it can be estimated from finite samples when the set $\{|h' - h| : h', h \in H\}$ has finite VC dimension, and it provides sharper learning bounds that other distances.

Before presenting the first theoretical results concerning the discrepancy distance, we turn our attention to a fundamental concept in learning theory, which will appear in many of the bounds presented later in this section. The *Rademacher Complexity* is a measure of complexity of a family of functions; it captures this richness by assessing the capacity of a hypothesis set to fit random noise. The following two definitions are taken from [20].

Definition 2.2.2. *Let G be a family of functions mapping from Z to $[a, b]$ and $S = (z_1, \ldots, z_M)$ a fixed sample of size m with elements in Z. Then, the* **Empirical Rademacher Complexity** *of G with respect to the sample S is defined as*

$$\hat{\mathfrak{R}}_S(G) = \underset{\boldsymbol{\sigma}}{E}\left[\sup_{g \in G} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(z_i)\right]$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)^T$, with σ_i's independent uniform random variables taking values in $\{-1, +1\}$.

Definition 2.2.3. *Let D denote the distribution according to which samples are drawn. For any integer $m \geq 1$, the* **Rademacher Complexity** *of G is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to D:*

$$\mathfrak{R}(G) = \underset{S \sim D^m}{E}[\mathfrak{R}_S(G)]$$

The Rademacher complexity is a very useful tool when trying to find generalization bounds. In such cases, one often tries to bound the generalization error (or risk) of choosing a hypothesis $h \in H$, when trying to learn a concept $c \in C$. Definition 2.2.4 formalizes this idea.

Definition 2.2.4. *Given a hypothesis $h \in H$, a target concept $c \in C$, and an underlying distribution D, the* ***generalization error*** *of h is defined by*

$$R(h) = P_{x \sim D}[h(x) \neq c(x)] = E_{x \sim D}[1_{h(x) \neq c(x)}]$$

Additionally, given a sample $S = \{x_1, \ldots, x_m\}$, the ***empirical error*** *is given by*

$$\hat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} 1_{h(x_i) \neq c(x_i)}$$

In both cases, 1_ω is the indicator function of the event ω.

[1]Note that despite its name, the discrepancy does not in general define a distance or metric in the mathematical way, for it is possible that $\operatorname{disc}(P, Q) = 0$ for $P \neq Q$, making it a pseudometric instead. Partly because of simplicity, and partly because this will not be the case for a large family of hypothesis sets, we shall nevertheless refer to it as a distance.

The following theorem provides a general bound for the expected risk $R(h)$ over samples of fixed size. It provides the structure that most data-dependent generalization bounds based on the Rademacher complexity have: it bounds the risk by a term containing the empirical risk, a term containing the empirical Rademacher complexity and a term that decays as the square root of the sample size.

Theorem 2.2.5. *Let H be a class of functions mapping $Z = X \times Y$ to $[0,1]$ and $\mathbb{S} = \{z_1, \dots, z_m\}$ a finite sample drawn i.i.d according to a distribution Q. Then, for any $\delta > 0$, with probability at least $1-\delta$ over samples $\mathbb{S}$ of size m, the following holds*

$$R(h) \leq \hat{R}(h) + \hat{\mathfrak{R}}(H) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \tag{2.2}$$

Proof. See [3] for a detailed proof of this theorem. □

With the definitions given above, and the very general bound given by Theorem 2.2.5, we are now ready to prove two fundamental results from [19] about the discrepancy distance. These results, at a high level, show how this notion of distance between distributions does indeed exhibit useful properties and offers solid guarantees, given in terms of Rademacher complexities. This first of these is a natural result that we would expect of such a notion of distance. It shows that as sample size increases, the discrepancy between a distribution and its empirical counterpart decreases.

Theorem 2.2.6. *Suppose that the loss function L is bounded by $M > 0$. Let Q be a distribution over X and $\hat{Q}$ its empirical distribution for a sample $\mathcal{S} = (x_1, \dots, x_m)$. Then, for any $\delta > 0$, with probability at least $1-\delta$ over samples $\mathcal{S}$ of size m drawn according to Q the following bound holds*

$$\operatorname{disc}(Q, \hat{Q}) \leq \hat{\mathfrak{R}}_{\mathcal{S}}(L_H) + 3M\sqrt{\frac{\log \frac{2}{\delta}}{2m}} \tag{2.3}$$

where L_H is the class of functions $L_H = \{x \mapsto L(h'(x), h(x)) : h, h' \in H\}$.

Proof. Let us first scale the loss L to $[0,1]$ to adapt it to Theorem 2.2.5. For this, we divide by M, and define the new class L_H/M, for which such theorem asserts that for any $\delta > 0$, with probability at least $1-\delta$, the following inequality holds for all $h, h' \in H$:

$$\frac{\mathcal{L}_Q(h', h)}{M} \leq \frac{\mathcal{L}_{\hat{Q}}(h', h)}{M} + \hat{\mathfrak{R}}(L_H/M) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

But the empirical Rademacher complexity has the property that $\hat{\mathfrak{R}}(\alpha H) = \alpha \hat{\mathfrak{R}}(H)$ for any hypothesis set H and scalar α [3]. In view of this, the inequality above becomes

$$\frac{\mathcal{L}_Q(h', h)}{M} - \frac{\mathcal{L}_{\hat{Q}}(h', h)}{M} \leq \frac{1}{M}\hat{\mathfrak{R}}(L_H) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}$$

which, multiplying by M both sides and noting that the left-hand side is lower-bounded by $\operatorname{disc}(Q, \hat{Q})$, yields the desired result. □

As a direct consequence of this result we obtain the following corollary, which is nonetheless much more revealing: it shows that for L_q regression loss functions the discrepancy distance can be estimated from samples of finite size.

Corollary 2.2.7. *With the same hypotheses as in Theorem 2.2.6, assume in addition that $L_q(h, h') = |h - h'|^q$ and that P is another distribution over X with corresponding empirical distribution $\hat{P}$ for a sample $\mathcal{T}$. Then, for any $\delta > 0$*

$$\mathrm{disc}_{L_q}(P, Q) \leq \mathrm{disc}_{L_q}(\hat{P}, \hat{Q}) + 4q\big(\hat{\mathfrak{R}}_{\mathcal{S}}(H) + \hat{\mathfrak{R}}_{\mathcal{T}}(H)\big) + 3M\left(\sqrt{\frac{\log\frac{2}{\delta}}{2m}} + \sqrt{\frac{\log\frac{2}{\delta}}{2n}}\right)$$

with probability at least $1 - \delta$ over samples $\mathcal{S}$ of size m drawn according to Q and samples $\mathcal{T}$ of size n drawn according to P.

Proof. First, we will prove that for an L_q loss, the following inequality holds

$$\hat{\mathfrak{R}}(L_H) \leq 4q\hat{\mathfrak{R}}(H) \tag{2.4}$$

For this, we note that for such kind of loss function, the class L_H is given by $L_H = \{x \mapsto |h'(x) - h(x)|^q : h, h' \in H\}$, and since the function $f : x \mapsto x^q$ is q-Lipschitz for x in the unit interval, we can use Talagrand's contraction lemma to bound $\hat{\mathfrak{R}}(L_H)$ by $2q\hat{\mathfrak{R}}(H')$, with $H' = \{x \mapsto (h'(x) - h(x)) : h, h' \in H\}$. Thus, we have

$$\begin{aligned}\hat{\mathfrak{R}}(L_H) \leq 2q\hat{\mathfrak{R}}(H') &= 2E_\sigma\left[\sup_{h,h'} \frac{1}{m}\Big|\sum_{i=1}^{m} \sigma_i(h(x_i) - h'(x))\Big|\right] \\ &\leq 2E_\sigma\left[\sup_{h} \frac{1}{m}\Big|\sum_{i=1}^{m} \sigma_i(h(x_i)\Big|\right] + 2E_\sigma\left[\sup_{h'} \frac{1}{m}\Big|\sum_{i=1}^{m} \sigma_i(h'(x_i)\Big|\right] = 4\hat{\mathfrak{R}}_{\mathcal{S}}(H)\end{aligned}$$

which proves (2.4). Now, using the triangle inequality we obtain

$$\mathrm{disc}_{L_q}(P, Q) \leq \mathrm{disc}_{L_q}(P, \hat{P}) + \mathrm{disc}_{L_q}(\hat{P}, \hat{Q}) + \mathrm{disc}_{L_q}(Q, \hat{Q})$$

and applying Theorem (2.2.6) on the first and third terms in the right-hand side

$$\begin{aligned}\mathrm{disc}_{L_q}(P, Q) &\leq \mathrm{disc}_{L_q}(\hat{P}, \hat{Q}) + \big(\hat{\mathfrak{R}}_{\mathcal{S}}(L_H) + \hat{\mathfrak{R}}_{\mathcal{T}}(L_H)\big) + 3M\left(\sqrt{\frac{\log\frac{4}{\delta}}{2m}} + \sqrt{\frac{\log\frac{4}{\delta}}{2n}}\right) \\ &\leq \mathrm{disc}_{L_q}(\hat{P}, \hat{Q}) + 4q\big(\hat{\mathfrak{R}}_{\mathcal{S}}(H) + \hat{\mathfrak{R}}_{\mathcal{T}}(H)\big) + 3M\left(\sqrt{\frac{\log\frac{4}{\delta}}{2m}} + \sqrt{\frac{\log\frac{4}{\delta}}{2n}}\right)\end{aligned}$$

where in the last step we used (2.4). This completes the proof. □

The two results above are of utmost importance. Without a theoretical guarantee that we can actually estimate the discrepancy distance between two distributions empirically, it would be futile to adopt it, for this abstract metric is unknown to us in most applications, and thus any results based on it would be uninformative.

2.3 Generalization bounds

In this section we present *generalization bounds* that involve the discrepancy distance. These type of bounds, crucial in machine learning theory, bound the error (or average loss) of a fixed hypothesis independently of the sample used to obtain it, usually in terms of empirical or computable quantities. Herein lies the importance of these results; they provide *a priori* guarantees of success in learning tasks.

Let h^*_Q and h^*_P be the best in-class hypotheses for the source and target labeling functions, that is, $h^*_Q = \arg\min_{h\in H} \mathcal{L}_Q(h, f_Q)$ and similarliy for f_P. The following theorem bounds the expected loss of any hypothesis in terms of these minimum losses and the discrepancy distance between P and Q.

Theorem 2.3.1. *Assume that the loss function L is symmetric and obeys the triangle inequality. Then, for any hypothesis $h \in H$, the following holds*

$$\mathcal{L}(h, f_P) \leq \mathcal{L}_P(h^*_P, f_P) + \mathcal{L}_Q(h, h^*_Q) + \operatorname{disc}(P, Q) + L_P(h^*_Q, h^*_P)$$

Proof. For a fixed hypothesis $h \in H$, we have

$$\begin{aligned}
\mathcal{L}_P(h, f_P) &\leq \mathcal{L}_P(h, h^*_Q) + \mathcal{L}_P(h^*_Q, h^*_P) + \mathcal{L}_P(h^*_P, f_P) && \text{(triangle inequality)}\\
&= \big(\mathcal{L}_P(h, h^*_Q) - \mathcal{L}_Q(h, h^*_Q)\big) + \mathcal{L}_Q(h, h^*_Q) + \mathcal{L}_P(h^*_Q, h^*_P) + \mathcal{L}_P(h^*_P, f_P)\\
&\leq \operatorname{disc}(P, Q) + \mathcal{L}_Q(h, h^*_Q) + L_P(h^*_Q, h^*_P) + \mathcal{L}_P(h^*_P, f_P) && \text{(by Definition 2.2.1)}
\end{aligned}$$

□

Clearly, the result given by Theorem 2.3.1 is far more general than needed for our purpose, but it is included here to show that this analysis can be carried out at a very high level. Now, however, let us head towards the context of our interest: adaptation in a regularized regression setting. For this, we will assume in the following that H is a subset of the reproducing kernel Hilbert space (RKHS) associated to a symmetric positive definite kernel K, namely $H = \{h \in \mathbf{H} : \|h\|_K \leq \Lambda\}$, were $\|\cdot\|_K$ is the norm induced by the inner product given by K, and $\Lambda \geq 0$. Suppose also that the Kernel is bounded: $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$.

There are various algorithms that deal with the problem of regression. A large class of them, the so-called *kernel-based regularization* algorithms, seek to minimize the empirical error of the hypothesis by introducing a magnitude-penalizing term. A typical objective function for one of these methods has the form

$$F_{(\hat{Q}, f_Q)}(h) = \hat{R}_{(\hat{Q}, f_Q)}(h) + \lambda \|h\|^2_K \tag{2.5}$$

where $\lambda > 0$ is the regularization parameter and $\hat{R}_{(\hat{Q}, f_Q)}(h) = \frac{1}{m}\sum_{i=1}^m L(h(x_i), f_Q(x_i))$. Algorithms as diverse as support vector machines, support vector regression and kernel ridge regression (KRR) fall in this category.

Next, we define a very desirable property of loss functions, which is a weaker version of multivariate Lipschitz continuity.

Definition 2.3.2. *The loss function L is μ-**admissible** for $\mu > 0$ if it is convex with respect to its first argument and for all $x \in \mathcal{X}$ and $y, y' \in \mathcal{Y}$ it satisfies the following Lipschitz conditions*

$$|L(h'(x), y) - L(h(x), y)| \leq \mu |h'(x) - h(x)|$$

$$|L(h(x), y') - L(h(x), y)| \leq \mu |y' - y|$$

As mentioned before, the labeling functions f_P and f_Q may not coincide on $\text{supp}(\hat{Q})$ or $\text{supp}(\hat{P})$. However, as pointed out in [8], they must not be too different in order for adaptation to be possible, so we can safely assume that the quantity

$$\eta_H(f_P, f_Q) = \inf_{h \in H} \left\{ \max_{x \in \text{supp}(\hat{P})} |f_P(x) - h(x)| + \max_{x \in \text{supp}(\hat{P})} |f_P(x) - h(x)| \right\}$$

is small.

For this large family of kernel-based regularization algorithms, Cortes and Mohri [7] provide the following guarantee.

Theorem 2.3.3. *Let L be a μ-admissible loss. Suppose h' is the hypothesis returned by the kernel-based regularization algorithm* (2.5) *when minimizing $F(\hat{P}, f_P)$ and h is the one returned when minimizing $F_{(\hat{Q}, f_Q)}$. Then,*

$$\left| L(h'(x), y) - L(h(x), y) \right| \leq \mu r \sqrt{\frac{\text{disc}(\hat{P}, \hat{Q}) + \mu \eta_H(f_P, f_Q)}{\lambda}} \tag{2.6}$$

for all $x \in \mathcal{X}$ and all $y \in \mathcal{Y}$.

Proof. (Given in Appendix A). □

The bound (2.6) reveals the depedency of the generalization error on the discrepancy distance. This is particularly true when $\eta_h \approx 0$, in which case the bound is dominated by the square root of $\text{disc}(\hat{P}, \hat{Q})$. This is a first sign that suggests the discrepancy is the appropriate measure of dissimilarity for this context.

Again, proceeding from general to particular, we will now consider the specific case of Kernel Ridge Regression (KRR), which is an instance of the kernel-based regularization algorithm family. This is the method we implement in later chapters, and thus it is of our interest to obtain a guarantee tailored to it.

For this purpose, we will make use of another measure of the difference between the source and target labeling functions, given by

$$\delta_H(f_P, f_Q) = \inf_{h \in H} \left\| \underset{x \sim \hat{P}}{E} [\Delta(h, f_P)(x)] - \underset{x \sim \hat{Q}}{E} [\Delta(h, f_Q)(x)] \right\|_K \tag{2.7}$$

where $\Delta(f,g) = \big(h(x) - f(x)\big)\Phi_K(x)$, with Φ_K a feature vector associated with K. The reader will readily note how this definition has the same flavor as the discrepancy distance defined before, especially if $f_P = f_Q$. It is also easy to see that $\delta_H(f_P, f_Q)$ is a finer measure than the function $\eta_H(f_P, f_Q)$ defined before. This will allow for a tighter generalization bound.

As Cortes and Mohri note, the term $\delta_H(f_P, f_Q)$ vanishes in various scenarios. The simplest of these cases is naturally when the source and target labelling functions coincide in $\operatorname{supp}(\hat{Q})$. Although this might not be true in general, for adaptation to be possible it is again reasonable to assume $\delta_H(f_P, f_Q)$ is small [19]. We are now ready to present the main learning guarantee for KRR.

Theorem 2.3.4. *Let L be the squared loss and assume that for all $(x,y) \in \mathcal{X} \times \mathcal{Y}$, $L(h(x), y) \leq M$ and $K(x,x) \leq r^2$ for some $M > 0$ and $r > 0$. Let h' be the hypothesis returned by KRR when minimizing $F(\hat{P}, f_P)$ and h the one returned when minimizing $F(\hat{Q}, f_Q)$. Then, for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$,*

$$\big|L(h'(x), y) - L(h(x), y)\big| \leq \frac{r\sqrt{M}}{\lambda}\left(\delta_H(f_P, f_Q) + \sqrt{\delta_H^2(f_P, f_Q) + 4\lambda \operatorname{disc}(\hat{P}, \hat{Q})}\right) \tag{2.8}$$

Proof. (Given in Appendix A). □

The bound (2.8) is again expected to be dominated by the discrepancy term. Furthermore, as we mentioned before, in many scenarios the term $\delta_H(f_P, f_Q)$ completely vanishes, yielding a much simpler bound

$$\big|L(h'(x), y) - L(h(x), y)\big| \leq 2r\sqrt{\frac{M \operatorname{disc}(\hat{P}, \hat{Q})}{\lambda}} \tag{2.9}$$

In either case, we see that the loss we incur in by minimizing the objective function over the source instead of the target domain depends almost exclusively on the discrepancy between these. This direct dependency of the bound on the discrepancy distance confirms that this is *the* right measure of dissimilarity between source and target distributions.

Furthermore, the bounds (2.8) and (2.9) suggest a strategy to minimize the loss of selecting a certain hypothesis h. If we were able to choose an empirical distribution $\hat{Q}^*$ that minimizes the discrepancy distance with respect to $\hat{P}$, and then use it for the regularization based algorithm, we would obtain a better guarantee. Note, however, that the training sample is given, and thus we do not have control over the support of $\hat{Q}$. Thus, our search would be restricted to distributions with a support included in that of $\hat{Q}$. This leads to our main optimization problem, the details of which will be presented in the next section.

2.4 Optimization Problem

To formalize the optimization problem to be solved, let $\mathcal{X}$ be a subset of $\mathbb{R}^N$, with $N > 1$. We denote by $S_Q = \operatorname{supp}(\hat{Q})$, and $S_P = \operatorname{supp}(\hat{P})$, two sets with $|S_Q| = \mathfrak{m} \leq m$ and $|S_P| = \mathfrak{n} \leq n$.

Their unique elements are $\mathbf{x}_1, \ldots \mathbf{x}_{\mathfrak{m}}$ and $\mathbf{x}_{\mathfrak{m}+1}, \ldots, \mathbf{x}_{\mathfrak{q}}$ respectively, with $\mathfrak{q} = \mathfrak{m} + \mathfrak{n}$.

As noticed in the previous section, the theoretical learning guarantees suggest a strategy of selecting the distribution q^* with support $\mathrm{supp}(\mathcal{Q})$ that minimizes $\mathrm{disc}(\hat{P}, \hat{Q})$. That is, if $\mathcal{Q}$ denotes the set of distributions with support $\mathrm{supp}(\hat{Q})$, we are looking for

$$q^* = \operatorname*{arg\,min}_{q \in \mathcal{Q}} \mathrm{disc}_L(\hat{P}, q)$$

which, using the definition of discrepancy distance (2.2.1) and the squared loss, becomes

$$\begin{aligned} \hat{Q}' &= \operatorname*{arg\,min}_{\hat{Q}' \in \mathcal{Q}} \max_{h,h' \in H} |\mathcal{L}_P(h', h) - \mathcal{L}_Q(h', h)| \\ &= \operatorname*{arg\,min}_{\hat{Q}' \in \mathcal{Q}} \max_{h,h' \in H} \left| E_{\hat{P}}[(h'(x) - h(x))^2] - E_{\hat{Q}'}[(h'(x) - h(x))^2] \right| \end{aligned}$$

Now, since in linear regression we seek an N-dimensional parameter vector, the hypothesis space can be described as the set $H = \{\mathbf{x} \mapsto \mathbf{w}^T\mathbf{x} : \|\mathbf{w}\| \leq 1\}$, so that the problem becomes

$$\begin{aligned} &\min_{\hat{Q}' \in \mathcal{Q}} \max_{\|\mathbf{w}\| \leq 1, \|\mathbf{w}'\| \leq 1} \left| E_{\hat{P}}\left[((\mathbf{w}'(\mathbf{x}) - \mathbf{w}(\mathbf{x}))^T\mathbf{x})^2\right] - E_{\hat{Q}'}\left[((\mathbf{w}'(\mathbf{x}) - \mathbf{w}(\mathbf{x}))^T\mathbf{x})^2\right] \right| \\ &= \min_{\hat{Q}' \in \mathcal{Q}} \max_{\|\mathbf{w}\| \leq 1, \|\mathbf{w}'\| \leq 1} \left| \sum_{\mathbf{x} \in S} (\hat{P}(\mathbf{x}) - \hat{Q}'(\mathbf{x}))[(\mathbf{w}'(\mathbf{x}) - \mathbf{w}(\mathbf{x}))^T\mathbf{x}]^2 \right| \\ &= \min_{\hat{Q}' \in \mathcal{Q}} \max_{\|\mathbf{u}\| \leq 2} \left| \sum_{\mathbf{x} \in S} (\hat{P}(\mathbf{x}) - \hat{Q}'(\mathbf{x}))[\mathbf{u}^T\mathbf{x}]^2 \right| \\ &= \min_{\hat{Q}' \in \mathcal{Q}} \max_{\|\mathbf{u}\| \leq 2} \left| \mathbf{u}^T \left(\sum_{\mathbf{x} \in S} (\hat{P}(\mathbf{x}) - \hat{Q}'(\mathbf{x}))\mathbf{x}\mathbf{x}^T \right) \mathbf{u} \right| \qquad (2.10) \\ & \qquad (2.11) \end{aligned}$$

To simplify this, let us denote by z_i the distribution weight at point $\mathbf{x}_i$, namely $z_i = q^*(\mathbf{x}_i)$. Then, we define the matrix

$$\mathbf{M}(\mathbf{z}) = \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i \mathbf{M}_i \qquad (2.12)$$

where $\mathbf{M}_0 = \sum_{j=\mathfrak{m}+1}^{\mathfrak{q}} \hat{P}(\mathbf{x}_j)\mathbf{x}_j\mathbf{x}_j^T$ and $\mathbf{M}_i = \mathbf{x}_i\mathbf{x}_i^T$ with $\mathbf{x}_i \in S_Q$ for $i = 1, \ldots, \mathfrak{m}$. Using this notation, (2.10) can be expressed as

$$\min_{\substack{\|\mathbf{z}\|_1 = 1 \\ \mathbf{z} \geq 0}} \max_{\|u\| = 1} |\mathbf{u}^T\mathbf{M}(\mathbf{z})\mathbf{u}| \qquad (2.13)$$

But since $\mathbf{M}(\mathbf{z})$ is symmetric, the term inside the absolute value is the Rayleigh quotient of $\mathbf{M}(\mathbf{z})$, so the inner term corresponds to finding the largest absolute eigenvalue of $\mathbf{M}(\mathbf{z})$. And, again, since $\mathbf{M}$ is symmetric, we have

$$|\lambda_{max}(\mathbf{M}(\mathbf{z}))| = \sqrt{\lambda_{max}(\mathbf{M}(\mathbf{z})^2)} = \sqrt{\lambda_{max}(\mathbf{M}(\mathbf{z})^T\mathbf{M}(\mathbf{z}))} = \|\mathbf{M}(\mathbf{z})\|_2$$

To simplify the notation even further, let us define the simplex $\Delta_m = \{\mathbf{z} \in \mathbb{R}^m : z_i \geq 0, \sum_{i=1}^m z_i = 1\}$. With this, we can finally formulate our problem in its definitive form. One way to do this is to express (2.13) as a norm-minimization problem

$$\min_{\mathbf{z}\in\Delta_m} \|\mathbf{M}(\mathbf{z})\|_2 \tag{2.14}$$

This a well known convex optimization problem, and it has been studied extensively (see for example Overton [25]). It is often expressed in an equivalent form as a semidefinite programming (SDP) problem:

$$\begin{aligned} \max_{\mathbf{z},s} \quad & s \\ \text{s.t.} \quad & \begin{bmatrix} sI & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & sI \end{bmatrix} \succeq 0 \\ & \mathbf{1}^T\mathbf{z} = 1 \\ & \mathbf{z} \geq 0 \end{aligned} \tag{2.15}$$

where $\mathbf{A} \succeq 0$ means $\mathbf{A}$ is positive semidefinite and $\mathbf{1}$ denotes a vector of ones. In the next chapter, we will justify why (2.14) and (2.15) are equivalent, in addition to presenting other fundamental properties of SDP problems and the crucial relation between their primal and dual formulations.

Before closing this chapter, we make a brief observation on the structure of the matrix $\mathbf{M}(\mathbf{z})$ and the implications in dimensionality that this has. This will prove crucial in the way the way the algorithm is designed in Chapter 5.

Note that the matrix $\mathbf{M}(\mathbf{z})$ can be written as a product of matrices as follows

$$\mathbf{M}(\mathbf{z}) = \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i\mathbf{M}_i = \sum_{i=\mathfrak{m}+1}^{\mathfrak{q}} P(\mathbf{x}_i)\mathbf{M}_i - \sum_{i=1}^{\mathfrak{m}} z_i\mathbf{M}_i = \mathbf{X}\mathbf{D}\mathbf{X}^T \tag{2.16}$$

where $\mathbf{X} = \left[\mathbf{x}_1|\dots|\mathbf{x}_{\mathfrak{m}}|\mathbf{x}_{\mathfrak{m}+1}|\dots|\mathbf{x}_{\mathfrak{m}+\mathfrak{n}}\right]$ and $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{ii} = z_i$ for $i = 1,\dots,\mathfrak{m}$ and $\mathbf{D}_{ii} = P(\mathbf{x}_i)$ for $i = \mathfrak{m}+1,\dots,\mathfrak{m}+\mathfrak{n}$.

Let us now define the *kernelized* version of $\mathbf{M}(\mathbf{z})$ by

$$\mathbf{M}'(\mathbf{z}) = \mathbf{X}^T\mathbf{X}\mathbf{D} \tag{2.17}$$

This name comes from the fact that frequently in regression the input space is a feature space $\mathcal{F}$ defined implicitly by a kernel $K(\mathbf{x},\mathbf{y})$. Here, the inputs are feature vectors $\Phi(\mathbf{x})$, where $\Phi : \mathcal{X} \to \mathcal{F}$ is a map such that $K(\mathbf{x},\mathbf{y}) = \Phi(\mathbf{x})^T\Phi(\mathbf{y})$. In this case, the problem of domain adaptation can be formulated analogously for the matrix of features $\mathbf{\Phi}$, instead of $\mathbf{X}$. As a result of this, in (2.17) in place of the Gram matrix $\mathbf{X}^T\mathbf{X}$, we obtain $\mathbf{\Phi}^T\mathbf{\Phi} = \mathbf{K}$, the kernel matrix.

As a consequence of their similar structure, the matrices $\mathbf{M}(\mathbf{z})$ and $\mathbf{M}'(\mathbf{z})$, share many properties, which in many cases allows us to work interchangeably with one or the other. The following lemma exhibits one such fundamental shared feature.

Lemma 2.4.1. *If* $\mathbf{v}$ *is an eigenvector with eigenvalue* λ *of* $\mathbf{XDX}^T$*, then* $\mathbf{Xv}$ *is an eigenvector of* $\mathbf{X}^T\mathbf{XD}$ *with the same eigenvalue. Furthermore,* $\mathbf{XDX}^T$ *and* $\mathbf{XX}^T\mathbf{D}$ *have the exact same eigenvalues (up to multiplicity of* $\lambda = 0$*).*

Proof. Let $\Lambda(\mathbf{A})$ denote the set of eigenvalues of the matrix $\mathbf{A}$. Suppose $(\mathbf{X}^T\mathbf{XD})\mathbf{v} = \lambda\mathbf{v}$. Then

$$(\mathbf{XDX}^T)(\mathbf{XDv}) = \mathbf{XD}(\mathbf{X}^T\mathbf{XDv}) = \mathbf{XD}(\lambda\mathbf{v}) = \lambda(\mathbf{XDv})$$

so $\mathbf{DXv}$ is eigenvector of $\mathbf{XDX}^T$ with the same eigenvalue. This means that $\Lambda(\mathbf{X}^T\mathbf{XD}) \subseteq \Lambda(\mathbf{XDX}^T)$. Now to prove the contention in the opposite direction, suppose λ is an eigenvalue (with eigenvector $\mathbf{v}$) of $\mathbf{XDX}^T$, then

$$(\mathbf{X}^T\mathbf{XD})\mathbf{X}^T\mathbf{v} = \mathbf{X}^T(\mathbf{XDX}^T\mathbf{v}) = \lambda\mathbf{X}^T\mathbf{v}$$

If $\mathbf{X}^T\mathbf{v} \neq 0$, then λ must also be an eigenvalue of $\mathbf{X}^T\mathbf{XD}$. If $\mathbf{X}^T\mathbf{v} = 0$, then $\mathbf{XDX}^T\mathbf{v} = 0$ too, so $\lambda = 0$. In any case, $\Lambda(\mathbf{XDX}^T) \subseteq \Lambda(\mathbf{X}^T\mathbf{XD})$. We conclude that $\mathbf{XDX}^T$ and $\mathbf{X}^T\mathbf{XD}$ have the same set of eigenvalues. □

Note, however, that the matrix $\mathbf{M}'$ is not symmetric. This might be inconvenient especially when dealing with eigendecompositions, since in that case eigenvectors corresponding to different eigenvalues will not be in general orthogonal. Thus, it is useful to find another matrix with the same dimensions (and eigenvalues) that is symmetric. For this, we notice that $\mathbf{X}^T\mathbf{X}$ is clearly positive semidefinite, so that it has a (unique) square root. So, by a similar argument as the one used in Lemma (2.4.1), we see that

$$\mathbf{M}'_s(\mathbf{z}) = (\mathbf{X}^T\mathbf{X})^{\frac{1}{2}}\mathbf{D}(\mathbf{X}^T\mathbf{X})^{\frac{1}{2}} \tag{2.18}$$

has the same eigenvalues as $\mathbf{M}(\mathbf{z})$ and $\mathbf{M}'(\mathbf{z})$. Again, this can be generalized for the kernel framework by taking $\mathbf{K}^{\frac{1}{2}}$, instead of the square root of the Gram matrix (recall that the Kernel must be PSD too).

The reader will note that, in general, the dimension of these new matrices $\mathbf{M}'(\mathbf{z})$ and $\mathbf{M}'_s(\mathbf{z})$ is not the same as that of $\mathbf{M}(\mathbf{z})$. While the former have dimension $(\mathfrak{m}+\mathfrak{n}) \times (\mathfrak{m}+\mathfrak{n})$, the later has dimension $N \times N$. This suggests a practical strategy for optimizing computations involving these matrices: if the dimension of the input space N is smaller than the sum of the dimensions of the source and target samples $(\mathfrak{m}+\mathfrak{n})$, it is convenient to work with $\mathbf{M}(\mathbf{z})$. In the opposite case, one can work instead with the kernelized versions $\mathbf{M}'(\mathbf{z})$ or $\mathbf{M}'_s(\mathbf{z})$. Since for the applications of our interest the dimension of the input space tends to be very large, we will work from now on with the latter forms. Which of the kernelized forms we use will depend on the specific properties needed in certain situations. We will explore this in further detail in Chapter 5.

Chapter 3

Semidefinite Programming

Having outlined the theoretical framework of domain adaptation, and having presented the problem of interest as a Semidefinite Programming (SDP) problem (2.15), we now review the main concepts underlying this area of Optimization. We present standard definitions and notation, some basic properties of semidefinite matrices, along with the key aspects of SDPs and their formulation.

3.1 Properties of Semidefinite Matrices

Let us restrict our attention for the moment to symmetric matrices. We will denote by $\mathbb{S}^n$ the set of symmetric $n \times n$ matrices with real entries. Although we will deal with the real case for simplicity, most of the results presented in this section are applicable to matrices with entries in $\mathbb{C}$ too.

The same way in which the inner product between vectors is crucial for defining objective functions in Linear Programming and various other branches of Optimization, a notion of inner product between matrices is needed in the context of semidefinite programming.

Definition 3.1.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be* $n \times m$ *matrices. Their* ***Frobenius inner product*** *is defined as*

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} B_{ij} = \mathbf{Tr}\left(\mathbf{A}^T \mathbf{B}\right) = \mathbf{Tr}\left(\mathbf{A}\mathbf{B}^T\right)$$

and is frequently denoted by $\mathbf{A} \bullet \mathbf{B}$.

It is easy to show that this is indeed an inner product, and that it arises from considering the matrices $\mathbf{A}$ and $\mathbf{B}$ as vectors of length nm and using the standard Euclidean inner product for them. The reader might also recognize in the formulation of (3.1.1) the Frobenius norm for matrices, namely, $\|\mathbf{A}\|_F = \sqrt{\mathbf{Tr}\,(\mathbf{A}^T\mathbf{A})}$, which this inner product induces. Although other definitions of inner products are possible for matrices, the one presented here is the most widely used, and for that reason it is sometimes refered to - as we will do here - simply as *the* inner product for matrices.

The second fundamental concept in semidefinite programming involving the properties of a matrix is the notion of positive-definiteness, which naturally extends the concept of positivity for scalar values.

Definition 3.1.2. *A symmetric $n \times n$ matrix $\mathbf{A}$ is said to be* ***positive-semidefinite*** *if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for every nonzero vector x, and such property is denoted by $\mathbf{A} \succeq 0$. If in addition, $\mathbf{x}^T\mathbf{A}\mathbf{x} = 0$ only for $\mathbf{x} = 0$, then $\mathbf{A}$ is said to be* ***positive-definite****, and we write $\mathbf{A} \succ \mathbf{0}$.*

The subset of $\mathbb{S}^n$ consisting of all positive-semidefinite (PSD) matrices is often denoted by $\mathbb{S}^n_+$, and, although less frequently, the subset of positive-definite is denoted by $\mathbb{S}^n_{++}$. Note that

$$\mathbb{S}^n_+ = \{\mathbf{X} \in \mathbb{S}^n : \mathbf{u}^T\mathbf{X}\mathbf{u} \geq 0 \, \forall \mathbf{u} \in \mathbb{R}^n\} = \bigcap_{\mathbf{u}\in\mathbb{R}^n} \{\mathbf{X} \in \mathbb{S}^n : \mathbf{X} \bullet \mathbf{u}\mathbf{u}^T \geq 0\}$$

And thus, being an intersection of convex and closed sets (half-spaces), then $\mathbb{S}^n_+$ is also closed and convex.

The definition above is extended to negative-definite and negative-semidefinite matrices by reversing the inequalities, or by using the definition on $-\mathbf{A}$. Furthermore, a partial ordering can be defined on $\mathbb{S}^n$ by denoting $\mathbf{A} \succeq \mathbf{B}$ when $\mathbf{A} - \mathbf{B} \succeq \mathbf{0}$.

A simple property linking Definitions 3.1.1 and 3.1.2 is the following.

Proposition 3.1.3. *For any square matrix $\mathbf{A}$ and vector $\mathbf{x}$, $\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{A} \bullet (\mathbf{x}\mathbf{x}^T)$.*

Proof. Suppose $\mathbf{A}$ has dimension $n \times n$, and $\mathbf{x}$ has length n.

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_{i,j}^{n} x_i \mathbf{A}_{ij} x_j = \sum_{i,j}^{n} \mathbf{A}_{ij}(\mathbf{x}\mathbf{x}^T)_{ij} = \mathbf{Tr}\,(\mathbf{A}\mathbf{x}\mathbf{x}^T) = \mathbf{A} \bullet \mathbf{x}\mathbf{x}^T$$

□

As a natural corollary of this, we note that $\mathbf{A}$ is positive-definite if and only if $(\mathbf{x}\mathbf{x}^T) \bullet \mathbf{A} > 0$ for any nonzero vector $\mathbf{x}$.

The following three simple results on positiveness of matrices will prove crucial later on in the context of duality theory for semidefinite programming. They characterize PSD matrices in terms of their interaction, through the inner product, with other matrices. The usefulness of these properties will be revealed particularly through the last results of this section, namely two theorems of the alternative, which build upon these lemmas.

Lemma 3.1.4. *Let $\mathbf{A}$ be a symmetric $n \times n$ matrix. Then, $\mathbf{A}$ is positive-semidefinite if and only if $\mathbf{A} \bullet \mathbf{B} \geq \mathbf{0}$ for every positive-semidefinite matrix $\mathbf{B}$.*

Proof. For the "if" part, suppose $\mathbf{A}$ is not PSD. Then there exists a vector $\mathbf{x}$ such that $\mathbf{x}^T\mathbf{A}\mathbf{x} < 0$. But then for $\mathbf{B} = \mathbf{x}\mathbf{x}^T$ we have

$$\mathbf{A} \bullet \mathbf{B} = \mathbf{A} \bullet (\mathbf{x}\mathbf{x}^T) = \mathbf{x}^T\mathbf{A}\mathbf{x} < 0$$

a contradiction.

For the "only if" part, let $\mathbf{A}$ and $\mathbf{B}$ be in $\mathbb{S}^n_+$. By the Spectral Theorem, $\mathbf{B}$ can be written as $\mathbf{B} = \sum_{i=1}^n \lambda_i \mathbf{v}_i\mathbf{v}_i^T$, where $\lambda_i \geq 0$ for $i = 1, \ldots, n$. Thus

$$\mathbf{A} \bullet \mathbf{B} = \mathbf{Tr}\,(\mathbf{A}\mathbf{B}) = \mathbf{Tr}\,\left(\mathbf{A}\sum_{i=1}^n \lambda_i \mathbf{v}_i\mathbf{v}_i^T\right) = \sum_{i=1}^n \lambda_i \mathbf{Tr}\,(\mathbf{A}\mathbf{v}_i\mathbf{v}_i^T) = \sum_{i=1}^n \lambda_i \mathbf{v}_i^T\mathbf{A}\mathbf{v}_i \geq 0$$

where we have used Proposition 3.1.3 for the last equality. This completes the proof. □

Lemma 3.1.5. *If* $\mathbf{A}$ *is positive-definite then* $\mathbf{A}\bullet\mathbf{B} > 0$ *for every nonzero positive-semidefinite matrix* $\mathbf{B}$.

Proof. Suppose $\mathbf{A}$ is positive-definite. Then, it must be orthogonally diagonalizable, that is, it can be expressed as $\mathbf{A} = \mathbf{P}\mathbf{\Sigma}\mathbf{P}^T$, with $\mathbf{P}$ orthonormal and $\mathbf{\Sigma}$ diagonal. Let $\mathbf{B}$ be any PSD matrix, and take $\widehat{\mathbf{B}} = \mathbf{P}^T\mathbf{B}\mathbf{P}$, so that $\mathbf{B} = \mathbf{P}\widehat{\mathbf{B}}\mathbf{P}^T$. Note that $\widehat{\mathbf{B}}$ is PSD, since

$$\mathbf{x}^T\widehat{\mathbf{B}}\mathbf{x} = (\mathbf{P}\mathbf{x})^T\mathbf{B}(\mathbf{P}\mathbf{x}) \geq 0$$

Therefore, all the diagonal entries in $\widehat{\mathbf{B}}$ must be nonnegative, and not all can be zero since $\mathbf{B} \neq \mathbf{0}$. Finally

$$\begin{aligned}\mathbf{Tr}\,(\mathbf{A}\mathbf{B}) &= \mathbf{Tr}\,((\mathbf{P}\mathbf{\Sigma}\mathbf{P}^T)(\mathbf{P}\widehat{\mathbf{B}}\mathbf{P}^T)) \\ &= \mathbf{Tr}\,(\mathbf{P}\mathbf{\Sigma}\widehat{\mathbf{B}}\mathbf{P}^T) = \mathbf{Tr}\,(\mathbf{\Sigma}\widehat{\mathbf{B}}\mathbf{P}^T\mathbf{P}) = \mathbf{Tr}\,(\mathbf{\Sigma}\widehat{\mathbf{B}}) = \sum \mathbf{\Sigma}_i\hat{\mathbf{B}}_{ii}\end{aligned}$$

And this sum is strictly positive since the elements of $\mathbf{\Sigma}$ - the eigenvalues of $\mathbf{A}$- are strictly positive, and those of $\widehat{\mathbf{B}}$ are nonnegative, with at least one of them being strictly positive. Thus, $\mathbf{A} \bullet \mathbf{B} > 0$. □

Lemma 3.1.6. *For* $\mathbf{A}$ *and* $\mathbf{B}$ *positive-semidefinite,* $\mathbf{A} \bullet \mathbf{B} = 0$ *if and only if* $\mathbf{A}\mathbf{B} = \mathbf{0}$.

Proof. One of the directions is trivial, since $\mathbf{A}\mathbf{B} = \mathbf{0}$ implies that $\mathbf{A} \bullet \mathbf{B} = \mathbf{Tr}\,(\mathbf{A}^T\mathbf{B}) = \mathbf{Tr}\,(\mathbf{A}\mathbf{B}) = 0$. For the other direction, let us use the spectral decompositions $\mathbf{A} = \sum_i \lambda_i \mathbf{v}_i\mathbf{v}_i^T$ and $\mathbf{B} = \sum_j \mu_j \mathbf{u}_j\mathbf{u}_j^T$, where $\lambda_i, \mu_i \geq 0$. With this, we have

$$\begin{aligned}\mathbf{A} \bullet \mathbf{B} &= \mathbf{Tr}\,(\sum_i \lambda_i \mathbf{v}_i\mathbf{v}_i^T \cdot \sum_j \mu_j \mathbf{u}_j\mathbf{u}_j^T) \\ &= \sum_i\sum_j \lambda_i\mu_j \mathbf{Tr}\,(\mathbf{v}_i\mathbf{v}_i^T\mathbf{u}_j\mathbf{u}_j^T) \\ &= \sum_i\sum_j \lambda_i\mu_j \mathbf{v}_i^T\mathbf{u}_j \mathbf{Tr}\,(\mathbf{v}_i\mathbf{u}_j^T) = \sum_i\sum_j \lambda_i\mu_j(\mathbf{v}_i^T\mathbf{u}_j)^2\end{aligned}$$

Thus, if $\mathbf{A} \bullet \mathbf{B} = 0$, then all the pairs $\lambda_i \mu_j (\mathbf{v}_i^T \mathbf{u}_j)$ must be equal to zero, which considering the product

$$\mathbf{AB} = \left(\sum_i \sum_j \lambda_i \mu_j (\mathbf{v}_i \mathbf{v}_i^T \mathbf{u}_j \mathbf{u}_j^T) \right)$$

implies that $\mathbf{AB} = \mathbf{0}$. □

The following result is a semidefinite programming version of the famous Farkas' lemma, which is one of the most widely used *theorems of the alternative* frequently found in optimization theory.

Theorem 3.1.7. *Let $\mathbf{A}_1, \ldots, \mathbf{A}_m$ be symmetric $n \times n$ matrices. Then the system $\sum_i^m y_i \mathbf{A}_i \succ 0$ has no solution in $\mathbf{y}$ if and only if there exists $\mathbf{X} \in \mathbb{S}_+^n$, with $\mathbf{X} \neq \mathbf{0}$, such that $\mathbf{A}_i \bullet \mathbf{X} = 0$ for all $i = 1, \ldots, n$.*

Proof. For the "if" direction, suppose there exists such $\mathbf{X}$, and $\sum_i y_i \mathbf{A}_i \succ 0$ is feasible, then by Lemma 3.1.5 we have

$$\left(\sum_i y_i \mathbf{A}_i \right) \bullet X > 0$$

which contradicts the hypothesis that $\mathbf{A}_i \bullet \mathbf{X} = 0$ for all i.

For the other direction, we require some results about convex cones. Recall that $\mathbb{S}_+^n$ forms a closed convex cone $\mathcal{K}_n$ in $\mathbb{S}_n$. If the system $\sum_i^m y_i \mathbf{A}_i \succ 0$ has no solution, it means the linear subspace $\mathcal{L}_n$ of matrices of the form $\sum y_i \mathbf{A}_i$ does not intersect $\mathcal{K}_n$. Therefore, this linear space is contained in a hyperplane of the form $\{\mathbf{Y} | \mathbf{X} \bullet \mathbf{Y} = 0\}$, with $\mathbf{X} \neq \mathbf{0}$.

Let us assume, without loss of generality, that $\mathcal{K}_n$ lies on the positive side of this plane, that is, $\mathbf{X} \bullet \mathbf{Y} \geq 0$ for every $\mathbf{Y} \in \mathbb{S}_+^n$. But then, by Lemma 3.1.4, $\mathbf{X} \succeq 0$. Also, $\mathbf{X} \bullet \mathbf{A}_i = 0$ for all i since $\mathbf{A}_i \in \mathcal{L}_n$. This completes the proof. □

It is natural a question whether a similar result holds if the positivity condition is relaxed, requiring only semidefintiveness. This is not the case, as one can see that Lemma 3.1.4, which would be required *in lieu* of Lemma 3.1.5, does not provide a strict inequality, necessary to yield a contradiction. However, there is another aspect in which Theorem 3.1.7 can indeed be generalized: extending it to the non-homogeneous case.

Theorem 3.1.8. *Let $\mathbf{A}_1, \ldots, \mathbf{A}_m$ and $\mathbf{C}$ be symmetric matrices. Then $\sum_i y_i \mathbf{A}_i \succ \mathbf{C}$ has no solution if and only if there exists a matrix $\mathbf{X} \succeq 0$, with $\mathbf{X} \neq \mathbf{0}$, such that $\mathbf{A}_i \bullet \mathbf{X} = 0$ for all $i = \{1, \ldots, m\}$ and $\mathbf{C} \bullet \mathbf{X} \geq 0$.*

Proof. The argument is analogous to the one used in Theorem 3.1.7. □

3.2 General Formulation of SDPs

Within the large field of convex optimization, arguably one of the best known classes of problems are those belonging to the subfield of semidefinite programming (SDP). These kind of problems frequently arise in many settings, such as in Mini-max games, eigenvalue optimization and combinatorics. These problems are concerned with the optimization of linear objective functions over the intersection of the cone of positive-semidefinite matrices and a spectrahedron (the equivalent of a simplex in $\mathbb{R}^{n\times n}$).

A general SDP problem has the form

$$\begin{aligned} \max_{\mathbf{X}} \quad & \mathbf{C} \bullet \mathbf{X} \\ \text{s.t.} \quad & \mathbf{A}_i \bullet \mathbf{X} = b_i \qquad i = 1, \dots, m, \\ & \mathbf{X} \succeq 0 \end{aligned} \tag{3.1}$$

where $\succeq$ and $\bullet$ are as defined as in the previous section. Just as in linear programming, several types of problems can be adapted to fit this general form, for example creating slack variables or adding non-negativity conditions to transform equality constraints into inequality constraints and vice versa. Another common feature of SDPs and other types of optimization problems is that frequently the same problem has many equivalent formulations, and the type used usually depends on the particular context.

One of the largest families of SDPs is that of eigenvalue optimization problems. The canonical example of these is the maximum eigenvalue problem:

$$\begin{aligned} \min_{s} \quad & s \\ \text{s.t.} \quad & s\mathbf{I} - \mathbf{A} \succeq 0 \end{aligned} \tag{3.2}$$

The reason for the name is the following. Note that the matrix $s\mathbf{I} - \mathbf{A}$ has eigenvalues $s - \lambda_i$, where λ_i are the eigenvalues of $\mathbf{A}$. Thus, $s\mathbf{I} - \mathbf{A} \succeq 0$ can only be true if $s - \lambda_i \geq 0$, for all i, or equivalently, $s \geq \max_i \lambda_i$. Thus, the solution to (3.2) is precisely the largest eigenvalue of the matrix $\mathbf{A}$.

If we now let $\mathbf{A}$ be an affine combination of matrices, namely $\mathbf{A}(\mathbf{x}) = \mathbf{A}_0 + \sum x_i \mathbf{A}_i$, then the problem

$$\begin{aligned} \min_{s,\mathbf{x}} \quad & s \\ \text{s.t.} \quad & s\mathbf{I} - \mathbf{A}(\mathbf{x}) \succeq 0 \end{aligned} \tag{3.3}$$

corresponds to finding the matrix $\mathbf{A}(x)$ with the smallest largest eigenvalue. This problem, usually referred to as *minimizing the maximal eigenvalue* arises frequently in several applications, and has been studied extensively (see for example Overton [25], and Lewis and Overton [15]). Note that (3.3) bares strong resemblance to the optimization problem for domain adaptation found in Section 2.4.

Note that the matrix

$$\begin{bmatrix} 0 & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & 0 \end{bmatrix}$$

has eigenvalues $\pm\lambda_i$, where λ_i are the eigenvalues of $\mathbf{M}(\mathbf{z})$. Thus, the problem

$$\begin{aligned} \max_{\mathbf{z},s} \quad & s \\ \text{s.t.} \quad & \begin{bmatrix} sI & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & sI \end{bmatrix} \succeq 0 \\ & \mathbf{1}^T\mathbf{z} = 1 \\ & \mathbf{z} \geq 0 \end{aligned}$$

corresponds to minimizing the largest absolute eigenvalue of $\mathbf{M}(\mathbf{z})$. This is naturally equivalent to minimizing the norm-2 of $\mathbf{M}(\mathbf{z})$, which we presented as the alternative formulation of problem (2.15). Thus, the two versions of the optimization problem for domain adaptation are indeed equivalent.

3.3 Duality Theory

Every SDP problem of the form (3.1) has a *dual formulation* of the following form

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n} \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \sum y_i\mathbf{A}_i \succeq \mathbf{C} \end{aligned} \tag{3.4}$$

Although this form is relatively common, another standard form frequently used for the dual is the following

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n} \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \sum_{i=1}^{m} y_i\mathbf{A}_i - \mathbf{S} = \mathbf{C} \\ & \mathbf{S} \succeq 0 \end{aligned} \tag{3.5}$$

Clearly, the problem (3.4) can be taken to the form (3.5) by setting $\mathbf{S} = \sum y_i\mathbf{A}_i - \mathbf{C}$ and then requiring that $\mathbf{S} \succeq 0$.

The relation between a primal problem and its dual is one of the main concepts behind the theory of optimization. This relation can be made more explicit by making use of Lagrangian functions, on which the notion of duality - formally referred to as *Lagrangian duality* [6] - relies. These functions incorporate

The SDP version of the Lagrangian, often called the *conic* Lagrangian, is a function $\mathcal{L} : \mathbb{R}^{n\times n} \times \mathbb{R}^n \to \mathbb{R}$ which incorporates both the objective function and the constraints of the problem (3.1), and is defined as follows

$$\mathcal{L}(\mathbf{X},\mathbf{y}) = \mathbf{C} \bullet \mathbf{X} + \sum_{i=1}^{m} y_i(b_i - \mathbf{A}_i \bullet \mathbf{X}) \tag{3.6}$$

where y_i are called the dual variables. The first observation we make is that

$$\min_{\mathbf{y}} \mathcal{L}(\mathbf{y},\mathbf{Z}) = \begin{cases} \mathbf{C} \bullet \mathbf{X} & \text{if } \mathbf{A}_i \bullet \mathbf{X} = b_i, \quad i = 1, 2, \ldots, n \\ -\infty & \text{otherwise} \end{cases}$$

The reason for this is that if $\mathbf{A}_j \bullet \mathbf{X} \neq b_j$ for some j then $\mathcal{L}(\mathbf{X},\mathbf{y})$ can be made arbitrarily large and negative by taking y_j with the opposite sign as $\mathbf{A}_j \bullet \mathbf{X}$ and letting $|y_j| \to \infty$ while keeping all other variables constant. Thus, the optimal value of the primal SDP problem (3.1) can be equivalently expressed as

$$p^* = \max_{\mathbf{X} \succeq 0} \min_{\mathbf{y}} \mathcal{L}(\mathbf{X},\mathbf{y}) \tag{3.7}$$

On the other hand, the same Lagrangian function (3.6) can be used to define the *Lagrangian dual function* (or just *dual function* for simplicity) as the maximum of $\mathcal{L}$ over the primal variable $\mathbf{X}$:

$$g(\mathbf{y}) := \max_{\mathbf{X} \succeq \mathbf{0}} \mathcal{L}(\mathbf{X},\mathbf{y})$$

Note that from this definition it follows that

$$g(\mathbf{y}) = \begin{cases} \mathbf{b}^T\mathbf{y} & \text{if } C - \sum y_i A_i \preceq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{3.8}$$

since $C - \sum y_i \mathbf{A}_i \succeq 0$ would imply, by Lemma 3.1.4, that $(C - \sum y_i \mathbf{A}_i) \bullet \mathbf{X} \succeq 0$ and thus $\mathcal{L}(\mathbf{X},\mathbf{y})$ could be made arbitrarily large.

The dual problem is then defined as finding the dual variable $\mathbf{y}$ that minimizes $g(\mathbf{y})$. By using (3.8), this problem can be written explicitly as

$$\begin{aligned} \min_{\mathbf{y} \in \mathbb{R}^n} \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \sum y_i \mathbf{A}_i \succeq \mathbf{C} \end{aligned} \tag{3.9}$$

which is precisely the standard from of the dual problem with which we opened this section. From the argument above it follows that the optimal value of this dual is given by

$$d^* = \min_{\mathbf{y}} g(\mathbf{y}) = \min_{\mathbf{y}} \max_{\mathbf{X} \succeq \mathbf{0}} \mathcal{L}(\mathbf{X},\mathbf{y}) \tag{3.10}$$

Let us pause here to analyze our way of proceeding so far. Until now, we have done nothing beyond *defining* another problem, the dual problem, which stems from the Lagrangian function. Besides the fact that $\mathcal{L}$ certainly has information about both the primal and dual embedded in it, and the conspicuous similarity between equations (3.7) and (3.10), it is not entirely clear yet, however, how these problems are related.

The rest of this section is devoted to answering this question. Particularly, we are interested in the relation between the optimal solutions, p^* and d^*, of the primal and dual problems. The reader familiar with minimax problems, game theory and convex optimization, will recognize this relation immediately. Indeed, the main result of this section - the strong duality theorem - can be thought of as a direct consequence of the celebrated Minimax Theorem (through Sion's generalized version [27] of von Neumann's original result). Here, however, we will use Farkas's Lemma (3.1.8) in a proof tailored for the semidefinite programming context.

As a warm-up for this result, we first present the *weak duality* property, which in spite of its simplicity is a first important step towards understanding how the objective functions of the primal and dual problems interact.

Proposition 3.3.1 (Weak Duality for SDP). *If $\mathbf{X}$ is primal-feasible for* (3.1) *and* $(\mathbf{y}, \mathbf{S})$ *are dual-feasible for* (3.5) *then* $\mathbf{C} \bullet \mathbf{X} \leq \mathbf{b}^T \mathbf{y}$.

Proof. The proof[1] is trivial, for is $(\mathbf{y}, \mathbf{S})$ and $\mathbf{X}$ are feasible, then

$$\mathbf{C} \bullet \mathbf{X} = \left(\sum \mathbf{A}_i y_i - \mathbf{S}\right) \bullet \mathbf{X} = \sum y_i (\mathbf{A}_i \bullet \mathbf{X}) - (\mathbf{S} \bullet \mathbf{X}) = \sum y_i b_i - (\mathbf{S} \bullet \mathbf{C})$$

But $\mathbf{C}$ and $\mathbf{S}$ are PSD, so Lemma (3.1.4) implies $\mathbf{S} \bullet \mathbf{C} \geq 0$. Thus, $\mathbf{C} \bullet \mathbf{X} \leq \mathbf{b}^T \mathbf{y}$. □

This duality result is said to be in a *weak* form since the relation between optimal values of the primal and dual problems is given as an inequality. A *strong* result, that is, one ensuring equality of these values, is not possible to give in general for SDPs. However, by adding further assumptions, we can indeed guarantee it. This is the main result we are interested in.

Theorem 3.3.2 (Strong Duality for SDP). *Assume both the primal and the dual of a semidefinite program have feasible solutions, and let p^* and d^* be, respectively, their optimal values. Then, $p^* \leq d^*$. Moreover, if the dual has a strictly feasible solution (i.e. one with $\sum_i y_i \mathbf{A}_i \succ C$) then:*

(1) The primal optimum is attained.

(2) $p^* = d^*$

Proof. (based on Lovasz's notes [17]). By weak duality we have

$$p^* = \mathbf{C} \bullet \mathbf{X}^* \leq \mathbf{b}^T \mathbf{y}^* = d^* \tag{3.11}$$

Now, since d^* is the optimal (i.e. minimal) solution of the dual, the system

$$\begin{gathered} \mathbf{b}^T \mathbf{y} < d^* \\ \sum_i y_i \mathbf{A}_i \succeq \mathbf{C} \end{gathered}$$

[1]More generally, this result is an intrinsic property of the interaction between the infimum and supremum, which (for any function $f(x, y)$, not necessarily convex) satisfy $\sup_{y \in \mathcal{Y}} \inf_{x \in \mathcal{X}} f(x, y) \leq \inf_{x \in \mathcal{X}} \sup_{y \in \mathcal{Y}} f(x, y)$.

is not feasible. Thus, let us define

$$\mathbf{A}'_i = \begin{pmatrix} -b_i & 0 \\ 0 & \mathbf{A}_i \end{pmatrix}, \qquad \mathbf{C}' = \begin{pmatrix} -d^* & 0 \\ 0 & \mathbf{C} \end{pmatrix}$$

Then, by the non-homogeneous SDP Farkas' Lemma (3.1.8) applied to $\mathbf{A}'_i$ and $\mathbf{C}'$, there must exist a nonzero PSD matrix $\mathbf{X}$ such that $\mathbf{X}' \bullet \mathbf{A}'_i = 0$ and $\mathbf{X}' \bullet \mathbf{C}' \geq 0$. Let us label the elements of this matrix as

$$\mathbf{X}' = \begin{pmatrix} \mathbf{x}_0 & \mathbf{x}^T \\ \mathbf{x} & \mathbf{X} \end{pmatrix}$$

Then

$$0 = \mathbf{X}' \bullet \mathbf{A}'_i = \mathbf{A}_i \bullet \mathbf{X} - b_i x_0$$

so $\mathbf{A}_i \bullet \mathbf{X} = b_i x_0$ for all i. Similarly, $\mathbf{C} \bullet \mathbf{X} \geq x_0 d^*$. But since both $\mathbf{X}$ and $\mathbf{C}$ are PSD, Lemma (3.1.4) implies $x_0 \geq 0$. We claim that $x_0 \neq 0$. Otherwise, by the semidefiniteness of $\mathbf{X}'$, we would have $\mathbf{x} = 0$, and since $\mathbf{X}' \neq \mathbf{0}$, that would mean $\mathbf{X} \neq \mathbf{0}$. The existence of such an $\mathbf{X}$ would in turn imply that the system $\sum_i y_i \mathbf{A}_i \succ \mathbf{C}$ is not solvable (Lemma 3.1.8), contradicting the hypothesis of the existence of a strictly feasible solution to the dual.

Thus, $x_0 \neq 0$. Then, by diving by x_0 throughout we obtain a solution $\hat{\mathbf{X}}$ with objective value $\mathbf{C} \bullet \hat{\mathbf{X}} \geq d^*$. By (3.11), this inequality must be an equality, and $\hat{\mathbf{X}}$ must be the optimal (maximal) solution to the primal. Thus $p^* = d^*$. This completes the proof. $\square$

This result, which is analogous to the corresponding duality theorem for Linear Programming, gives us the final ingredient to ensure that the solutions to the primal and dual versions of an SDPs are equivalent. According to the argument above, this can be enforced by requiring strict positive-definiteness in the constraints of the dual problem, as opposed to the general case (3.4) where the constraints were simply positive-semidefinite.

3.4 Solving Semidefinite Programs Numerically

Besides being a very vast subfield of optimization, semidefinite programming is also a very important one, for many reasons. For example, SDPs arise in a wide variety of contexts and applications, such as in operations research, control theory and combinatorics. Another reason is that other convex optimization problems, for instance, linear programs or quadratically constrained quadratic programs, can be cast as SDPs, and thus the latter offer a unified study of the properties of all these [30].

In addition, the formulation of SDPs - as seen in the previous section- is simple and concise. Their numerical solution, however, is a matter of more controversy. Depending on whom one asks, SDPs can be solved very efficiently [30] or rather slowly [2]. The issue here is scalability. In Machine Learning, where the dimension of the data is often very large, methods that work well in low dimensions might not be a very good approach.

Most current off-the-shelf algorithms for solving SDPs are based on primal-dual interior-point methods, and run in polynomial time, albeit with large exponents. SeDuMi[2], one of the state-of-the-art generic SDP solvers, has a computational complexity of $O(n^2m^{2.5} + m^{3.5})$, in a problem with n decision variables and m rows in the semidefinite constraint. This makes it impractical for high dimensional problems.

In machine learning, however, scalability is prefered to accuracy when it comes to optimization tasks. As [11] points out, it is often the case that the data used is assumed to be noisy, and thus one can settle for an approximate solution. This is particularly true when the solution of the optimization problem, an SDP for example, is only an intermediate tool and not the end goal of the learning task [14]. This is the case for the problem (2.15) posed in Section 2.4, where we seek to solve an SDP to obtain a re-weighting of the training points to be used in a regression algorithm, and thus we are not interested in its accuracy *per se*.

On the other hand, SDPs often have special structure or sparsity, which can be exploited to solve them much more efficiently. This suggests tailoring algorithms instead of using generic solvers. Therefore, the approach to solving problem (2.15), as in other SDPs for machine learning, is to combine the idea of approximate solutions with special features of the constraints in order to design methods that are as efficient as possible.

[2]Self-Dual-Minimization toolbox, available for MATLAB. `http://sedumi.ie.lehigh.edu/`

Chapter 4

The Matrix Multiplicative Weights Algorithm

In this chapter we study a generalization for matrices of the well-known weighted majority algorithm [16]. This generalization has been independently discovered, in different versions, by Tsuda, Rätsch and Warmuth [28] as the *Matrix Exponentiated Gradient Updates* method, and later by Arora, Hazan and Kale [1] as the *Matrix Multiplicative Weights* (MMW) algorithm. Since it adapts more easily to the context of our problem, we follow here the derivation presented in the latter, and thus refer to the algorithm with that name.

In Section 4.1 we present the standard MMW algorithm, analyzing it from a game-theory point of view. Then, in Section 4.2 we present Arora and Kale's [2] adaptation of the algorithm to the context of semidefinite programming, which is naturally relevant to our problem of interest.

4.1 Motivation: learning with expert advice

We will motivate the Multiplicative Weights algorithm from a online-learning theory point of view, which can also be understood from a game theory approach. The ideas presented here generalize the notion of *learning with expert advice* used to motivate the more well-known (and simpler) weighted majority algorithm.

Suppose that we are trying to predict an outcome from within a set of outcomes $\mathbf{P}$ with the advice of n "experts". A well known approach consists of deciding based on a weighted majority vote, where the weights of the experts are to be modified to include the information obtained in each round of the game. For this purpose, we assume the existence of a matrix $\mathbf{M}$ for which the (i, j) entry is the penalty that the expert i pays when the observed outcome is $j \in \mathbf{P}$. For reasons that will be explained later, we will suppose that these penalties are in the interval $[-\ell, \rho]$, where $\ell \leq \rho$. The number ρ is called the *width* of the problem.

Transforming this learning setting to a 2-player zero-sum game is easy. For this, we let $\mathbf{M}$ be a payoff matrix, so that when player one plays the strategy i and the second player plays the strategy j, the payoff to the latter is $\mathbf{M}(i,j)$. Then, if we denote by $\mathbf{M}(i,\mathcal{C})$ the values of this payoff matrix by varying the columns, the strategy of the first player should be to minimize $\mathrm{E}_{i\in\mathcal{R}}[\mathbf{M}(i,j)]$, while the second player would try to maximize $\mathrm{E}_{j\in\mathcal{R}}[\mathbf{M}(i,j)]$, where $\mathcal{R}$ varies over the rows of the payoff matrix. To match the setting above, we would be viewing this game from the perspective of the first player.

Back to the learning setting, however, the Multiplicative Weights Algorithm as proposed by Arora et al. [1] proceeds as follows:

Multiplicative Weights Update Algorithm

Set initially $w_i^T = 1$ for all i. For rounds $t = 1, 2, \dots$

(1) Associate the distribution $\mathcal{D}^t = \{p_1^t, \dots, p_n^t\}$ on the experts, where $p_i^t = w_i^t / \sum_k w_k^t$.

(2) Pick an expert according to $\mathcal{D}^t$ and use it to make a prediction.

(3) Observe the outcome $j \in \mathbf{P}$ and update the weights as follows

$$w_i^{t+1} = \begin{cases} w_i^t(1-\epsilon)^{\mathbf{M}(i,j^t)/\rho} & \text{if } \mathbf{M}(i,j^t) \geq 0 \\ w_i^t(1+\epsilon)^{-\mathbf{M}(i,j^t)/\rho} & \text{if } \mathbf{M}(i,j^t) < 0 \end{cases}$$

A reasonable desire about a prediction algorithm is that it performs not much worse that then best expert in hindsight. In fact, it is shown by the authors that for $\epsilon \leq \frac{1}{2}$ the following bound holds

$$\sum_t \mathbf{M}(\mathcal{D}^t, j^t) \leq \frac{\rho \ln n}{\epsilon} + (1+\epsilon) \sum_{\geq 0} \mathbf{M}(i, j^t) + (1-\epsilon) \sum_{<0} \mathbf{M}(i, j^t) \tag{4.1}$$

In a subsequent publication [2], Arora et al. provide a matrix version of this algorithm, of which a particular version is used in the context of SDPs. This adaptation will be the main focus of the following section. For the moment, let us generalize the 2-player game shown above to its matrix form.

In this new setting, the first player chooses a unit vector $\mathbf{v} \in \mathbb{R}^{n-1}$ from a distribution $\mathcal{D}$, and the other player chooses a matrix $\mathbf{M}$ with $0 \preceq \mathbf{M} \preceq \mathbf{I}$. The first player then has to "pay" $\mathbf{v}^T\mathbf{M}\mathbf{v}$ to the second. Again, we are interested in the expected loss of the first player, namely

$$\mathop{\mathrm{E}}_{\mathcal{D}}[\mathbf{v}^T\mathbf{M}\mathbf{v}] = \mathbf{M} \cdot \mathop{\mathrm{E}}_{\mathcal{D}}[\mathbf{v}\mathbf{v}^T] = \mathbf{M} \bullet \mathbf{P} \tag{4.2}$$

where $\mathbf{P}$ is a *density matrix*, that is, it is positive semidefinite and has unit trace. Note in the relation in (4.2) that the game can be equivalently cast in terms of $\mathbf{P}$, since the vector $\mathbf{v}$ appears only through this matrix.

To turn this game into its online version, we suppose that in each round we choose a density matrix $\mathbf{P}$ and observe the event $\mathbf{M}^{(t)}$. For a fixed vector $\mathbf{v}$, the best possible outcome for the first player is given by the vector that minimizes the total loss, given by

$$\sum_{t=1}^{T} \mathbf{v}^T \mathbf{M}^{(t)} \mathbf{v} = \mathbf{v}^T \left(\sum_{t=1}^{T} \mathbf{M}^{(t)} \right) \mathbf{v} = \mathbf{v}^T \overline{\mathbf{M}} \mathbf{v} \tag{4.3}$$

Naturally, this value is minimized with $\mathbf{v}_n$, the eigenvector corresponding to λ_n the smallest eigenvalue of $\overline{\mathbf{M}}$, for which the loss is $\lambda_n \overline{\mathbf{M}}$. The algorithm we seek should not perform much worse than this.

The Matrix Multiplicative Weights (MMW) algorithm is - as it name indicates - a generalization of the algorithm shown above, which iteratively updates a weight matrix instead of the vector $\mathbf{w}$. The method proceeds in an analogous fashion, evidently taking into account the fact that the observed event and density take now the form of matrices. The algorithm is the following.

Matrix Multiplicative Weights Update Algorithm

Fix an $\epsilon < \frac{1}{2}$ and let $\epsilon' = -\ln(1-\epsilon)$. For rounds $t = 1, 2, \ldots$

(1) Compute $\mathbf{W}^{(t)} = (1-\epsilon)^{\sum_{\tau=1} \mathbf{M}^{(\tau)}} = \exp\big(-\epsilon'(\sum_{\tau=1} \mathbf{M}^{(\tau)})\big)$

(2) Use the density matrix $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\mathbf{Tr}\,(\mathbf{W}^{(t)})}$ and observe the event $\mathbf{M}^{(t)}$.

As mentioned before, the algorithm should perform not much worse that the minimum possible loss after T round. Indeed, in the last section of this chapter we prove a theorem that provides such a guarantee in terms of the minimum loss.

4.2 Multiplicative Weights for Semidefinite Programming

In [2], the authors propose using the MMW algorithm to solve SDPs approximately. For this, they devise a way to treat the constrains of an optimization problem as *experts*, and then design a method which alternatively solves a feasibility problem in the dual, and updates the primal variable with an exponentiated matrix update. The result is a Primal-Dual algorithm template, which they then customize for several problems from combinatorial optimization to obtain considerable improvements over previous methods.

For this purpose, let us consider, as done in [2], a general SDP with n^2 variables and m

constraints in the form

$$\begin{aligned} \max_{X} \quad & \mathbf{C} \bullet \mathbf{X} \\ \text{s.t.} \quad & \mathbf{A}_i \bullet \mathbf{X} \leq b_k \qquad i = 1, \ldots, m, \\ & \mathbf{X} \succeq \mathbf{0} \end{aligned} \tag{4.4}$$

To simplify the notation, we will assume that $\mathbf{A}_1 = I$ and $b_1 = R$. With this condition, a bound on the trace of the feasible solutions is created, namely $\mathbf{Tr}\,\mathbf{X} \leq R$. In light to the SDP theory presented in Chapter 3, the dual of this problem is

$$\begin{aligned} \min_{\mathbf{y}} \quad & \mathbf{b} \cdot \mathbf{y} \\ \text{s.t.} \quad & \sum_{j=1}^{m} \mathbf{A}_i y_j \succeq \mathbf{C} \\ & \mathbf{y} \geq 0 \end{aligned} \tag{4.5}$$

To adapt the general MMW algorithm to this context, we first define the candidate solution to be

$$\mathbf{X}^{(t)} = R\mathbf{P}^{(t)} \tag{4.6}$$

Also, we now take the *observed event* to be

$$\mathbf{M}^{(t)} = \frac{1}{2\rho} \left(\sum \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho \mathbf{I} \right) \tag{4.7}$$

Leaving aside the meaning of the parameter $\rho > 0$ for a moment, we notice that using (4.7) as our observation matrix in the MMW algorithm would imply updating the primal variable with a term that depends on how feasible the dual problem is. The improvement that this update allows us can be tracked by the use of an additional auxiliary routine, called the ORACLE, which tests the *validity* of the current solution $\mathbf{X}^{(t)}$, by verifying the statement

$$\exists\, \mathbf{y} \in \mathcal{D}_\alpha \quad \text{such that} \quad \sum (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \geq 0 \tag{4.8}$$

where $\mathcal{D}_\alpha = \{\mathbf{y} \mid \mathbf{y} \geq 0, \mathbf{b}^T \mathbf{y} \leq \alpha\}$ and α is the algorithm's current guess for the optimum value of the problem. The following lemma shows why this criterion is useful.

Lemma 4.2.1. *Suppose the* ORACLE *finds* $\mathbf{y}$ *satisfying* (4.8)*, then* $\mathbf{X}^{(t)}$ *is primal infeasible or* $\mathbf{C} \bullet \mathbf{X}^{(t)} \leq \alpha$*. If, on the contrary, the* ORACLE *fails, then some scalar multiple of* $\mathbf{X}^{(t)}$ *is a primal-feasible solution with objective value at least* α*.*

Proof. Suppose, for the sake of contradiction, that the ORACLE finds such a $\mathbf{y}$ but $\mathbf{X}^{(t)}$ is feasible with $\mathbf{C} \bullet \mathbf{X}^{(t)} > \alpha$. Then

$$\begin{aligned} \sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) &\leq \sum_{j=1}^{m} b_j y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) && (\text{since } \mathbf{X}^{(t)} \text{ is primal feasible}) \\ &\leq \alpha - (\mathbf{C} \bullet \mathbf{X}^{(t)}) && (\text{since } \mathbf{y} \in \mathcal{D}_\alpha) \\ &< \alpha - \alpha = 0 && (\text{since we suppose } \mathbf{C} \bullet \mathbf{X}^{(t)} > \alpha) \end{aligned}$$

But this contradicts (4.8). Thus, either $\mathbf{X}^{(t)}$ is infeasible, or $\mathbf{C} \bullet \mathbf{X}^{(t)} \leq \alpha$.

Now suppose the ORACLE fails. Consider the following linear program with its dual:

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j \\ \text{s.t.} \quad & \mathbf{b}^T \mathbf{y} \leq \alpha \\ & \mathbf{y} \geq 0 \end{aligned} \qquad \begin{aligned} \min_{\phi} \quad & \alpha\phi \\ \text{s.t.} \quad & b_j \phi \geq (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) \\ & \phi \geq 0 \end{aligned} \tag{4.9}$$

Since no $\mathbf{y}$ exists satisfying the condition of the ORACLE, this means that for any $\mathbf{y}$ with $\mathbf{y} \geq 0$ and $\mathbf{b}^T\mathbf{y} \leq \alpha$, we have $\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) < 0$. Thus, the optimal value of the primal of (4.9) must be less than $\mathbf{C} \bullet \mathbf{X}^{(t)}$. Since this optimum is finite, from the theory of linear programming we know that the dual is feasible, and thus must have the same optimum. In other words, $\alpha\phi^* \leq \mathbf{C} \bullet \mathbf{X}^{(t)}$ for the optimal α^*. The condition $\mathbf{A}_1 \bullet \mathbf{X}^{(t)} = \mathbf{Tr}\,(\mathbf{X}^{(t)}) = b_1 = R$ implies that $\phi^* \geq 1$. So, if we define $\mathbf{X}^* = \frac{1}{\phi^*}\mathbf{X}^{(t)}$, then $\mathbf{A}_j \bullet \mathbf{X}^* = \frac{1}{\phi^*} \leq b_j \frac{\phi}{\phi^*} \leq b_j$ and $\mathbf{C} \bullet \mathbf{X}^* \geq \frac{\alpha\phi^*}{\phi^*} = \alpha$. Therefore, $\mathbf{X}^*$ is primal feasible (for the SDP) and has objective value at least α. □

Thus, if the ORACLE succeeds, it means that the primal candidate $\mathbf{X}^{(t)}$ is not yet optimal, either because it is not feasible or because its objective value is below α, which -if our guess is correct- is the optimal value of the dual. By weak duality, this last statement implies that there might be another primal variable $\widehat{\mathbf{X}}^{(t)}$ with a larger objective value, and thus the algorithm continues. Furthermore, the vector $\mathbf{y}$ retrieved contains information as to how to improve the candidate solution $\mathbf{X}^{(t)}$. This is why $\mathbf{y}$ is used the construction of the update (4.7). If the algorithm finishes after T rounds without the ORACLE failing, it means the guess for α was too high, so that it is reduced for the next round. On the contrary, if the ORACLE fails at any point, Lemma 4.2.1 asserts that there exists a primal feasible solution with objective value at least α, so by weak duality, the optimal dual solution must be larger than this. We must conclude then that the guess α was too low, so it is increased and the algorithm restarts. The optimal solution is found in this way through binary search on α.

The key for the efficiency of this algorithm comes from the fact that the problem which the ORACLE solves has only two linear constrains and no PSD constraint. In other words, it finds a solution $\mathbf{y}$ which is not required to be dual feasible. The LP problem that the ORACLE has to solve can often be fixed by *a priori* rules, which making its implementation very efficient.

Now, ρ in (4.7) is a parameter that depends on the particular structure of the ORACLE used. It is defined as the smallest value ρ such that for any $\mathbf{X}$, the output $\mathbf{y}$ of the ORACLE satisfies $\|\mathbf{A}_j y_j - \mathbf{C}\| \leq \rho$. This value plays a critical role on the performance of the algorithm, for it controls the rate at which progress can be made in each iteration. A large value of ρ means that the algorithm can only make progress slowly, and thus most of the design of the ORACLE is aimed towards make this width parameter small.

Putting all the pieces together, we obtain the following algorithm.

Algorithm 1 Primal-Dual Algorithm for SDP

```
Require: $\delta, \alpha, \rho$
  Set $\mathbf{X}^{(1)} = \frac{R}{n}I$, $\epsilon = \frac{\delta\alpha}{2\rho R}$, $\epsilon' = \ln(1-\epsilon)$, $T = \frac{8\rho^2 R^2 \ln(n)}{\delta^2}$
  for $t = 1, 2, \ldots, T$ do
    if ORACLE Fails then
      Output $\mathbf{X}^{(t)}$ and exit.
    else
      Get $\mathbf{y}^{(t)}$ from ORACLE.
    end if
    $\mathbf{M}^{(t)} := (\sum \mathbf{A}_j y_j^{(t)} - C + \rho I)/2\rho$
    $\mathbf{W}^{(t+1)} := (1-\epsilon)^{\sum_{\tau=1}\mathbf{M}^{(\tau)}} = \exp\big(-\epsilon'(\sum_{\tau=1}\mathbf{M}^{(\tau)})\big)$
    $\mathbf{X}^{(t+1)} := \frac{R\mathbf{W}^{(t+1)}}{\mathbf{Tr}\,\mathbf{W}^{(t+1)}}$
  end for
  return X
```

Note that the last two steps of Algorithm 1 are identical to those of the general MMW algorithm presented in the previous section. What changes now is that instead of being an *observed* event, the matrix $\mathbf{M}$ is obtained by means of the ORACLE in each round. The choice of parameters ϵ and T is made so as to take the exact number of steps required theoretically to achieve a δ-accurate solution. This guarantee is shown in Theorem 4.3.2, in the next section.

The reader will have noticed by this point that Algorithm 1 is not specific at all in terms of implementation details. In this sense, it can be more correctly described as a *meta-algorithm*, which requires a fair amount of customization to be used on a particular SDP problem. It is a general scheme with which this type of problems can be solved iteratively with matrix exponential updates, although all the details of an eventual implementation must be derived on a case-by-case basis. Chapter 5 is devoted to this derivation for our problem interest, the SDP of discrepancy minimization.

It is important to mention that the intrinsic generality of Arora and Kale's algorithm is a double-edged sword. On the one hand, it provides an optimization method that can be used for a very vast family of SDP problems, and it offers equally general guarantees in terms of iterations. On the other hand, its efficiency, which depends heavily on the details of the implementation and work-per-iteration, will vary greatly from case to case. This idea will be revisited in Chapter 6, when we analyze the efficiency of our implementation of this algorithm for domain adaptation, and in our concluding remarks.

4.3 Learning Guarantees

In this last section of this chapter, we prove learning guarantees for the algorithms presented in the previous sections. The first result gives a bound for the expected loss $\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$ of the general MMW algorithm in terms of the minimum loss, which we have shown is given by the smallest eigenvalue of $\sum_{t=1}^{T} \mathbf{M}^{(t)}$.

Theorem 4.3.1. *Suppose* $\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \ldots \mathbf{P}^{(T)}$ *are the density matrices generated by the Matrix Multiplicative Weights algorithm. Then*

$$\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq (1+\epsilon)\lambda_n \left(\sum_{t=1}^{T} \mathbf{M}^{(t)} \right) + \frac{\log n}{\epsilon} \tag{4.10}$$

Proof. The proof is done by focusing on the weight matrix $\mathbf{W}^{(t)}$, and using its trace as a potential function, a strategy common to many proofs of learning bounds.

First, by using the Golden-Thompson inequality for the trace of matrix exponentials (namely, $\mathbf{Tr}\,(e^{A+B}) \leq \mathbf{Tr}\,(e^A e^B)$), we can bound $\mathbf{Tr}\,(\mathbf{W}^{(t+1)})$ as follows:

$$\begin{aligned}
\mathbf{Tr}\,(\mathbf{W}^{(t+1)}) &= \mathbf{Tr}\,\left(\exp\left\{ -\epsilon' \sum_{\tau=1}^{t} \mathbf{M}^{(\tau)} \right\} \right) \\
&\leq \mathbf{Tr}\,\left(\exp\left\{ -\epsilon' \sum_{\tau=1}^{t-1} \mathbf{M}^{(\tau)} \right\} \exp\left\{ -\epsilon' \mathbf{M}^{(t)} \right\} \right) = \mathbf{W}^{(t)} \bullet \exp\left\{ -\epsilon' \mathbf{M}^{(t)} \right\}
\end{aligned}$$

Now, using the fact that $(1-\epsilon)^A \preceq (I - \epsilon\mathbf{A})$ for a matrix satisfying $0 \preceq \mathbf{A} \preceq I$, then the second term can be bounded as

$$\exp\left\{ -\epsilon' \mathbf{M}^{(t)} \right\} = \exp\left\{ \log(1-\epsilon)\mathbf{M}^{(t)} \right\} = (1-\epsilon)^{\mathbf{M}^{(t)}} \preceq I - \epsilon\mathbf{M}^{(t)}$$

so that

$$\begin{aligned}
\mathbf{Tr}\,(\mathbf{W}^{(t+1)}) &\leq \mathbf{W}^{(t)} \bullet (I - \epsilon\mathbf{M}^{(t)})) \\
&= \mathbf{Tr}\,(\mathbf{W}^{(t)}) - \epsilon\mathbf{W}^{(t)} \bullet \mathbf{M}^{(t)}) \\
&= \mathbf{Tr}\,(\mathbf{W}^{(t)}) \left[1 - \epsilon\left(\frac{\mathbf{W}^{(t)}}{\mathbf{Tr}\,(\mathbf{W}^{(t)})} \bullet \mathbf{M}^{(t)}\right) \right] \\
&= \mathbf{Tr}\,(\mathbf{W}^{(t)}) \left[1 - \epsilon\mathbf{P}^{(t)} \bullet \mathbf{M}^{(t)} \right] \\
&\leq \mathbf{Tr}\,(\mathbf{W}^{(t)}) \cdot \exp(-\epsilon\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)})
\end{aligned}$$

where the last inequality is true since $1 - a \leq e^{-a}$ for $a \geq 1$. Now, using induction and the fact that $\mathbf{Tr}\,\mathbf{W}^{(1)} = \mathbf{Tr}\,(\mathbf{I}) = n$, we get

$$\mathbf{Tr}\,(\mathbf{W}^{(T+1)}) \leq n \exp(-\epsilon \sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) \tag{4.11}$$

On the other hand, let us denote by $\lambda_k(A)$ the eigenvalues of $\mathbf{M}$, λ_n being the smallest of them. Then

$$\begin{aligned}
\mathbf{Tr}\,(\mathbf{W}^{(T+1)}) &= \mathbf{Tr}\,(\exp\{-\epsilon' \sum_{t=1}^{T} \mathbf{M}^{(t)}\}) \\
&= \sum_k \lambda_k(e^{-\epsilon' \sum_t \mathbf{M}^{(t)}}) = \sum_k e^{-\epsilon' \lambda_k(\sum_t \mathbf{M}^{(t)})} \geq e^{-\epsilon' \lambda_n(\sum_t \mathbf{M}^{(t)})}
\end{aligned}$$

If we combine the two expressions for $\mathbf{Tr}\left(\mathbf{W}^{(T+1)}\right)$ above, we obtain

$$e^{-\epsilon' \lambda_n(\sum_t \mathbf{M}^{(t)})} \leq n \exp(-\epsilon \sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)})$$

which after some manipulation, becomes

$$\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq (1+\epsilon)\lambda_n \left(\sum_{t=1}^{T} \mathbf{M}^{(t)} \right) + \frac{\log n}{\epsilon}$$

This completes the proof. □

Let us interpret the statement of Theorem 4.3.1 carefully. This result tells us that the expected loss is upper-bounded by a multiple of the minimum loss and a term depending on the number of experts n. For a fixed n, the only ingredient we can control is the *discount rate* ϵ. Unfortunately, changes in this parameter have opposite effects on the terms making up the bound (4.10): positive in the first one and negative in the second one.

This trade-off has a clear learning interpretation. For this, let us analyze, as done frequently in the literature, the *learning rate* η, where $e^{-\eta} = \epsilon$. A large value of η (small ϵ) causes a high learning rate, that is, weight is quickly removed from poor performing experts. This, however, might cause probability to be concentrated on just a few select experts, neglecting potential information by other not top-performing experts and thus results in a "reduced" expert number. Naturally, this negative effect is more dramatic when there are few experts, making $\ln n/\epsilon$ more sensitive on ϵ.

As a direct corollary of Theorem 4.3.1, we can obtain a bound on the number of iterations required by the MMW for Semidefinite Programming to achieve a δ-accurate solution for the guessed optimal value α.

Theorem 4.3.2. *In the Primal-Dual SDP Algorithm 1, assume that the* ORACLE *never fails for* $T = \frac{8\rho^2 R^2 \ln(n)}{\delta^2 \alpha^2}$ *iterations. Let* $\bar{\mathbf{y}} = \frac{\delta\alpha}{R} e_1 + \frac{1}{T}\sum_{t=1}^{T} \mathbf{y}^{(t)}$. *Then* $\bar{\mathbf{y}}$ *is a feasible dual solution with the objective value at most* $(1+\delta)\alpha$.

Proof. In the context of Theorem (4.3.1), let us take $\mathbf{M}^{(t)} = (\sum \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I})/2\rho$ and $\mathbf{X}^{(t)} = R\mathbf{P}^{(t)}$. Then

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} = \frac{1}{2\rho}(\sum \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I}) \bullet \frac{1}{R}\mathbf{X}^{(t)} \geq \frac{\rho}{2\rho R}\mathbf{I} \bullet \mathbf{X}^{(t)} \geq \frac{1}{2}$$

where the last inequality is true because the ORACLE finds a $\mathbf{y}^{(t)}$ such that $\frac{1}{2\rho}(\sum \mathbf{A}_j y_j^{(t)} -$

$\mathbf{C}) \bullet \mathbf{X}^{(t)} \geq 0$. Using this in the bound (4.10) of Theorem 4.3.1 we get

$$\begin{aligned}
\frac{1}{2} &\leq (1+\epsilon)\lambda_n \left(\sum_{t=1}^{T} \mathbf{M}^{(t)}\right) + \frac{\ln n}{\epsilon} \\
&= (1+\epsilon)\,\lambda_n \left(\sum_{t=1}^{T} \frac{1}{2\rho}(\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \rho\mathbf{I})\right) + \frac{\ln n}{\epsilon} \\
&= (1+\epsilon)\left(\frac{T}{2\rho}\right)\left[\lambda_n \left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C}\right) + \rho\right] + \frac{\ln n}{\epsilon}
\end{aligned}$$

Multiplying both sides by $\frac{2\rho}{T(1+\epsilon)}$ and reordering we obtain

$$\frac{\rho}{T(1+\epsilon)} - \frac{2\rho \ln n}{T\epsilon(1+\epsilon)} - \rho \leq \lambda_n \left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C}\right)$$

By substituting the values $\epsilon = \frac{\delta\alpha}{2\rho R}$ and $T = \frac{8\rho^2 R^2 \ln n}{\delta^2\alpha^2}$, and after some simplification, this becomes

$$-\frac{\delta\alpha}{R} \leq \lambda_n \left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C}\right) \tag{4.12}$$

Using $\bar{\mathbf{y}} = \frac{\delta\alpha}{R} e_1 + \frac{1}{T}\sum_{t=1}^{T} y^{(t)}$, and recalling that $\mathbf{A}_i = \mathbf{I}$, we see that

$$\sum_{j=1}^{m} \mathbf{A}_j \bar{y}_j - \mathbf{C} = \mathbf{A}_1 \left(\frac{\delta\alpha}{R}\right) + \sum_{j=1}^{m} \frac{1}{T}\sum_{t=1}^{T} \mathbf{A}_i y_j^{(t)} - \mathbf{C} = \frac{\delta\alpha}{R}\mathbf{I} + \left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m} \mathbf{A}_i y_j^{(t)} - \mathbf{C}\right)$$

And by (4.12), we must have that the smallest eigenvalue of this matrix is positive. In other words, $0 \preceq \sum_{j=1}^{m} \mathbf{A}_j \bar{y}_j - \mathbf{C}$, which implies $\bar{\mathbf{y}}$ is a dual feasible solution. In addition, since $b_1 = R$ and $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$ for all $t = 1, \ldots, T$, then

$$\begin{aligned}
\mathbf{b}^t \mathbf{y} = b_1 \left(\frac{\delta\alpha}{R}\right) + \mathbf{b}^T \left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{y}^{(t)}\right) &= \delta\alpha + \frac{1}{T}\sum_{t=1}^{T} \mathbf{b}^T \mathbf{y}^{(t)} \\
&\leq \delta\alpha + \frac{1}{T}\sum_{t=1}^{T} \alpha = (1+\delta)\alpha
\end{aligned}$$

This completes the proof. □

Notice the dependency of the bound of Theorem 4.3.2 on $\frac{1}{\delta^2}$. This squared accuracy term, which is irremediably embedded in the algorithm, can prove to be too slow for many applications. We thus see that in order to make the MMW algorithm for SDPs competitive with other methods, the computational cost per iteration has to be decreased to the minimum possible. We will discuss this issue further in the next Chapter.

Chapter 5

Matrix Multiplicative Weights for Domain Adaptation

In this, the main chapter of the thesis, we introduce a modified version of the MMW algorithm for the specific context of domain adaptation. Combining ideas from the three previous chapters, we show how Arora and Kale's primal dual meta-algorithm (Algorithm 1 from the previous chapter) can be tailored to this context.

We include an introductory section with some properties of matrix exponentials that will be necessary in the derivation of the algorithm. We then proceed with a step-by-step derivation of the algorithm, exploiting the structure of the problem to whenever possible to improve efficiency. We conclude the chapter with a theoretical comparison of our algorithm with the smooth approximation method used by Cortes and Mohri [7].

5.1 Some properties of matrix exponentials

Despite intuitively challenging at first, the notion of matrix exponentials is a natural generalization of the usual scalar-valued exponential function. Since sums, scalar multiplication and powers of square matrices are properly defined, we have all the elements required to use the power-series characterization of the exponential, but this time for matrix arguments. This leads to the following definition.

Definition 5.1.1. *Let* $\mathbf{A}$ *be an* $n \times n$ *real or complex matrix. The power series*

$$e^{\mathbf{A}} = \mathbf{I} + \mathbf{A} + \frac{1}{2}\mathbf{A}^2 + \frac{1}{3!}\mathbf{A}^3 + \cdots = \sum_{k=1}^{\infty} \frac{1}{k!}\mathbf{A}^k$$

is called the **matrix exponential** *of* $\mathbf{A}$.

A natural question upon analyzing Definition 5.1.1 is whether this power-series converges in general. The answer is affirmative and easy to prove by making use of the Frobenius norm. For any squared matrix, if we let $a_k = \frac{1}{k!}(\mathbf{A}^k)_{ij}$ we have

$$\sum_{k=0}^{\infty} a_k = \sum_{k=0}^{\infty} \frac{1}{k!}(\mathbf{A}^k)_{ij} \leq \sum_{k=0}^{\infty} \frac{1}{k!}\|\mathbf{A}^k\|_F \leq \sum_{k=0}^{\infty} \frac{1}{k!}\|\mathbf{A}\|_F^k = e^{\|\mathbf{A}\|_F}$$

so each entry of the matrix power-series converges. Thus, the matrix exponential is well-defined.

This notion of matrix exponentials lets us generalize other real valued functions into their matrix equivalents. These functions, although used less frequently than the normal exponential, are nonetheless equally useful and will arise naturally in our adaptation of the MMW algorithm in Section 5.2.

Definition 5.1.2. *Let* $\mathbf{A}$ *be a real or complex* $n \times n$ *matrix. We define*

$$\sinh(\mathbf{A}) = \mathbf{A} + \frac{1}{3!}\mathbf{A}^3 + \frac{1}{5!}\mathbf{A}^5 + \cdots = \sum_{k=0}^{\infty} \frac{1}{(2k+1)!}\mathbf{A}^{2k+1}$$
$$\cosh(\mathbf{A}) = \mathbf{I} + \frac{1}{2!}\mathbf{A}^2 + \frac{1}{4!}\mathbf{A}^4 + \cdots = \sum_{k=0}^{\infty} \frac{1}{(2k)!}\mathbf{A}^{2k}$$

Note that such hyperbolic trigonometric functions are, again, defined analogously to their real-valued counterparts. Indeed, any function defined by a power series can be generalized to a matrix-function equivalent, paying the necessary attention to convergence conditions. From the Definitions 5.1.1 and 5.1.2, the reader will notice that the identities

$$\sinh(\mathbf{A}) = \frac{1}{2}(e^{\mathbf{A}} - e^{-\mathbf{A}}), \quad \cosh(\mathbf{A}) = \frac{1}{2}(e^{\mathbf{A}} + e^{-\mathbf{A}}) \tag{5.1}$$

are also true for matrices, and thus can be equivalently used to define $\cosh(\mathbf{A})$ and $\sinh(\mathbf{A})$.

The reader might wonder if such a notion of hyperbolic functions of matrices is necessary beyond theoretical curiosity. As it turns out, some common classes of matrices require these functions to express their exponentials. For example,

$$\begin{aligned}
\exp\begin{bmatrix}\mathbf{0} & \mathbf{A}\\ \mathbf{A} & \mathbf{0}\end{bmatrix} &= \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{A}\\ \mathbf{A} & \mathbf{0}\end{bmatrix} + \frac{1}{2!}\begin{bmatrix}\mathbf{0} & \mathbf{A}^2\\ \mathbf{A}^2 & \mathbf{0}\end{bmatrix} + \frac{1}{3!}\begin{bmatrix}\mathbf{A}^3 & \mathbf{0}\\ \mathbf{0} & \mathbf{A}^3\end{bmatrix} + \cdots \\
&= \begin{bmatrix}\sum_{k=0}^{\infty}\frac{1}{(2k)!}\mathbf{A}^{2k} & \sum_{k=0}^{\infty}\frac{1}{(2k+1)!}\mathbf{A}^{2k+1}\\ \sum_{k=0}^{\infty}\frac{1}{(2k+1)!}\mathbf{A}^{2k+1} & \sum_{k=0}^{\infty}\frac{1}{(2k)!}\mathbf{A}^{2k}\end{bmatrix} \\
&= \begin{bmatrix}\cosh(\mathbf{A}) & \sinh(\mathbf{A})\\ \sinh(\mathbf{A}) & \cosh(\mathbf{A})\end{bmatrix}
\end{aligned} \tag{5.2}$$

In the next section, we will encounter precisely this type of matrices.

The following two lemmas provide a quick way to evaluate the trace of matrix exponentials and hyperbolic functions if the eigenvalues of the matrix $\mathbf{A}$ are known. This is a trivial consequence of a more general fact that we demonstrate in the proof: the eigenvalues of $e^{\mathbf{A}}$ are the exponentials of those of $\mathbf{A}$.

Lemma 5.1.3. *Suppose $\mathbf{A}$ is a diagonalizable $n\times n$ matrix with eigenvalues λ_i, $i \in \{1,\dots,n\}$. Then $\mathbf{Tr}\,(e^{\mathbf{A}}) = \sum_{i=1}^n e^{\lambda_i}$.*

Proof. Let $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}$ be the eigendecomposition of $\mathbf{A}$. Then, $\mathbf{A}^k = \mathbf{V}\mathbf{\Lambda}^k\mathbf{V}^{-1}$ for every k, so that

$$e^{\mathbf{A}} = \sum_{k=1}^{\infty} \frac{1}{k!}\mathbf{A}^k = \mathbf{V}\left(\sum_{k=1}^{\infty}\frac{1}{k!}\mathbf{\Lambda}^k\right)\mathbf{V} = \mathbf{V}\text{diag}\left(\sum_{k=1}^{\infty}\frac{1}{k!}\lambda_i^k\right)\mathbf{V}^{-1} = \mathbf{V}\text{diag}(e^{\lambda_i})\mathbf{V}^{-1}$$

where $\text{diag}(a_i)$ denotes a diagonal matrix composed of the elements a_i. This implies that e^{λ_i}, $i \in \{1,\dots,n\}$, are the eigenvalues of $e^{\mathbf{A}}$ (with the same eigenvector as $\mathbf{A}$), and since the trace is equal to the sum of the eigenvalues, this yields $\mathbf{Tr}\,e^{\mathbf{A}} = \sum_{i=1}^n e^{\lambda_i}$. □

Lemma 5.1.4. *In the same setting as Lemma 5.1.3, it is also true that $\mathbf{Tr}\,(\cosh(A)) = \sum_i^n \cosh(\lambda_i)$ and $\mathbf{Tr}\,(\sinh(A)) = \sum_i^n \sinh(\lambda_i)$.*

Proof. The result readily follows from using the definition of the hyperbolic functions in conjunction with Lemma 5.1.3. For the first of these identities, for example, we would proceed as follows

$$\mathbf{Tr}\,\cosh(\mathbf{A}) = \frac{1}{2}\mathbf{Tr}\,(e^{\mathbf{A}} + e^{-\mathbf{A}}) = \frac{1}{2}\left(\sum_{i=1}^n e^{\lambda_i} + \sum_{i=1}^n e^{-\lambda_i}\right) = \sum_i^n \frac{1}{2}(e^{\lambda_i} + e^{-\lambda_i}) = \sum_i^n \cosh(\lambda_i)$$

The proof for $\sinh(\mathbf{A})$ is analogous. □

We these lemmas at hand, we are now ready to handle the matrix exponential updates that lie in the core of the MMW algorithm.

5.2 Tailoring MMW for Adaptation

In Chapter 4, we presented a primal-dual algorithm for SDPs based on the MMW Algorithm. We showed how this *meta-algorithm* requires a great level of customization before being used for a particular problem. In this section, we propose such an implementation to solve the discrepancy minimization problem of domain adaptation (2.15).

The first step towards tailoring Algorithm 1 for the case of domain adaptation is to reformulate our original problem and cast it in the form of the generic SDP to which Arora's method applies. For this, we remind the reader of the original form of the problem, as obtained in Section 2.4:

$$\begin{array}{lll} \min\limits_{\mathbf{z},s} & s & \\ \text{s.t.} & \begin{bmatrix} sI & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & sI \end{bmatrix} & \succeq 0 \\ & \mathbf{1}^T\mathbf{z} & = 1 \\ & \mathbf{z} & \geq 0 \end{array}$$

where the matrix $\mathbf{M}(\mathbf{z})$ is given by the affine combination $\mathbf{M}(\mathbf{z}) = \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i\mathbf{M}_i$. Note that the positive-semidefiniteness constraint can be reexpressed as

$$\begin{bmatrix} tI & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & tI \end{bmatrix} = s\begin{bmatrix} I & \\ & I \end{bmatrix} + \begin{bmatrix} & \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i\mathbf{M}_i \\ \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i\mathbf{M}_i & \end{bmatrix} \succeq 0$$

where we use, for ease of notation, blank sections in the matrices to denote blocks composed only of zeros. This is equivalent to

$$s\begin{bmatrix} I & \\ & I \end{bmatrix} + \sum_{i=1}^{\mathfrak{m}} z_i \begin{bmatrix} & -\mathbf{M}_i \\ -\mathbf{M}_i & \end{bmatrix} \succeq \begin{bmatrix} & -\mathbf{M}_0 \\ -\mathbf{M}_0 & \end{bmatrix}$$

so if we set

$$C = \begin{bmatrix} & -\mathbf{M}_0 \\ -\mathbf{M}_0 & \end{bmatrix}, \quad \mathbf{A}_0 = I_{2\mathfrak{m}\times 2\mathfrak{m}}, \quad z_0 = s, \quad \mathbf{A}_i = \begin{bmatrix} & -\mathbf{M}_i \\ -\mathbf{M}_i & \end{bmatrix}, \quad i \in \{1, \dots, \mathfrak{m}\}$$

and use the vector $\mathbf{b} = (1, 0, \dots, 0)^T$ for the objective function, then the problem can be finally restated as

$$\begin{array}{ll} \min\limits_{\mathbf{z}} & \mathbf{b}\cdot\mathbf{z} \\ \text{s.t.} & \sum\limits_{j=1}^{m} \mathbf{A}_i z_j \succeq \mathbf{C} \\ & \mathbf{z} \geq 0 \end{array} \tag{5.3}$$

which is precisely in the form of the general dual SDP (5.3) analyzed in Section 4.2. Note that we have conveniently defined $\mathbf{A}_0$ and $\mathbf{b}_0$ in such a way that the solution of the primal must satisfy $\mathbf{Tr}\,\mathbf{X} = \mathbf{A}_0 \bullet \mathbf{X} \leq b_0 = 1$, so in this case the parameter bounding the trace of the primal variable is simply $R = 1$.

In Algorithm 1, it was necessary to compute in each iteration the exponential of a sum of matrices[1] of the form

$$\mathbf{E}^{(t)} = \frac{1}{2\rho}\left(\sum_{j=1}^{\mathfrak{m}} \mathbf{A}_j z_j^{(t)} - \mathbf{C} + \rho\mathbf{I}\right)$$

which, in this case, becomes

$$\mathbf{E}^{(t)} = \frac{1}{2\rho}\left(s\begin{bmatrix} I & \\ & I \end{bmatrix} + \sum_{i=1}^{\mathfrak{m}} z_i \begin{bmatrix} & -\mathbf{M}_i \\ -\mathbf{M}_i & \end{bmatrix} + \begin{bmatrix} & \mathbf{M}_0 \\ \mathbf{M}_0 & \end{bmatrix}\right) = \frac{1}{2\rho}\begin{bmatrix} (s+\rho)\mathbf{I} & \mathbf{M}(\mathbf{z}^{(t)}) \\ \mathbf{M}(\mathbf{z}^{(t)}) & (s+\rho)\mathbf{I} \end{bmatrix}$$

[1]Although both in [2] and in Chapter 4 these matrices are denoted by $\mathbf{M}^{(t)}$, we use here $\mathbf{E}^{(t)}$ to avoid conflicting notation with the matrices $\mathbf{M}(\mathbf{z})$ of the adaptation problem.

so that

$$\sum_{\tau=1}^{t} \mathbf{E}^{(\tau)} = \frac{1}{2\rho}\begin{bmatrix} t(s+\rho)\mathbf{I} & \sum_{\tau=1}^{t}\mathbf{M}(\mathbf{z}^{(\tau)}) \\ \sum_{\tau=1}^{t}\mathbf{M}(\mathbf{z}^{(\tau)}) & t(s+\rho)\mathbf{I} \end{bmatrix}$$

We will abuse the notation by denoting, from here onward, $\mathbf{M}_\tau := \mathbf{M}(\mathbf{z}^{(\tau)})$ for simplicity. From the equation above it follows that the exponential matrix weight updates are given in this case by

$$\begin{aligned}
\mathbf{W}^{(t+1)} &= \exp\left\{-\epsilon'\left(\sum_{\tau=1}^{t}\mathbf{E}^{(\tau)}\right)\right\} \\
&= \exp\left\{\frac{-\epsilon' t}{2\rho}(s+\rho)\begin{bmatrix}\mathbf{I} & \\ & \mathbf{I}\end{bmatrix} + \frac{-\epsilon'}{2\rho}\begin{bmatrix} & \sum_{\tau=1}^{t}\mathbf{M}_\tau \\ \sum_{\tau=1}^{t}\mathbf{M}_\tau & \end{bmatrix}\right\} \\
&= \exp\left\{\frac{-\epsilon' t}{2\rho}(s+\rho)\begin{bmatrix}\mathbf{I} & \\ & \mathbf{I}\end{bmatrix}\right\}\cdot\exp\left\{\frac{-\epsilon'}{2\rho}\begin{bmatrix} & \sum_{\tau=1}^{t}\mathbf{M}_\tau \\ \sum_{\tau=1}^{t}\mathbf{M}_\tau & \end{bmatrix}\right\} \\
&= e^{\frac{-\epsilon' t}{2\rho}(s+\rho)}\cdot\exp\begin{bmatrix} & \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau \\ \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau & \end{bmatrix} & \text{(since } e^{\alpha\mathbf{I}} = e^{\alpha}\mathbf{I}\text{)} \\
&= e^{\frac{-\epsilon' t}{2\rho}(s+\rho)}\cdot\begin{bmatrix}\cosh\left(\frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau\right) & \sinh\left(\frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau\right) \\ \sinh\left(\frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau\right) & \cosh\left(\frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau\right)\end{bmatrix} & \text{(by (5.2))} \\
&= e^{\frac{-\epsilon' t}{2\rho}(s+\rho)}\cdot\begin{bmatrix}\cosh(\mathbf{M}_{1:t}) & \sinh(\mathbf{M}_{1:t}) \\ \sinh(\mathbf{M}_{1:t}) & \cosh(\mathbf{M}_{1:t})\end{bmatrix}
\end{aligned}$$

with $\mathbf{M}_{1:t} = \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau$. Note that we have used the fact that scalar multiples of the identity commute with any matrix, so their exponentials satisfy the additivity property.

Thus, the primal variable maintained by the MMW algorithm is updated in each iteration with the rule

$$\mathbf{X}^{(t+1)} = \frac{R\mathbf{W}^{(t+1)}}{\mathbf{Tr}\left(\mathbf{W}^{(t+1)}\right)} = \frac{1}{2\mathbf{Tr}\,\cosh\left(\mathbf{M}_{1:t}\right)}\begin{bmatrix}\cosh(\mathbf{M}_{1:t}) & \sinh(\mathbf{M}_{1:t}) \\ \sinh(\mathbf{M}_{1:t}) & \cosh(\mathbf{M}_{1:t})\end{bmatrix} \tag{5.4}$$

5.3 The ORACLE

Based on the ORACLE of the original MMW algorithm, our own ORACLE will have to search for a dual variable $\mathbf{z}$ satisfying

$$\sum_{j=1}^{\mathfrak{m}}(\mathbf{A}_j \bullet \mathbf{X}^{(t)})z_j - (\mathbf{C}\bullet\mathbf{X}^{(t)}) \geq 0 \tag{5.5}$$

Note that we can fold the additional constraint $\sum_{i=1}^{\mathfrak{m}} z_i = 1$ into the polytope $\mathcal{D}_\alpha$ where these dual solutions are seeked. Thus, we have

$$\mathcal{D}_\alpha = \left\{\mathbf{z} : \quad \mathbf{z}\geq 0, \quad \sum_{i=1}^{\mathfrak{m}} z_i = 1, \quad z_0 \leq \alpha\right\}$$

For the inequality (5.5), we need to compute products of the form $\mathbf{A}_i \bullet \mathbf{X}^{(t)}$ for each $i \in \{0, \dots, \mathfrak{m}\}$. Using the form of the matrices $\mathbf{A}_i$ and $\mathbf{X}^{(t)}$ in this case, we see that

$$\begin{aligned}
\mathbf{A}_i \bullet \mathbf{X}^{(t)} &= \frac{1}{2\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \begin{bmatrix} & -\mathbf{M}_i \\ -\mathbf{M}_i & \end{bmatrix} \bullet \begin{bmatrix} \cosh(\mathbf{M}_{1:t}) & \sinh(\mathbf{M}_{1:t}) \\ \sinh(\mathbf{M}_{1:t}) & \cosh(\mathbf{M}_{1:t}) \end{bmatrix} \\
&= \frac{-1}{2\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \mathbf{Tr} \begin{bmatrix} \mathbf{M}_i \sinh(\mathbf{M}_{1:t}) & \mathbf{M}_i \cosh(\mathbf{M}_{1:t}) \\ \mathbf{M}_i \cosh(\mathbf{M}_{1:t}) & \mathbf{M}_i \sinh(\mathbf{M}_{1:t}) \end{bmatrix} \\
&= \frac{-2\ \mathbf{Tr}\ \big(\mathbf{x}_i \mathbf{x}_i^T \sinh(\mathbf{M}_{1:t})\big)}{2\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \\
&= -\frac{\mathbf{x}_i^T \sinh(\mathbf{M}_{1:t}) \mathbf{x}_i}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \qquad (5.6)
\end{aligned}$$

On the other hand

$$\mathbf{A}_0 \bullet \mathbf{X}^{(t)} = \frac{1}{2\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \begin{bmatrix} \mathbf{I} & \\ & \mathbf{I} \end{bmatrix} \bullet \begin{bmatrix} \cosh(\mathbf{M}_{1:t}) & \sinh(\mathbf{M}_{1:t}) \\ \sinh(\mathbf{M}_{1:t}) & \cosh(\mathbf{M}_{1:t}) \end{bmatrix} = 1 \qquad (5.7)$$

and

$$\begin{aligned}
\mathbf{C} \bullet \mathbf{X}^{(t)} &= \frac{1}{2\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \begin{bmatrix} & -\mathbf{M}_0 \\ -\mathbf{M}_0 & \end{bmatrix} \bullet \begin{bmatrix} \cosh(\mathbf{M}_{1:t}) & \sinh(\mathbf{M}_{1:t}) \\ \sinh(\mathbf{M}_{1:t}) & \cosh(\mathbf{M}_{1:t}) \end{bmatrix} \\
&= -\frac{\mathbf{Tr}\ \big(\mathbf{M}_0 \sinh(\mathbf{M}_{1:t})\big)}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \qquad (5.8)
\end{aligned}$$

Using (5.6), (5.7) and (5.8), the objective of the ORACLE becomes

$$s + \frac{\mathbf{Tr}\ \mathbf{M}_0 \sinh(\mathbf{M}_{1:t})}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} - \sum_{i=1}^{\mathfrak{m}} z_i \mathbf{x}_i^T \left(\frac{\sinh(\mathbf{M}_{1:t})}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \right) \mathbf{x}_i \geq 0 \qquad (5.9)$$

Let

$$\eta_0 = \frac{\mathbf{Tr}\ \mathbf{M}_0 \sinh(\mathbf{M}_{1:t})}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)}, \quad \eta_i = \mathbf{x}_i^T \left(\frac{\sinh(\mathbf{M}_{1:t})}{\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big)} \right) \mathbf{x}_i$$

Then, the ORACLE finally reduces to finding $\mathbf{z}$ such that

$$\begin{gathered}
z_0 - \sum_{i=1}^{\mathfrak{m}} z_i \eta_i \geq -\eta_0 \\
\sum_{i=1}^{\mathfrak{m}} z_i = 1, \quad z_0 \leq \alpha, \quad \mathbf{z} \geq 0 \qquad (5.10)
\end{gathered}$$

Here, we make some important observations. First, note that in every iteration all the information the ORACLE requires is α and the $\mathfrak{m} + 1$ scalars η_i. These have all the relevant information of the current primal candidate solution condensed in them. Secondly, note that (5.3) is a simple linear programming problem. Intuitively, it can be solved easily by taking $z_0 = \alpha$ and then $z_j = 1$ corresponding to the η_j with the smallest value. If the inequality is not true in this case, then the no other choice of $\mathbf{z}$ will satisfy the problem, so the ORACLE should return a failure status. The downside of this approach is that the width of the Oracle will not be under control, which would reflect in a slow progress of the algorithm. Thus, we need a smarter approach.

Recall that he width of the Oracle, as presented in Chapter 4, is the smallest value for which every solution returned by this method satisfies $\| \sum z_i \mathbf{A}_i - \mathbf{C}\| \leq \rho$. In this case, we have

$$\begin{aligned}\rho = \left\| \sum_{j=0}^{\mathfrak{m}} \mathbf{A}_j z_j^{(t)} - \mathbf{C} \right\| &= \max \lambda_i \left(\begin{bmatrix} sI & \mathbf{M}(\mathbf{z}) \\ \mathbf{M}(\mathbf{z}) & sI \end{bmatrix} \right) \\ &\leq s + \max |\lambda_i(\mathbf{M}(\mathbf{z}))| \\ &\leq \alpha + \max |\lambda_i(\mathbf{M}(\mathbf{z}))| \qquad \text{(since } s = z_0 \text{ is chosen s.t. } z_0 \leq \alpha)\end{aligned}$$

Therefore, ρ can be minimized by concentrating all the weight in the x_i for which $\max |\lambda_i(\mathbf{M}_0 - A_i)|$ is smallest. These values can be computed *a priori*, sorted and fed to the Oracle, so that it proceeds from the top of the list down trying to satisfy the main inequality of (5.3) by assigning all the weight to the corresponding element of $\mathbf{z}$. In practice, the Oracle is rarely observed to require moving past the first few elements of this list. The parameter ρ can then be set safely as α plus one of the first elements in this list. Altough and attempt could also be made to minimize α in the expression above whenever possible, there is little use to this, for there is no obvious way to bound its value without obstructing the task of the Oracle in terms of gathering information about the feasibility of the primal solution.

The intuition behind the way our Oracle works is clear. In every iteration, it tries to update the candidate matrix by giving more weight to those elements $\mathbf{x}_i$ that, while preserving positive-semidefiniteness, individually minimize the norm of the objective matrix $\mathbf{M}(\mathbf{z})$. It does so in a seemingly naïve way: simply alloting all the weight to the best possible candidate in this sense. This strategy, however, accounts for an extremely efficient Oracle that no longer has to solve a LP problem, which might otherwise be expensive computationally.

5.4 Computing Matrix Exponentials

The vanilla-MMW update method has one major disadvantage: it requires the computation of a matrix exponential in each iteration. Matrix exponentiation, besides suffering from accuracy issues, has a computational cost of $O(n^3)$ for general square matrices. There are various methods to compute these exponentials, some of which make use of particular matrix structure (see the classic survey by Moler [21] and the revisited Twenty-Five Year anniversary version [22]). In practice, they are all *dubious*, to use Moler's terminology, in the sense that none are completely satisfactory in terms of efficiency or stability.

In the original Primal-Dual MMW algorithm [2], the authors opt for computing the matrix exponential only approximately, and thus recur to Johnson-Lindenstrauss dimensionality reduction. They compute ϵ-accurate projections of the columns of the Cholesky decomposition of $\mathbf{X}^{(t)}$ into a $O(\frac{\log n}{\delta^2})$-dimensional space, given by $e^{\mathbf{M}}\mathbf{u}$, where $\mathbf{u}$ are random vectors in the reduced dimension space. For our problem, however, more efficient ways of approaching this problem are possible, mainly by exploiting the very particular structure of the matrices $\mathbf{M}(\mathbf{z})$ and our ORACLE.

First of all, the reader will note that in Algorithm 1 the matrix $\mathbf{X}^{(t)}$, which is a trace-normalized version of $\mathbf{W}^{(t)} = \exp(-\epsilon'(\sum_{\tau=1}^{t} \mathbf{E}^{(t)}))$, is only really required by the ORACLE to solve its feasibility problem. And we saw in Section 5.3 that the latter makes use of this exponential matrix through scalar values η_i, all defined in terms of hyperbolic functions of the matrices $-\frac{\epsilon'}{2\rho}\sum_{\tau=1}^{t} \mathbf{M}(\mathbf{z}^{(\tau)})$. We propose two different alternatives for exploiting this fact: the first one obtains these scalar values in terms of the eigenvalues and eigenvectors of $\mathbf{M}_{1:t}$, and the second one computes them by an efficient implementation of matrix powering. Both of these avoid storing complete matrices and, whenever possible, reduce computations to inner vector products and -to a lesser extent- matrix-vector products.

5.4.1 Through Eigenvalue Decomposition

To analyze the first of these alternatives, suppose we had an eigendecomposition of $\mathbf{M}_{(1:\tau)} = -\frac{\epsilon'}{2\rho}\sum_{\tau=1}^{t} \mathbf{M}_\tau$ of the form

$$\mathbf{M}_{1:t} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$$

where $\mathbf{\Lambda} = diag(\lambda_i)$ contains the eigenvalues of $\mathbf{M}_{1:t}$, and V has $\mathbf{v}_i$ the eigenvectors as its columns. Note that at most $\text{rank}(\mathbf{X}) \leq \min\{\mathfrak{m}+\mathfrak{n}, N\}$ of the λ_i are nonzero. Using this decomposition, we now revisit each of the terms in the ORACLE (5.9) individually. First, using Lemma 5.1.4 we note that

$$\mathbf{Tr}\ \cosh\big(\mathbf{M}_{1:t}\big) = \sum_{i=1}^{N} \cosh(\lambda_i)$$

Next, we look at the products $\mathbf{x}_i^T \sinh(\mathbf{M}_{1:t})\mathbf{x}_i$. These are given by

$$\mathbf{x}_i^T \sinh(\mathbf{M}_{1:t})\mathbf{x}_i = \mathbf{x}_i^T \mathbf{V}\text{diag}(\sinh(\lambda_i))\mathbf{V}^T x_i = \sum_{j=1}^{N} \sinh(\lambda_j)([\mathbf{V}^T\mathbf{x}_i]_j)^2$$

The third term of interest can be computed either by avoiding the construction of the matrix $\mathbf{M}_0$ altogether by using

$$\begin{aligned}\mathbf{Tr}\,\mathbf{M}_0 \sinh(\mathbf{M}_{1:t}) &= \mathbf{Tr}\left(\sum_{i=\mathfrak{m}+1}^{\mathfrak{q}} P(\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i^T\right)\sinh(\mathbf{M}_{1:t}) \\ &= \sum_{i=\mathfrak{m}+1}^{\mathfrak{q}} P(\mathbf{x}_i)\mathbf{x}_i^T \sinh(\mathbf{M}_{1:t})\mathbf{x}_i = \sum_{i=\mathfrak{m}+1}^{\mathfrak{q}}\sum_{j=1}^{N} P(\mathbf{x}_i)\sinh(\lambda_j)([\mathbf{V}^T x_i]_j)^2\end{aligned}$$

when $\mathfrak{n} < N$, or, in the opposite case, by using

$$\mathbf{Tr}\mathbf{M}_0 \sinh(\mathbf{M}_{1:t}) = \mathbf{Tr}\mathbf{M}_0\mathbf{V}\sinh(\boldsymbol{\Lambda})\mathbf{V}^T = \mathbf{Tr}\mathbf{M}_0\left(\sum_{j=1}^{N}\sinh(\lambda_j)\mathbf{v}_j\mathbf{v}_j^T\right) = \sum_{j=1}^{N}\sinh(\lambda_j)\mathbf{v}_i^T\mathbf{M}_0\mathbf{v}_i$$

which avoids the computational dependency on the dimension $\mathfrak{n}$ by explicitly building the matrix $\mathbf{M}_0$. Using these properties, we can re-express the terms η required by the ORACLE as follows:

$$\eta_0 = \frac{\sum_{i=\mathfrak{m}+1}^{\mathfrak{q}}\sum_{j=1}^{N} P(\mathbf{x}_i)\sinh(\lambda_j)([\mathbf{V}^T\mathbf{x}_i]_j)^2}{\sum_{i=1}^{N}\cosh(\lambda_i)}, \qquad \eta_i = \frac{\sum_{j=1}^{N}\sinh(\lambda_j)([\mathbf{V}^T\mathbf{x}_i]_j)^2}{\sum_{i=1}^{N}\cosh(\lambda_i)} \tag{5.11}$$

Since these is the only form moment in which we reuquire the exponentiated matrices, if we have access to the eigendecomposition of the matrices $\mathbf{M}_{1:t}$, then there is no need to actually exponentiate, multiply nor store any matrix apart form $\mathbf{V}$. It is important to mention that the terms η_i should always be computed directly as a quotient, for computing denominators and numerators separately and then dividing them will often cause rounding errors if floating-point arithmetic.

Unfortunately, obtaining eigendecompositions is in general a computationally expensive procedure. However, a crucial observation is pertinent at this point. In each iteration, the matrices $\mathbf{M}(\mathbf{z}^{(t)}) = \mathbf{X}\mathbf{D}^{(t)}\mathbf{X}^T$ change only in the top $\mathfrak{m}$ elements of the diagonal matrix $\mathbf{D}^{(t)}$. Naturally, so do the matrices

$$\mathbf{M}_{1:t} = \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau = \frac{-\epsilon'}{2\rho}\mathbf{X}\big(\sum_{\tau=1}^{t}\mathbf{D}^{(\tau)}\big)\mathbf{X}^T$$

This causes the eigenvalues and eigenvectors of these to vary only very slightly in each iteration, so eigenvalue methods that depend on the proximity of the starting vector, such as the Power Method or the Inverse Iteration, can have rapid convergence.

Furthermore, recall from the last part of Chapter 2 that we exposed the possibility of working with the original matrices of the adaptation problem $\mathbf{M}(\mathbf{z})$, or equivalently, with their kernelized version $\mathbf{M}'(\mathbf{z}) = (\mathbf{X}^T\mathbf{X})^{1/2}\mathbf{D}(\mathbf{X}^T\mathbf{X})^{1/2}$. We showed in Lemma 2.4.1 that their set of eigenvalues is the same, and that the eigenvectors of either can easily be obtained from the other's. Thus, assuming that the dimension of the input space N is larger than the sum of source and target sample sizes $\mathfrak{m} + \mathfrak{n}$, it will be computationally easier to compute the eigendecomposition of the latter type of matrices. For our experiments, we updated the eigendecomposition at every step using the the Inverse Iteration method on the kernelized version, which empirically showed convergence after very few iterations.

5.4.2 Through efficient matrix powering

The second alternative for computing the required matrix exponentials relies on exploiting the structure of the matrix $\mathbf{M}(\mathbf{z})$. Recall that $\mathbf{M}(\mathbf{z}) = \mathbf{M}_0 - \sum_{i=1}^{\mathfrak{m}} z_i \mathbf{M}_i$, and $\mathbf{M}_0 = \sum_{j=\mathfrak{m}+1}^{\mathfrak{q}} \hat{P}(\mathbf{x}_j)\mathbf{x}_j\mathbf{x}_j^T$, with $\mathfrak{q} = \mathfrak{m} + \mathfrak{n}$. Thus

$$\mathbf{M}_{1:t} = \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t}\mathbf{M}_\tau = \frac{-\epsilon'}{2\rho}\mathbf{X}\big(\sum_{\tau=1}^{t}\mathbf{D}^{(\tau)}\big)\mathbf{X}^T = \mathbf{X}\hat{\mathbf{D}}\mathbf{X}^T$$

where $\hat{\mathbf{D}}_i = \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t} z_i^{(\tau)}$ for $i \in \{1,\dots,\mathfrak{m}\}$ and $\hat{\mathbf{D}}_i = \frac{-\epsilon'}{2\rho} t P(\mathbf{x}_i)$ for $i \in \{\mathfrak{m}+1,\dots,\mathfrak{m}+\mathfrak{n}\}$. Hence $\mathbf{M}_{1:t}$ can be expressed in a similar fashion to $\mathbf{M}(\mathbf{z})$ as follows

$$\mathbf{M}_{1:t} = \mathbf{M}_{1:t}^0 - \sum_{i=1}^{\mathfrak{m}} \hat{z}_i^{(t)}\mathbf{x}_i\mathbf{x}_i^T$$

with $\mathbf{M}_{1:t}^0 = \sum_{j=\mathfrak{m}+1}^{\mathfrak{q}} \frac{-\epsilon'}{2\rho} t P(\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i^T$ and $\hat{z}_i^{(t)} = \frac{-\epsilon'}{2\rho}\sum_{\tau=1}^{t} z_i^{(\tau)}$. For this reason, we carry out the following analysis for the more general matrix $\mathbf{M}(\mathbf{z})$, bearing in mind that the same procedure can be used on the matrix of interest $\mathbf{M}_{1:t}$.

To compute the products of the form $\mathbf{x}^T \sinh(\mathbf{M}(\mathbf{z}))\mathbf{x}$ we need $x_i \mathbf{M}^k x_i$ for $i = 1,\dots,m$, and $k = 1,\dots,K$, where K can be fixed to a relatively small integer and thus compute the exponentials approximately. Note that $\mathbf{Tr}\ \cosh(\mathbf{M}(\mathbf{z}))$ can also be obtained in this way, by computing $\sum_{i=1}^{N} \mathbf{e}_i \cosh(\mathbf{M}(\mathbf{z}))\mathbf{e}_i$, for $\mathbf{e}_i$ the elements of the standard basis. The method for obtaining products of this form proceeds as follows.

Efficient Matrix Powering

Initially, build the matrix $\mathbf{M}_0$ and set $\mathbf{u}_j^{(0)} = \mathbf{x}_j$.
For rounds $k = 1, 2, 3, \dots, K$:

(1) Compute $\mathbf{M}_0\mathbf{u}_j^{(k-1)}$ for $j = 1,\dots,\mathfrak{m}$.

(2) Compute $\mathbf{x}_l^T\mathbf{u}_j^{(k-1)}$ for $j, l \in \{1,\dots,\mathfrak{m}\}$.

(3) Update, for $j = 1,\dots,\mathfrak{m}$:

$$\mathbf{u}_j^{(k)} = \mathbf{M}_0\mathbf{u}_j^{(k-1)} + \sum_{j=1}^{\mathfrak{m}} z_l\big(\mathbf{x}_l^T\mathbf{u}_j^{(k-1)}\big)\mathbf{x}_l \tag{5.12}$$

(4) Return $\mathbf{x}_j^T\mathbf{u}_j^{(K)}$, for $j = 1,\dots,\mathfrak{m}$.

The reason for the update (5.12) is clear, since

$$\begin{aligned}\mathbf{M}(\mathbf{z})^k\mathbf{x}_j &= \mathbf{M}(\mathbf{z})\left[\mathbf{M}(\mathbf{z})^{k-1}\mathbf{x}_j\right] \\ &= \left(\mathbf{M}_0 + \sum_{l=1}^{m} z_l\mathbf{M}_l\right)\mathbf{M}(\mathbf{z})^{k-1}\mathbf{x}_j = \mathbf{M}_0(\mathbf{M}(\mathbf{z})^{k-1}x_j) + \sum_{l=1}^{\mathfrak{m}} z_l\mathbf{x}_l(\mathbf{x}_l^T\mathbf{M}(\mathbf{z})^{k-1}\mathbf{x}_j)\end{aligned}$$

and the algorithm is made so that $\mathbf{u}_j^{(k)} = \mathbf{M}(\mathbf{z})^k\mathbf{x}_j$.

Let us analyze the computational complexity of the this scheme. Building the matrix $\mathbf{M}_0$ requires $\mathfrak{n}$ outer products, each of which requires N^2 products, to the work for this initialization is $O(\mathfrak{n}N^2)$. In the iterative part of the algorithm, the first element computed, $\mathbf{M}_0\mathbf{u}_j^{k-1}$, requires $O(N^2)$ work. Then, $\mathfrak{m}$ inner products with $O(N)$ each are computed, and during the update rule, for each of the N entries of the vector, $\mathfrak{m}$ multiplications and $\mathfrak{m}-1$ sums are required, adding up to $O(N\mathfrak{m})$. Therefore, the total work per round is $O(\mathfrak{m}N^2 + N\mathfrak{m}^2)$. So, to obtain $\mathbf{x}_i^T\mathbf{M}^k\mathbf{x}_i$ for all $i = 1, \dots, \mathfrak{m}$ and all powers $k = 1, \dots, K$, we need work $O(K\mathfrak{m}N^2 + K\mathfrak{m}^2N + N^2\mathfrak{n})$ where the third term corresponds to the pre-computation.

If, on the contrary, $n << N$, it will be computationally more efficient not to build the matrix $\mathbf{M}_0$, and to compute the products $\mathbf{M}_0\mathbf{u}$ as sums of the form $\sum \mathbf{x}_i^T\mathbf{u}\mathbf{x}_i$. This would instead yield a total work of $O(K\mathfrak{m}\mathfrak{n}N + K\mathfrak{m}^2N)$.

5.5 The Algorithm

After having developed and customized over the previous sections every component necessary for an implementation of Algorithm 1, we finally put all the pieces together and present the Matrix Multiplicative Weights method for Domain Adaptation (MMW-DA).

Our algorithm proceeds, as explained for Algorithm 1, with a binary search on the optimal value α of the dual SDP problem, which is in fact the original version of the discrepancy minimization problem given in Chapter 2. It requires three secondary routines: the ORACLE, `EigenUpdate` and `getEta`. The first of these was explained in detail in Section 5.3, the second updates the eigendecomposition of the matrix $\mathbf{M}_{1:t}$, and the last obtains the scalars η_i required for the ORACLE. The particular implementation of each of these functions can be chosen with relative freedom, and these different choices would naturally impact running times and efficiency in different ways. It is important, however, to stress the fact there is room for improvements on our algorithm by designing faster versions of these sub-routines.

In our implementation, the function `EigenUpdate` uses inverse iteration to update the eigenvalues and eigenvectors $\mathbf{V}$ and $\mathbf{\Lambda}$, and the function `getEta` simply computes the elements of η as in (5.11). The input $\mathcal{L}$ is a list with the indices of the maximal eigenvalues of $\mathbf{M}(\mathbf{e}_j)$ sorted in increasing order, as described in Section 5.3.

The pseudo-code for the our MMW-DA algorithm is the following.

Algorithm 2 MMW-DA

```
Require: $\delta > 0$, $tol$, $\mathcal{L}$, $\rho$.
  Initialize:
  $\alpha = \|\mathbf{M}(\mathbf{z}_0)\|$
  $\alpha_L \leftarrow 0$
  $\alpha_R \leftarrow \alpha$.
  $(\mathbf{V}_0, \mathbf{\Lambda}_0) = eig(\mathbf{M}(\mathbf{z}))$
  while $\alpha_R - \alpha_L > tol$ do
      Set $\epsilon \leftarrow \frac{\delta\alpha}{2\rho}$
      Set $\epsilon' \leftarrow -\ln(1-\epsilon)$
      $\forall i \; \eta_i \leftarrow 0$.
      Set $T = 8\frac{\rho^2 \ln(n)}{\delta^2\alpha^2}$
      for $t = 1, \ldots, T$ do
          Run ORACLE$(\eta, \mathcal{L}, \alpha)$
          if ORACLE failed then
              $\alpha_L \leftarrow \alpha$
              $\alpha \leftarrow \frac{1}{2}(\alpha_L + \alpha_R)$
              Break
          else
              $\mathbf{z}^{(t)} \leftarrow$ORACLE$(\eta, \mathcal{L}, \alpha)$
              $(\mathbf{V}^{(t)}, \mathbf{\Lambda}^{(t)}) \leftarrow$EigenUpdate$(\mathbf{V}^{(t-1)}, \mathbf{\Lambda}^{(t-1)})$
              $\eta \leftarrow$ getEta$(\mathbf{V}^{(t)}, \mathbf{\Lambda}^{(t)}, \mathbf{X})$
              $\mathbf{z}^* \leftarrow \delta\alpha\mathbf{e}_1 + \frac{1}{t}\sum_{\tau=0}^{t} \mathbf{z}^{(\tau)}$
              $\alpha_R \leftarrow \alpha$.
              $\alpha \leftarrow \frac{1}{2}(\alpha_R + \alpha_L)$
          end if
      end for
  end while
  return $\mathbf{z}^*$
```

The ORACLE is implemented as follows.

Algorithm 3 Oracle for MMW-DA

```
Require: $\eta$, $\mathcal{L}$, $\alpha$.
  $z_0 \leftarrow \alpha, z_j \leftarrow 0 \quad \forall j = 1, \dots \mathfrak{m}$
  $j \leftarrow 1$.
  while $z_0 - \eta_{\mathcal{L}(j)} < -\eta_0$ and $j < m$ do
      $j \leftarrow j + 1$.
  end while
  if $z_0 - \eta_{\mathcal{L}(j)} < -\eta_0$ then return Fail.
  else
      $z_{\mathcal{L}(j)} \leftarrow 1$
      return z
  end if
```

5.6 Comparison to Previous Results

The MMW-DA Algorithm presented in the previous section has the following running time:

Theorem 5.6.1. *Algorithm 2 finds a δ-approximate solution of the problem* (2.15) *in time $O(\frac{1}{\delta^2}\mathcal{N}^3 \log \mathcal{N})$, where $\mathcal{N} = \min\{\mathfrak{m} + \mathfrak{n}, N\}$.*

Proof. By Theorem 4.3.2, for each guess of optimal α, a total of $T = \frac{8\rho^2 R^2 \log(n)}{\delta^2 \alpha^2}$ iterations are needed to achieve a δ-approximate solution. In this case, $R = 1$ and $\rho \leq 2\alpha$, so each inner run requires $T = \frac{32 \log(N)}{\delta^2}$ iterations.

The cost of each of these iterations is dominated by the computation of the vector η, defined in terms of matrix hyperbolic functions (or exponentials). With the efficient matrix powering scheme of Section 5.4.2, all the required values can be approximately obtained in $O((\mathfrak{m} + \mathfrak{n})N^2 + (\mathfrak{m}^2 + \mathfrak{n}^2)N)$, or in $O((\mathfrak{m} + \mathfrak{n})^3)$ if working with the kernelized version.

If instead we use the eigenvalue approach of Section 5.4.1, each iteration is dominated by an eigendecomposition, which, without further exploiting the structure of the matrices, has a computational cost of $O(\min\{\mathfrak{m} + \mathfrak{n}, N\}^3)$, since the number of nonzero eigenvalues of $\mathbf{M}(\mathbf{z})$ can be at most the rank of $\mathbf{X}$. Since we have a relatively accurate starting eigenvector basis and set of eigenvalues, the constant hidden in this bound (the number of steps required by the eigenvalue algorithm) should be small.

Therefore, the overall complexity of the algorithm is $O(\frac{N^3 \log(N)}{\delta^2})$ for the standard version and $O(\frac{(\mathfrak{m}+\mathfrak{n})^3 \log(\mathfrak{m}+\mathfrak{n})}{\delta^2})$ for the kernelized version. Since we know *a priori* the order betweem $\mathfrak{m} + \mathfrak{n}$ or N, we naturally choose the corresponding version of the problem corresponding to the smallest of these. □

In comparison, the original algorithm by Cortes and Mohri [8] obtains an ϵ-accurate solution in at most $4\sqrt{(1+\epsilon)r\log r}/\epsilon$ iterations each of which has a computational cost of $O\big((\log p)(\mathfrak{m}+\mathfrak{n})^3\big)$, where p is the maximum power used in the smooth approximation of the objective function and $r = \max_{\mathbf{z}} \operatorname{rank}(\mathbf{M}(\mathbf{z}))$. In order to yield the desired number of iterations, the value p must be at least $\frac{(1+\epsilon)\log r}{\epsilon}$.

Since our algorithm would also benefit is an analogous manner from $r < (\mathfrak{m}+\mathfrak{n})$ in the way they do, let us suppose for simplicity that $r \approx \mathfrak{m}+\mathfrak{n} = \mathcal{N}$, which is a -not unlikely- worst case scenario. In that case, their cost per iteration has cost $O\big(\mathcal{N}^3 \log(\frac{1}{\epsilon}\log\mathcal{N})\big)$, so both ours and their algorithm have cost per iteration of $\tilde{O}(\mathcal{N}^3)$, but the hidden polylogarithmic factor in theirs is smaller than ours whenever $\epsilon > \mathcal{N}e^{-\mathcal{N}}$, which will almost surely be true except for unrealistic problems.

With respect to number of iterations, the term $\log\mathcal{N}$ in our algorithm cannot be reduced even if $r < \mathcal{N}$. Thus, our algorithm requires more iterations if $\epsilon < \frac{8\log\mathcal{N}}{\sqrt{r\log r}}$. As an example scenario, if the dimension of the training sample is $\mathcal{N} = 10^5$ and $r = 8\times 10^4$, the desired accuracy should be no smaller than 0.035 for our algorithm to require less iterations. We will extended this comparison, now from an empirical point of view, in the next Chapter.

Chapter 6

Experiments

In this chapter we report some empirical results obtained with the Matrix Multiplicative Weights algorithm for Domain Adaptation (MMW-DA) presented in the previous chapter. Since our objective was to have a fair comparison with the method used in [7], we also programmed all our routines in R, and in addition used their code for the Smooth Approximation algorithm (denoted SA-DA henceforth).

We show two sets of experiments, each in its own section below. The first set makes use of an artificially generated data set meant to mimick the hypothesis which make adaptation possible, as seen in the previous chapters. The second set corresponds to a real-life domain adaptation task from natural language processing. In both of these experiments we follow the methodology of Cortes and Mohri [7].

6.1 Artificial Data

For our first set of experiments, which is meant to test the efficiency and speed of our algorithm compared to SA-DA, we define a target domain by means of a single gaussian distribution with width $\sigma = 1$ and a mean vector generated randomly and uniformly on $[-1, 1]^N$. For the source domain, the distribution is comprised of a mixture of N gaussians, each generated as before, plus a 20% mass from the target gaussian. This last ingredient is meant to ensure a certain similarity between source and target domain, a condition which is naturally necessary for adaptation to be possible.

The labeling function is taken to be the same for source and target domain, and its simply given by $f_P(\mathbf{x}) = \sum_{i=1}^{N} |x_i|$. We fixed the dimension of the input space in $N = 200$ and varied the dimension of the source and target sampeles, $\mathfrak{m}$ and $\mathfrak{n}$ respectively, in order to test the scalability of the algorithm. We took in every trial $\mathfrak{m} < \mathfrak{n} < N$, which is a scenario commonly encountered in adaptation problems.

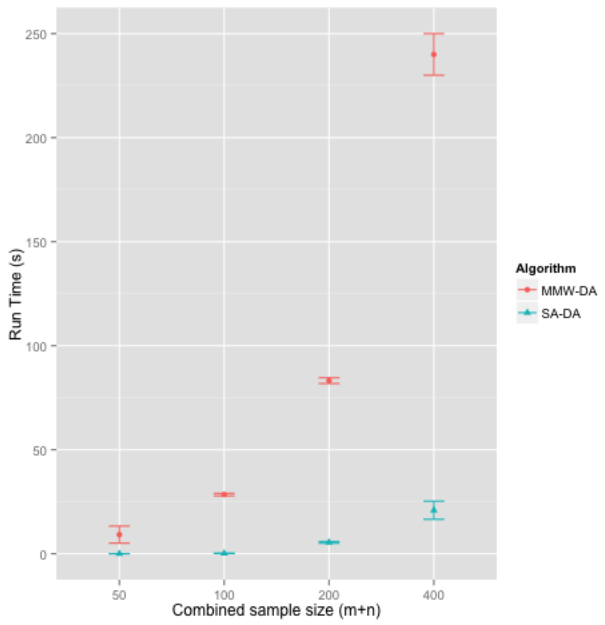

Figure 6.1: Average running time (± one standard deviation) over 20 realizations of the minimization algorithms for several sample sizes, for our algorithm (red/round points) and the smooth approximation algorithm (blue/squared points).

For this experiment, we used the same configuration of SA-DA that the authors use, and we took the number of iterations in their algorithm to be fixed at $T = 10$, which they claim empirically guarantess convergence. For our algorithm, we used an adaptive approximation error δ, which decreases in each succesive run until in reaches a minimum value of $\delta = 0.1$. The tolerance of the interval width for the binary search was set to this same value.

Figure 6.1 shows average running times with standard deviations over 30 realizations of three experiments: with sample sizes $m + n$ taking values 50, 100, 200 and 400 respectively. From this graph it is clear that even though for smaller problems the performance is similar, our algorithm scale worse than SA-DA. Running time for sample sizes beyond $\mathfrak{m} + \mathfrak{n} = 500$ become prohibitely long for our method. The main reason for this was that for this type of experiment the optimal value for large sample sizes tended to be small compared to the parameter ρ. Since the number of iterations our algorithm requires depends on the ratio $\frac{\rho^2}{\alpha^2}$, this caused excessively long runs. Note, however, in Figure 6.2, that our algorithm almost always finds a solution with a lower objective value than SA-DA. This could be explained by the way each algorithm handles *approximation* to the optimal value. This suggests that our algorithm's speed could be taken closer to that of SA-DA by allowing for less accurate solutions. However, the simultaneous changes imposed by the binary search on α and δ do not provide an obvious way to fine-tune this accuracy.

In the same setting as before, but this time fixing the source sample sizes in $\mathfrak{m} = 100$ and letting the unlabeled sample size $\mathfrak{n}$ vary, we now run the full adaptation experiment by using

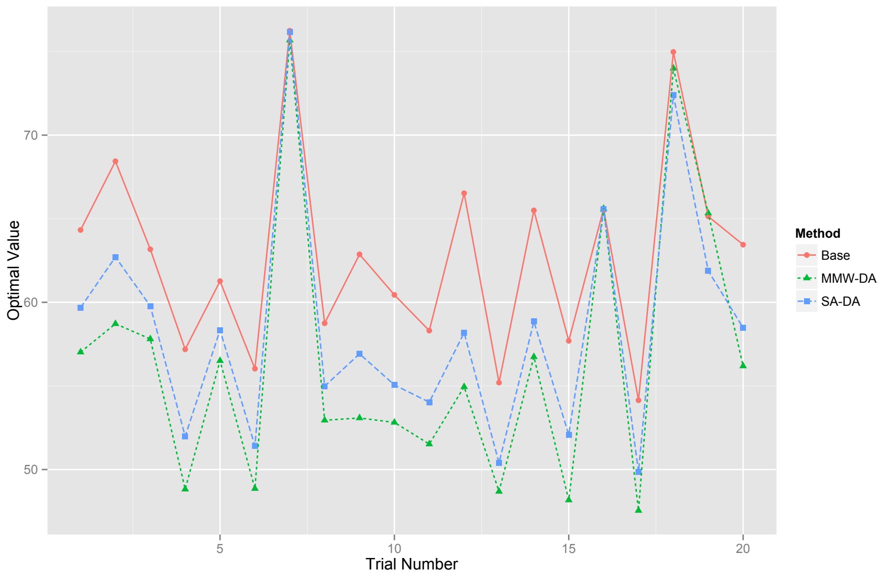

Figure 6.2: Optimal value of the problem (norm of the matirx $\mathbf{M}(\mathbf{z})$) obtained for a fixed sample size with $\mathfrak{m} = 50, \mathfrak{n} = 150$. The different lines corresponds to the value for the matrices built with the weight vector $\mathbf{z}$ obtained from: naive uniform distribution (red/straight), our method (green, dotted) and the smooth approximation algorithm (blue/dashed).

weighted Kernel Ridge Regression (wKRR) to predict the labels of a test set drawn from the target distribution. We used $\sigma = N$ and a ridge parameter $\lambda = 0.02$, chosen to provide the best performance when training on the target domain. This learning algorithm incorporates the weight vector obtained from the discrepancy minimization problem by modifiying the loss of each trianing point according to these weights. The results are shown in Figure 6.3. There, we plot the root mean squared error (RMSE) as a function of the unlabeled sample size $\mathfrak{n}$.

This graph shows how the reweighted solution significantly outperforms the baseline solution that does not use discrepancy minimization. The results are as expected: as the size of the unlabeled sample increases, the better knowledge we acquire of the target dsitribution P, and thus the better we are able to ponder the training data to include this information in the regression. This naturally yields improved predicting performance.

6.2 Adaptation in Sentiment Analysis

For the second experiment, we now turn to a problem with real-life data where domain adaptation is likely to be needed. Again, to achieve comparable results, we used the same data as [7], which has in fact been used in various publications concerning domain adaptation.

The data consists of product reviews, grouped by category, with a text entry and a rating

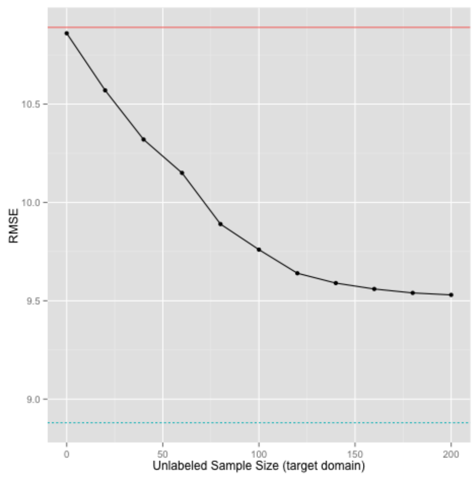

Figure 6.3: Result of KRR after using our algorithm for re-weighting the training set. The plot shows root mean squared error (RMSE) as a function of the unlabeled sample size from the target domain. The top line shows the baseline results of this learning task when no re-weighting is used, and the bottom line shows the RMSE from training directly on labeled dated from the target domain.

label ranging from 1 to 5. The processed data set includes a vector of unigrams, bigrams and trigrams for each review, along with its corresponding tag. Since the performance of discrepancy minimization used here has already been proven to be effective in [7], our objective was mainly to compare efficiency between our algorithm and the one based on smooth approximation. Thus, we only use one pair of tasks as source and target domains, and focus on experimenting with different sample sizes for this combination. From the four domains (`books`, `dvd`, `electronics` and `kitchen`), we chose `books` as the source and `kitchen` as the target domain, since Cortes and Mohri report good results for this combination.

Most of the setting-up of the experiment we replicate from that same paper, although for efficiency reasons, we used as feature vectors only the counts of the top 3,000 unigrams and bigrams, which we normalized per feature, across both tasks, to yield mean zero and variance one. Also, analaogously as we proceeded in the previous experiment, we try to enforce the hyopthesis of relative similarity between domains by defining the source empirical distribution from a mixture of 500 labeled points from `books` and 200 from `kithcen`. Thus, throughout our experiments, the labeled training set size is fixed at $\mathfrak{m} = 700$. Additionally, we simulate continuous-valued labels by fitting the discrete rating values by using kernel regression, with width $\sigma = N$. This fitted values are used in the algorithm as the labels.

We ran 20 trails for each configuration of the experiment, varying the unlabeled sample size $\mathfrak{n}$ between 300 and 800. Beyond $\mathfrak{n} = 1000$, our algorithm became prohibitely slow or caused crashes. It did manage to solve, however, problems of size larger than those for which the state-of-the-art general SDP solver SeDuMi failed, as reported in [7]. The results are shown in Figure 6.4, where average values of the root mean squared error (plus-minus standard error) are shown for our algorithm and for SA-DA.

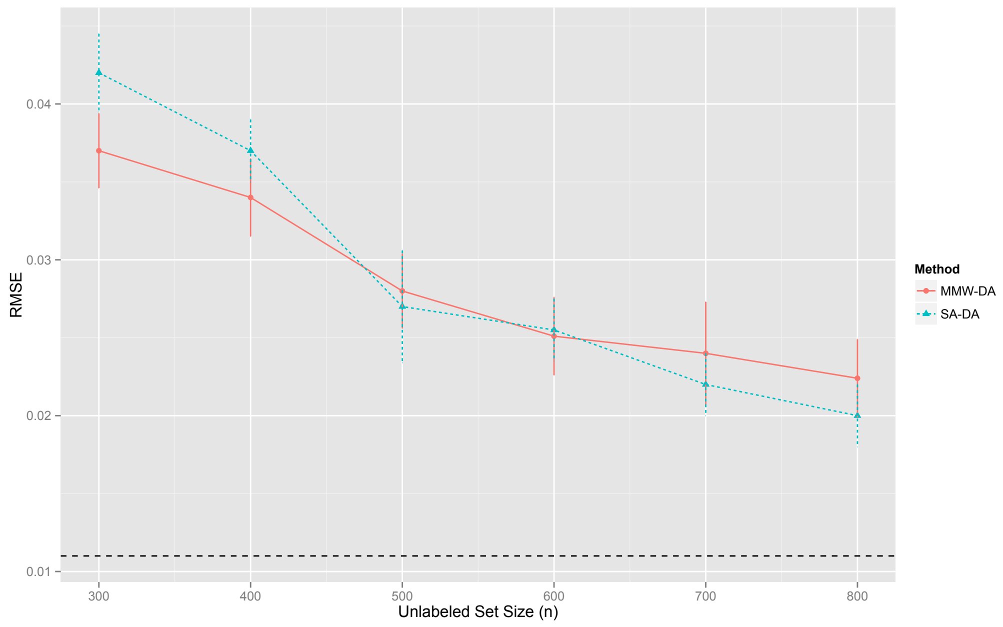

Figure 6.4: Performance improvement on the RMSE for the sentiment analysis adaptation tasks. The plot shows RMSE as a function of the unlabeled sample size used in the discrepancy minimization problem. The continuous line corresponds to our method, while the dotted line corresponds to SA-DA. The horizontal dashed line shows the error obtained when training directly on the target domain.

Our results, although on a different scale probably due to a different type of normalization, show a very similar behavior to what was obtained in [7]. Again, the adaptation task was carried out successfully, and the larger the unlabeled set from the target domain that was used, the better accuracy obtained with the weighted KRR.

Although similar in performance, our algorithm again showed considerably slower running times throughout the realizations of this experiment. As in the previous experiment, SA-DA outperforms our algorithm in this sense, and this difference becomes more significant as sample size increases.

Again, we ran into a similar obstacle as with the first experiment. Even though our algorithm consistently found feasible solutions for the SDP problem (2.15) with a lower optimal value than SA-DA, this price is payed for with a slower running time. For a problem where the discrepancy minimization is only an intermediate step - such as this one - a less accurate but faster solver might be desirable, an idea portrayed by the fact that the accuracy of the optimization task which used the optimal solution found by our algorithm differed only slightly from that found by SA-DA, and was only better for small sample sizes. Unfortunately, the current form of our algorithm does not allow for an easy manipulation of the accuracy-speed trade-off.

The results from this experiment and the one from the previous section show that our algorithm does provide a *useful* solution to the discrepancy minimization problem of domain adaptation, although it does so in less-than-desirable running times. However, these results, and their comparison to those delivered by the smooth approximation algorithm of Cortes and Mohri, also suggest that our method would greatly benefit from a tolerance for less accurate solutions. In addition, heuristics such as forcing sparsification of vectors and matrices used in the computation, as done in SA-DA [7], could also improve the efficiency of our algorithm. These are currently directions of our research and we expect to modify our algorithm accordingly in future work.

Chapter 7

Conclusions

In this thesis we explored the use of the Matrix Multiplicative Weights algorithm in the context of domain adaptation. After providing a brief review of the key features of this problem, in addition to the fundamental ideas behind semidefinite programming and the family of iterative weight-update learning algorithms, we showed how these three topics become strongly interlaced through a discrepancy minimization problem.

Based on a template of a primal-dual algorithm for solving semidefinite programs, we showed how to tailor the Matrix Multiplicative Weights algorithm for this optimization problem, which arises when trying to minimize the dissimilarity between the distributions of source and target domains. We provided an explicit and efficient way to implement each of its components, including an effortless ORACLE.

Even though our implementation tried to exploit the structure of the matrices involved in adaptation as much as possible, the empirical results show that there is still large room for improvements. Minor heurisitic modifications to the algorithm could be added in order to perform a smarter binary search, to force sparsification in intermediate computations and to solve other computations only approximately. An improvement on the ORACLE to allow for information and computations to be recovered even in case of failure is also a promising direction.

Despite all these potential improvements, the results obtained in this work suggest that MMW is not the best approach for domain adaptation. Even though it is a conceptually interesting algorithm and provides state-of-the-art complexities for many combinatorial optimization problems, its use for more general SDPs, such as that of discrepancy minimization, does not deliver equally admirable results. The main reason for this can be traced to the $\frac{1}{\epsilon^2}$ factor that appears in the bound for the iterations, which is intrinsic to most online weight-updating methods that generalize the weighted majority algorithm. This squared dependecy on the accuracy makes the MMW approach handicapped *a priori* when compared to the Smooth Approximation method proposed in [7].

Nonetheless, it must be noted that other algorithms, such as that of Cortes and Mohri, could benefit from many of the techniques derived here which exploit the structure of the matrices of the problem to provide more efficient computations. These modifications could be added to such algorithms to improve their computational cost per iteration and deliver even better results. Another candidate for such kind of enhancements would be a projected-subgradient method, which could be made very efficient in terms of work-per-iteration. An interesting direction for future work would be to compare these three optimized methods *vis-à-vis* in terms of efficiency and accuracy.

In summary, this thesis provides a stepping-stone towards researching other possible efficient approaches to the discrepancy minimization problem. It also provides a brief introduction to the theoretical concepts required to comprehend the wide range of subtleties that hide behind that seemingly *innocent* optimization problem. Last but not least, it will have hopefully brought the reader to share the author's fascination by how different branches of mathematics and computer science interweave in such an alluring pattern around this one problem.

Appendix A

Proofs of Generalization Bounds

The two following proofs are adapted from [8]. In that article, it is proven, using the generalized Bregman divergence, that

$$2\lambda\|h' - h\|_K^2 \leq \big(\mathcal{L}_{\hat{P}}(h, f_P) - L_{\hat{Q}}(h, f_Q)\big) - \big(\mathcal{L}_{\hat{P}}(h', f_P) - \mathcal{L}_{\hat{Q}}(h', f_Q)\big) \tag{A.1}$$

we will use this bound for the following two proofs.

Proof of Theorem 2.3.3. Let h_0 be an arbitrary element of H. The right-hand side of (A.1) can be decomposed as follows

$$\begin{aligned}2\lambda\|h' - h\|_K^2 \leq\; &\big(\mathcal{L}_{\hat{P}}(h, f_P) - \mathcal{L}_{\hat{P}}(h, h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h', h_0) - \mathcal{L}_{\hat{P}}(h', h_0)\big)\\ &+ \big(\mathcal{L}_{\hat{P}}(h, h_0) - \mathcal{L}_{\hat{Q}}(h, h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h', h_0) - \mathcal{L}_{\hat{Q}}(h', h_0)\big)\\ &+ \big(\mathcal{L}_{\hat{Q}}(h, h_0) - \mathcal{L}_{\hat{Q}}(h, f_Q)\big) - \big(\mathcal{L}_{\hat{Q}}(h', h_0) - \mathcal{L}_{\hat{Q}}(h', f_Q)\big)\end{aligned} \tag{A.2}$$

Using the μ-admissibility of the loss and the triangle inequality we obtain

$$\big(\mathcal{L}_{\hat{P}}(h, f_P) - \mathcal{L}_{\hat{P}}(h, h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h', f_P) - \mathcal{L}_{\hat{P}}(h', h_0)\big) \leq 2\mu E_{x\sim\hat{P}}\big[|f_P(x) - h_0(x)|\big]$$
$$\big(\mathcal{L}_{\hat{Q}}(h, f_Q) - \mathcal{L}_{\hat{Q}}(h, h_0)\big) - \big(\mathcal{L}_{\hat{Q}}(h', f_Q) - \mathcal{L}_{\hat{Q}}(h', h_0)\big) \leq 2\mu E_{x\sim\hat{Q}}\big[|f_Q(x) - h_0(x)|\big]$$

Since h_0 is in H, the other terms can be bounded in terms of the discrepancy:

$$\big(\mathcal{L}_{\hat{P}}(h, h_0) - \mathcal{L}_{\hat{Q}}(h, h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h', h_0) - \mathcal{L}_{\hat{Q}}(h', h_0)\big) \leq 2\text{disc}(\hat{P}, \hat{Q})$$

Thus,

$$2\lambda\|h' - h\|_K^2 \leq 2\big(\text{disc}(\hat{P}, \hat{Q}) + \mu E_{x\sim\hat{P}}[|f_P(x) - h_0(x)|] + \mu E_{x\sim\hat{Q}}[|f_Q(x) - h_0(x)|]\big)$$

Since the inequality holds for all h_0, we can write

$$\begin{aligned}\lambda\|h' - h\|_K^2 \leq\; &\text{disc}(\hat{P}, \hat{Q})\\ &+ \mu \min_{h_0\in H}\big\{\max_{x\in\text{supp}(\hat{P})} E_{x\sim\hat{P}}[|f_P(x) - h_0(x)|] + \max_{x\in\text{supp}(\hat{Q})} E_{x\sim\hat{Q}}[|f_Q(x) - h_0(x)|]\big\}\end{aligned}$$

Therefore

$$2\lambda\|h'-h\|_K^2 \leq 2\mathrm{disc}(\hat{P},\hat{Q}) + 2\mu\eta_H(f_P,f_Q) \tag{A.3}$$

By the reproducing property of the space, we have that for any $x \in \mathcal{X}$, $(h'-h)(x) = \langle h'-h, K(x,\cdot)\rangle$, thus, for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$,

$$\begin{aligned}\big|L(h'(x),y) - L(h(x),y)\big| &\leq \mu|h'(x)-h(x)| \\ &\leq \mu r\|h'-h\|_K \\ &\leq \mu r\sqrt{\frac{\mathrm{disc}(\hat{P},\hat{Q}) + 2\mu\eta_H(f_P,f_Q)}{\lambda}} \qquad \text{(by using (A.3))}\end{aligned}$$

which is precisely what we wanted to prove. □

Proof of Theorem 2.3.4. For any arbitrary $h_0 \in H$, we decompose as in the previous, the right-hand side of (A.1) as:

$$\begin{aligned}2\lambda\|h'-h\|_K^2 \leq &\big(\mathcal{L}_{\hat{P}}(h,f_P) - \mathcal{L}_{\hat{P}}(h,h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h',h_0) - \mathcal{L}_{\hat{P}}(h',h_0)\big) \\ &+ \big(\mathcal{L}_{\hat{P}}(h,h_0) - \mathcal{L}_{\hat{Q}}(h,h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h',h_0) - \mathcal{L}_{\hat{Q}}(h',h_0)\big) \\ &+ \big(\mathcal{L}_{\hat{Q}}(h,h_0) - \mathcal{L}_{\hat{Q}}(h,f_Q)\big) - \big(\mathcal{L}_{\hat{Q}}(h',h_0) - \mathcal{L}_{\hat{Q}}(h',f_Q)\big)\end{aligned} \tag{A.4}$$

Using the definition of the squared loss and the triangle inequality we obtain

$$\begin{aligned}\mathcal{L}_{\hat{P}}(h,f_P) - \mathcal{L}_{\hat{P}}(h,h_0) &= E_{x\sim\hat{P}}\big[(h_0(x)-f_P(x))(2h(x)-f_P(x)-h_0(x))\big] \\ \mathcal{L}_{\hat{P}}(h',f_P) - \mathcal{L}_{\hat{P}}(h',h_0) &= E_{x\sim\hat{P}}\big[(h_0(x)-f_P(x))(2h'(x)-f_P(x)-h_0(x))\big]\end{aligned}$$

Note that hidden in the right hand side of each of these inequalities is a term $\mu = 1$, from the $\mu-$admissibility of the squared loss. Taking the difference of these two equalities yields

$$\begin{aligned}&\big(\mathcal{L}_{\hat{P}}(h,f_P) - \mathcal{L}_{\hat{P}}(h,h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h',f_P) - \mathcal{L}_{\hat{P}}(h',h_0)\big) \\ &\qquad = 2\,E_{x\sim\hat{P}}\big[(h_0(x)-f_P(x))(h(x)-h'(x))\big]\end{aligned} \tag{A.5}$$

Analogously, for $\hat{Q}$, we obtain

$$\begin{aligned}&\big(\mathcal{L}_{\hat{Q}}(h,h_0) - \mathcal{L}_{\hat{Q}}(h,f_Q)\big) - \big(\mathcal{L}_{\hat{Q}}(h',h_0) - \mathcal{L}_{\hat{Q}}(h',f_Q)\big) \\ &\qquad = -2\,E_{x\sim\hat{Q}}\big[(h_0(x)-f_Q(x))(h(x)-h'(x))\big]\end{aligned} \tag{A.6}$$

Now, h_0 is in H, so by the definition of the discrepancy, we have

$$\big(\mathcal{L}_{\hat{P}}(h,h_0) - \mathcal{L}_{\hat{Q}}(h,h_0)\big) - \big(\mathcal{L}_{\hat{P}}(h',h_0) - \mathcal{L}_{\hat{Q}}(h',h_0)\big) \leq 2\mathrm{disc}(\hat{P},\hat{Q}) \tag{A.7}$$

Thus, replacing (A.5), (A.6) and (A.7) in (A.4), we have that $2\lambda\|h'-h\|_K^2 \leq 2\mathrm{disc}(\hat{P},\hat{Q}) + 2\Delta$, where

$$\Delta = E_{x\sim\hat{P}}\big[(h_0(x)-f_P(x))(h(x)-h'(x))\big] - E_{x\sim\hat{Q}}\big[(h_0(x)-f_Q(x))(h(x)-h'(x))\big]$$

By the reproducing property of the space, the identity $h(x) - h'(x) = \langle < h - h', K(x, \cdot)\rangle_K$ holds for any $x \in \mathcal{X}$. In view of that, Δ can be expressed and bounded as follows:

$$\begin{aligned}\Delta &= \left\langle h - h', E_{x\sim\hat{P}}[(h_0(x) - f_P(x))K(x,\cdot)] - E_{x\sim\hat{Q}}[(h_0(x) - f_Q(x))K(x,\cdot)]\right\rangle \\ &\leq \|h - h'\|_K \left\| E_{x\sim\hat{P}}[(h_0(x) - f_P(x))K(x,\cdot)] - E_{x\sim\hat{Q}}[(h_0(x) - f_Q(x))K(x,\cdot)] \right\|_K\end{aligned}$$

Since this inequality holds for all $h_0 \in H$, we can write $\Delta \leq \|h - h'\|_K \delta_H(f_P, f_Q)$. Thus,

$$2\lambda\|h' - h\|_K^2 \leq 2\text{disc}(\hat{P}, \hat{Q}) + 2\|h - h'\|_K \delta_H(f_P, f_Q)$$

Regrouping and solving this second-order inequality for $\|h' - h\|_K$ yields

$$\|h - h'\|_K \leq \frac{1}{2\lambda}\left(\delta_H(f_P, f_Q) + \sqrt{\delta_H(f_P, f_Q)^2 + 4\lambda\text{disc}(\hat{P}, \hat{Q})}\right) \tag{A.8}$$

For any $(x, y) \in \mathcal{X} \times \mathcal{Y}$, using the definition of the squared loss and the reproducing property, we can write

$$\begin{aligned}|L(h'(x), y) - L(h(x), y)| &= |h'(x) - y)^2 - (h(x) - y)^2| \\ &= |(h'(x) - h(x))(h'(x) - y + h(x) - y)| \\ &\leq 2\sqrt{M}|h'(x) - h(x)| \\ &= 2\sqrt{M}|\langle h' - h, K(x\cdot)\rangle| \leq 2\sqrt{M}r\|h' - h\|_K\end{aligned}$$

So finally, bounding $\|h' - h\|$ in the inequality above with (A.8), we obtain the conclusion of the theorem:

$$\left|L(h'(x), y) - L(h(x), y)\right| \leq \frac{r\sqrt{M}}{\lambda}\left(\delta_H(f_P, f_Q) + \sqrt{\delta_H^2(f_P, f_Q) + 4\lambda\text{disc}(\hat{P}, \hat{Q})}\right)$$

□

Bibliography

[1] S. Arora, E. Hazan, and S. Kale, *The multiplicative weights update method: a meta algorithm and applications*, tech. rep., 2005.

[2] S. Arora and S. Kale, *A combinatorial, primal-dual approach to semidefinite programs*, in In Proceedings of the thirty-ninth annual ACM symposium on Theory of Computing, ACM Press, 2007, pp. 227–236.

[3] P. Bartlett and S. Mendelson, *Rademacher gaussian complexities: Risk bounds and structural results*, Journal of Machine Learning Research, 3 (2002).

[4] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, *Analysis of representations for domain adaptation*, in Advances in Neural Information Processing Systems, vol. 20, MIT Press, 2007.

[5] J. Blitzer, K. Crammer, F. Pereira, and J. Wortman, *Learning bounds for domain adaptation*, in Advances in Neural Information Processing Systems 2007, MIT Press, 2008.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[7] C. Cortes and M. Mohri, *Domain adaptation in regression*, in Proceedings of the 22nd International Conference on Algorithmic Learning Theory, vol. 6925, Springer, October 2011, pp. 308–323.

[8] ——, *Domain adaptation and sample bias correction theory and algorithm for regression*, in Theoretical Computer Science (ALT2011 special issue), 2013.

[9] S. Della Pietra, V. Della Pietra, L. Mercer, and S. Roukos, *Adaptive language modeling using minimum discriminant estimation.*, in HLT '91: workshop on Speech and Nat. Lang., 1992.

[10] M. Frank and P. Wolfe, *An algorithm for qudratic programming*, Naval Research Logistics, 3 (1956), pp. 95–110.

[11] D. Garber and E. Hazan, *Approximating semidefinite programs in sublinear time*, in NIPS 2011, 2011.

[12] E. Hazan, *Sparse approximate solutions to semidefinite programs*, in Proceedings of the 8th Latin American conference on Theoretical Information, Springer-Verlag, 2008, pp. 306–316.

[13] J. Jiang and C. Zhai, *Instance weighting for domain adaptation in nlp*, in Proceedings of ACL 2007, 2007, pp. 264–271.

[14] B. Kulis, S. Sra, S. Jegelka, and I. S. Dhillon, *Scalable semidefinite programming using convex perturbations*, tech. rep., UTCS, 2007.

[15] A. S. Lewis, *Eigenvalue optimization*, Acta Numerica, 5 (1996), pp. 149–190.

[16] N. Littlestone and M. Warmuth, *The weighted majority algorithm*, Information and Computation, 108 (1994), pp. 212 – 261.

[17] L. Lovász, *Semidefinite programs and combinatorial optimization*, tech. rep., Microsoft Research, 1995.

[18] Y. Mansour, M. Mohri, and A. Rostamizadeh, *Domain adaptation with multiple sources*, in NIPS 2008, 2008, pp. 1041–1048.

[19] ——, *Domain adaptation: Learning bounds and algorithms*, in Proceedings of COLT 2009, Omnipress, 2009.

[20] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, MIT Press, 2012.

[21] C. Moler and C. Van Loan, *Ninteen dubious ways to compute the exponential of a matrix*, SIAM Review, 20 (1978), pp. 801–836.

[22] ——, *Ninteen dubious ways to compute the exponential of a matrix, twenty-five years later*, SIAM Review, 45 (2003), pp. 3–49.

[23] Y. Nesterov and A. Nemirovsky, *Interior point polynomial methods in convex programming*, vol. 13, SIAM, 1994.

[24] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, 2nd ed., 2006.

[25] M. Overton, *On minimizing the maximum eigenvalue of a symmetric matrix*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 256–268.

[26] S. Shalev-Shwartz, Y. Singer, and A. Ng, *Online and batch learning of pseudo-metrics*, in Proceedings of the Twenty First International Conference on Machine Learning (ICML-04), 2004, pp. 94–101.

[27] M. Sion, *On general minimax theorems*, Pacific J. Math, 8 (1958), pp. 171–176.

[28] K. Tsuda, G. Rätsch, and M. K. Warmuth, *Matrix exponentiatied gradient updates for on-line learning and bregman projection*, Journal of Machine Learning Research, 6 (2005), pp. 995–1018.

[29] L. Valiant, *A theory of the learnable*, ACM Press, New York, 1984.

[30] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[31] ——, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[32] V. Vapnik and A. Chervonenkis, *On the unifrom convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.