Optimal Data Distributions in Machine Learning

Thesis by

Carlos R. González

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2015

(Defended May 22, 2015)

For my little brother Daniel,

thanks for all the joy you bring. May this inspire you!

Acknowledgements

This work would not be possible with the help and support of very important people in my life.

I would like to thank my advisor, Dr. Yaser Abu-Mostafa, for his invaluable guidance, not only in my research but also in so many aspects of life. I am very thankful to have received hints of his wisdom throughout these five years. The road was bumpy in many turns we did not expect, but your perseverance helped me through.

In the same way, I was lucky to have not only one advisor, but two advisors. Thank you so much Dr. Pietro Perona for all your help, both in and outside of academic life. Your never-ending curiosity and interest for so many topics will always be an inspiring example in my research career.

To my family, thank you for your unconditional support and love throughout my *whole* life. The strong values and ethics you instilled in me have payed great dividends. I hope to continue making you proud.

To my friends, thanks for making life happier, especially in the tough times.

Abstract

In the first part of the thesis we explore three fundamental questions that arise naturally when we conceive a machine learning scenario where the training and test distributions can differ. Contrary to conventional wisdom, we show that in fact mismatched training and test distribution can yield better out-of-sample performance. This optimal performance can be obtained by training with the dual distribution. This optimal training distribution depends on the test distribution set by the problem, but not on the target function that we want to learn. We show how to obtain this distribution in both discrete and continuous input spaces, as well as how to approximate it in a practical scenario. Benefits of using this distribution are exemplified in both synthetic and real data sets.

In order to apply the dual distribution in the supervised learning scenario where the training data set is fixed, it is necessary to use weights to make the sample appear as if it came from the dual distribution. We explore the negative effect that weighting a sample can have. The theoretical decomposition of the use of weights regarding its effect on the out-of-sample error is easy to understand but not actionable in practice, as the quantities involved cannot be computed. Hence, we propose the Targeted Weighting algorithm that determines if, for a given set of weights, the out-of-sample performance will improve or not in a practical setting. This is necessary as the setting assumes there are no labeled points distributed according to the test distribution, only unlabeled samples.

Finally, we propose a new class of matching algorithms that can be used to match the training set to a desired distribution, such as the dual distribution (or the test distribution). These algorithms can be applied to very large datasets, and we show how they lead to improved performance in a large real dataset such as the Netflix dataset. Their computational complexity is the main reason for their advantage over previous algorithms proposed in the covariate shift literature.

In the second part of the thesis we apply Machine Learning to the problem of behavior recognition. We develop a specific behavior classifier to study fly aggression, and we develop a system that allows analyzing behavior in videos of animals, with minimal supervision. The system, which we call CUBA (Caltech Unsupervised Behavior Analysis), allows detecting movemes, actions, and stories from time series describing the position of animals in videos. The method summarizes the data, as well as it provides biologists with a mathematical tool to test new hypotheses. Other benefits of CUBA include

finding classifiers for specific behaviors without the need for annotation, as well as providing means to discriminate groups of animals, for example, according to their genetic line.

Contents

Acknowledgements **iv**

Abstract **v**

I Optimal Data Distributions in Machine Learning **1**

1 Introduction **2**

1.1 Overview 3

1.2 Literature overview 4

1.3 The learning setup 5

2 Is it better to have $P_R = P_S$? **7**

2.1 Empirical results in the classification setting 8

2.1.1 Fixing the training distribution 10

2.1.2 Fixing the test distribution 11

2.2 Empirical and analytic results in the regression setting 13

3 The dual distribution **20**

3.1 Discrete input spaces 20

3.2 The continuous case 23

3.2.1 Analytic condition for the dual distribution 24

3.2.2 Dual distribution examples 27

3.3 Variability of the dual distribution 29

3.3.1 Asymptotic behavior 30

3.3.2 Effect of noise and complexity 32

3.4 Using the dual distribution in a practical setting 33

3.5 Computational and implementation details 36

3.6 Differences with active learning 39

4 To weight or not to weight **41**

4.1 What makes weighting work sometimes only? . . . 41

4.1.1 The effective sample size . . . 42

4.1.2 Weighting vs sampling . . . 44

4.1.3 Decomposition of the weighting effect . . . 49

4.2 The algorithm: Targeted matching . . . 50

4.3 Experimental results . . . 53

4.3.1 Results on the Netflix dataset . . . 53

4.3.2 Results on further benchmark datasets . . . 55

5 A novel class of matching algorithms **57**

5.1 Previous algorithms . . . 57

5.1.1 Indirect ratio estimation via KDE . . . 57

5.1.2 Logistic regression methods . . . 58

5.1.3 Kernel mean matching (KMM) . . . 59

5.1.4 Parametric models for the ratio: KLIEP, LSIF, RuLSIF, etc. . . . 59

5.1.5 Discrepancy minimization . . . 61

5.2 A new class of algorithms . . . 63

5.2.1 Hard matching . . . 63

5.2.2 Soft Matching . . . 67

5.2.3 Hard matching with slack variables . . . 70

5.2.4 Statistical approach . . . 72

5.2.5 Probabilistic approach . . . 74

II Behavior Analysis with Machine Learning **77**

6 Supervised behavior classification **78**

6.1 Pre-processing stage . . . 79

6.2 Learning algorithm . . . 80

6.3 Post-processing stage . . . 80

7 CUBA: Caltech Unsupervised Behavior Analysis **82**

7.1 Problem statement . . . 83

7.2 The method . . . 84

7.2.1 Detecting movemes . . . 84

7.2.2 Detecting actions . . . 87

7.2.3 Finding stories 88
7.3 Results on real datasets 90
7.3.1 CUBA in the Fear in Flies dataset 90
7.3.2 CUBA in the Fly Bowl dataset 101

Conclusion **106**

A An analytic learning setup **108**

Bibliography **111**

Part I

Optimal Data Distributions in Machine Learning

Chapter 1

Introduction

A basic assumption in learning theory is that the training and test sets are drawn from the same probability distribution. However, in many practical situations, this assumption about the training and test distributions does not hold. To illustrate this, take, for example, a recommender system. In this case, the system is trained with data gathered over a long period of time during which users rate certain items. Nevertheless, the system will be used and tested not only on old items that the user has not rated yet, but also on new items and new users. Due to changes in opinions, moods, trends, etc., with time, there is no guarantee that the distribution of the test data will be the same as that of the training data. Rather, a realistic assumption is to model this situation as one in which training and test distributions may differ. Other examples where this is the case have been reported in natural language processing [40] and speech recognition [15]. These systems are commonly trained by gathering speech samples from only a few individuals due to resource constraints. However, the system is tested later on the general population and hence training and test distributions are likely to differ. In this case it is not the effect of time that makes the two distributions differ, but rather the effect of having a biased sample. Other examples of differing distributions are commonly found in applications that involve experimental set-ups. Some of these set-ups can involve conditions such as lighting, temperature, etc., which vary from experiment to experiment, making the training and test distributions differ, as in [6].

The problem described above is referred to as dataset shift, and sometimes subdivided into covariate shift and sample selection bias, as described in [54]. Covariate shift occurs when the training distribution P_R of the input variable x is different from the test distribution P_S of the same variable x. Sample selection bias occurs when the sample used for training, is not representative of the overall distribution, due to some bias (intended or unintended) in the sampling process. This can be modeled as having $P_R(x) \neq P_S(x)$, but it also can be modeled with an additional random variable s called the sample selection variable. The selection variable indicates if a sample is included or not in the

training or test sets. In this case, if the overall distribution from which data is sampled is P, then $P_R(x) = P(x|s = 1)$.

There are various methods that have been devised to correct for this problem, and is part of the ongoing work on domain adaptation and transfer learning. Although adjustments to the theory become necessary, and numerous methods that will be described shortly have been devised to correct the problem of mismatched training and test distributions, the fact that the theory requires a matched distribution assumption to go through does not necessarily mean that matched distributions will lead to better performance; just that they lead to theoretically more predictable performance. However, the question of whether they do lead to better performance has not been addressed in the case of supervised learning, perhaps because of an intuitive expectation that the answer would be yes. Hence, in this part of the thesis the work is aimed to answer three fundamental questions in this learning scenario:

1. Is it better, in terms of out-of-sample performance, to have the training distribution P_R equal to the test distribution P_S?
2. If so, is it advantageous to apply weights to the training points to achieve this?
3. What is the algorithmic way to achieve it?

Answering these three questions led to the results reported in this part of the thesis.

1.1 Overview

The seemingly obvious answer to the first question is much more interesting than expected and is the topic of Chapter 2. In that chapter, we first show the simulation setup that led us to conceive the idea that using mismatched training and test distributions could lead to better performance in the supervised learning setting. We present both empirical and analytic results of this evidence.

We then introduce in Chapter 3 the formal notion of the *dual distribution*, which is the optimal training distribution to draw samples from, for the learning algorithm. We then formulate the optimization problem that allows us to find this dual distribution, and describe how to solve for it in the general case. We also analyze various properties and parameters that affect the dual distribution.

Chapter 4 describes the various effects that come into play when weights are used to change the original training distribution. On the one hand, training with data sampled from the dual distribution will improve performance, and so using weights that make the training distribution look like the dual distribution should be advantageous. On the other hand, weighting samples rather than sampling from the desired distribution are not equivalent. The former can have a negative effect, in terms of

an increase in the variance of our error estimates, which can also be viewed as an effective sample size reduction. Each learning scenario yields a different bottom line performance after adding up these effects, so that it is sometimes beneficial to use weights while other times it is not. Hence, we introduce an algorithm that determines when weighting is beneficial in a given practical scenario.

Chapter 5 introduces a class of algorithms that can be used to match the training distribution to any desired distribution, for example, the dual distribution. The algorithms we introduce have the advantage of selecting only the desired coordinates along which matching is desired. They are also efficient so they can be used in very large datasets. The efficiency issue was a constraint that we took into account since we conceived the method initially for recommender systems, which have very large datasets composed of ratings for thousands or millions of items, by thousands or millions of users.

1.2 Literature overview

As discussed, the problem of dataset shift has led to a substantial amount of work aimed at correcting the problem. All of the work assumes that the answer to the first question we pose is affirmative, and hence try to make $P_R = P_S$. The numerous methods can be roughly divided into four types [48].

The first type is referred to as instance weighting for covariate shift, in which weights are given to points in the training set, such that the two distributions become effectively matched. Some of these methods include discriminative approaches as in [13, 14]. To do this, these methods train a classifier that can distinguish between samples coming from the training distribution and samples coming from the test distributions. Other methods make assumptions regarding the source of the bias and explicitly model a selection bias variable [71]. Others try to match the two distributions in some Reproducing Kernel Hilbert Space as Kernel Mean Matching [38], while others use parametric models for the ratio of test to train densities, using the Kullback-Liebler divergence as in KLIEP (Kullback-Liebler importance estimation procedure) [67], or least squares deviation as in LSIF (least squares importance fitting) [42], among others. Additional approaches are given in [57, 27, 55, 64]. A detailed description of these methods is given in Chapter 5. All these methods rely on finding weights, which is not trivial as the actual distributions are not known. Furthermore, the addition of weights reduces the effective sample size of the training set, hurting the out-of-sample performance [61]. Another issue that comes up regards cross-validation, as it becomes necessary to match the distribution of the validation set to the test set. As some of the methods find weights that are only meaningful with respect to the rest of the sample, aggregating weights for different sets in K-fold type validation methods is no longer trivial. This issue is addressed in methods like importance weighting cross-validation [67]. On the theoretical side, learning bounds for the instance weighting setting are shown in [25, 72]. Further theoretical results in a more general setting of learning from different

domains are given in [10].

The second type of methods use self-labeling or co-training techniques so that samples from the test set, which are unlabeled, are introduced in the training set in order to match the distributions, and are labeled using the labeled data. A final model is then re-estimated with these new points. Some of these methods are described in [18, 46, 32]. A third approach is to change the feature representation, so that features are selected, discarded, or transformed in an effort to make training and test distributions similar. This idea is explored in various methods, including [16, 15, 11, 52], among many others. Finally, cluster based methods rely on the assumption that the decision boundaries have low density probabilities [34], and hence try to label new data in regions that are under-represented in the training set through clustering, as proposed in [17, 51]. For a more detailed review on these and other methods, refer to [48] and [64].

1.3 The learning setup

Before we answer the questions presented, we introduce the notation that will be used throughout the thesis and that describes the learning problem. Let $R = \{x_i, y_i\}_{i=1}^N$ be the training set, with $x_i \in \mathcal{X}$, and $y_i \in \mathcal{Y}$. $\mathcal{X}$ is known as the input space, and $\mathcal{Y}$ as the output space. We assume x_i are iid $\sim P_R$, where P_R is the training distribution. The objective of the learning algorithm is to find a hypothesis $h \in \mathcal{H}$ that is closest to the target function f, where $f : \mathcal{X} \to \mathcal{Y}$. $\mathcal{H}$ is known as the hypothesis set, where each $h \in \mathcal{H}$ is $h : \mathcal{X} \to \mathcal{Y}$. The notion of closeness to the target function is determined by a chosen loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The returned hypothesis by the learning algorithm is denoted by g. Finally, it is conventional to model the noisy data using a stochastic noise process ϵ, where ϵ_i is the corresponding realization for x_i, so that $y_i = f(x_i) + \epsilon_i$. In this framework, learning consists of solving the following optimization problem:

$$g = \arg\min_h \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i), y_i). \tag{1.1}$$

In parametric learning, as the name suggests, the hypothesis set $\mathcal{H}$ is parametrized by $\theta \in \mathcal{Z}^M$, where $\mathcal{Z}^M$ is the M-dimensional space where the parameters live. That is $\mathcal{H} = \{h(\cdot;\theta)|\theta \in \mathcal{Z}^M\}$. The learning algorithm outputs an optimal parameter $\theta^\star \in \mathcal{Z}^M$ given by

$$\theta^\star = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i;\theta), y_i), \tag{1.2}$$

and

$$g(x) = h(x;\theta^\star). \tag{1.3}$$

Now, let $x \sim P_S$ where P_S denotes the test distribution. In the usual learning setting $P_R = P_S$, but here, we consider precisely the scenario where $P_R \neq P_S$. Finally, for a point $x \sim P_S$, the out-of-sample error E_{out} is given by

$$E_{\text{out}}(x, R, f, \epsilon) = \ell(g(x), y). \tag{1.4}$$

The out-of-ample error E_{out} at x, depends not only on the point itself, but also on the dataset R, which in turn determines which $g \in \mathcal{H}$ is returned. Finally, the target function f and the noise process ϵ also affect this error (y depends on f and ϵ). The overall out-of-sample error is the expected value of the pointwise error,

$$E_{\text{out}} = \mathbb{E}_x[E_{\text{out}}(x, R, f, \epsilon)]. \tag{1.5}$$

Here $\mathbb{E}_x[\cdot]$ denotes the expected value of the expression with respect to variable x.

Chapter 2

Is it better to have $P_R = P_S$?

While great effort has been spent in the literature trying to match the training and test distributions, a thorough analysis of the need for matching has not been carried out. In particular, the first fundamental question we asked in Chapter 1 has not been answered: is it better to have training and test distributions matched, in terms of out-of-sample performance? As we will show shortly, the main contribution in this chapter is to show that mismatched distributions can in fact outperform matched distributions in the supervised learning setting, *regardless* of the specific target function. We first published the results of this chapter in [36].

This statement is not very surprising if we are under the active learning paradigm. Under such paradigm, training data is selected sequentially or in batches, making use of feedback obtained from the target function. The goal is to select the least amount of training data that will lead to the best performance. Therefore, the choice of such data may be tilted towards regions that best pin down the target function further, irrespective of the test distribution. Some methods belonging to the active learning paradigm exploit this idea by finding a 'design' distribution, from which the training data should be sampled. The idea is that training the algorithm with data sampled from the design distribution would result in better performance. Examples of these techniques are found in [70], [43], [63], [66], [62], and [58], among others. Hence, it is clear that the active learning paradigm makes use of unmatched distributions to improve performance.

In the supervised learning paradigm, however, the location of the training data is chosen without any feedback from the target function. Therefore, it is more surprising in this case that a data distribution that is mismatched to the test distribution would perform better. Recognizing that the system may perform better under a scenario of mismatched distributions can influence the need for, and the extent of, matching techniques, as well as the quantitative objective of matching algorithms.

In our analysis, we show that a mismatched distribution can be better than a matched distribution in two different directions in the supervised learning paradigm:

- For a given training distribution P_R, the best test distribution P_S can be different from P_R.
- For a given test distribution P_S, the best training distribution P_R can be different from P_S.

The justifications for these two directions, as well as their implications, are quite different. In a practical setting, the test distribution is usually fixed, so the second direction reflects the practical learning problem about what to do with the training data if it is drawn from a different distribution than that of the test environment. One of the ramifications of this direction is the new notion of a *dual distribution*. This is a training distribution P_R that is optimal to use when the test distribution is P_S, regardless of the specific target function. A dual distribution serves as a new objective for matching algorithms. Instead of matching the training distribution to the test distribution, it is matched to a dual of the test distribution, for optimal performance.

We cover both classification and regression settings in the sections that follow. The classification setting is analyzed through empirical results obtained via Monte Carlo simulations. We then present both empirical and analytic results in the regression setting.

2.1 Empirical results in the classification setting

Consider the learning scenario where the data set R used for training by the learning algorithm is drawn from probability distribution P_R, while the data set S that the algorithm will be tested on is drawn from distribution P_S. We show here that the performance of the learning algorithm in terms of the out-of-sample error can be better when $P_S \neq P_R$, averaging over target functions and data set realizations. The empirical evidence, which is statistically significant, is based on an elaborate Monte Carlo simulation that involves various target functions and probability distributions. The details of that simulation follow, and the results are illustrated in Figures 2.1 and 2.3.

We consider the input space $\mathcal{X} = [-1, 1]$. There is no loss of generality by limiting our domain as in any practical situation, the data has a finite domain and can be rescaled to the desired interval. We pick a one-dimensional space to have a better understanding in this simpler case, before generalizing to multiple dimensions. We run the learning algorithm for different target functions and different training and test distributions. We then average the out-of-sample error over a large number of data sets generated by those distributions and over target functions. Finally we compare the results for matched and mismatched distributions.

Distributions. We use 31 different probability distributions to generate R and S including a uniform distribution $U(-1, 1)$, ten truncated Gaussian distributions $\mathcal{N}^*(0, \sigma^2)$ where σ is increased in steps of 0.3, ten truncated exponential distributions $Exp^*(\tau)$ where τ is increased also in steps of 0.3, and ten truncated mixture of Gaussian distributions, such that $MG^*(\sigma) = \frac{1}{2}\left(\mathcal{N}^*(-0.5, \sigma^2) + \mathcal{N}^*(0.5, \sigma^2)\right)$,

with σ increased in steps of 0.25. By truncating the distributions we mean that we zero-out the probability distributions outside $\mathcal{X}$ and renormalize the densities accordingly. That is, if X has a truncated Gaussian distribution such that $X \sim \mathcal{N}^*(0, \sigma^2)$ and $\tilde{X}$ has a Gaussian distribution with $\tilde{X} \sim \mathcal{N}(0, \sigma^2)$, then

$$P(X \leq x) = \begin{cases} 0 & x \leq -1 \\ \frac{1}{Z} P(\tilde{X} \leq x) & -1 \leq x \leq 1 \\ 1 & x \geq 1 \end{cases} \tag{2.1}$$

where $Z = P(-1 \leq \tilde{X} \leq 1)$. Similarly, this applies for the truncated Exponential and Mixture of Gaussian distributions.

Data Sets. For each pair of probability distributions, we carry out the simulation generating 1,000 different target functions, running the learning algorithm, comparing the out-of-sample performance, and then averaging over 100 different data set realizations. That is, each point in Figures 2.1 and 2.3 is an average over 100,000 runs with the same pair of distributions but with different combinations of target functions and training and test sets. The sizes of the data sets are $N_R = 100$ and 300, and $N_S = 10,000$, where N_R and N_S are the number of points in the training and test sets R and S.

Target Functions. The target functions $f : [-1, 1] \to [-1, 1]$ were generated by taking the sign of a polynomial in the desired interval. The polynomials were formed by choosing at random one to five roots in the interval [-1,1]. This choice of target functions allows the decision boundaries to vary both in number and location in each realization. Hence, the results presented do not depend on a particular target function, so that the distributions cannot favor the regions around the boundaries, as these are changing in each realization.

Learning model The learning algorithm minimized a squared loss function. For the hypothesis set $\mathcal{H}$ we used linear functions of a non-linear transformation of the input space. The non-linear transformation used powers of the input variable up to the number of roots of the polynomial that describes the target function, plus a sinusoidal feature, which allows the model to learn a function that is close to, but not identical to, the target. That is, for every $h \in \mathcal{H}$

$$h(x; \theta) = \theta^T \phi_M(x), \tag{2.2}$$

with $\phi_M : \mathcal{X} \to \mathbb{R}^M$, and $\theta \in \mathbb{R}^M$, where

$$\phi_M(x) = [1 \quad x \quad x^2 \cdots \quad x^{M-2} \quad \sin(\pi x)]^T. \tag{2.3}$$

Out-of-sample error. The expected out-of-sample error E_{out} in this classification task is estimated using the test set generated according to each of the P_S with $N_S = 10,000$. The error at a

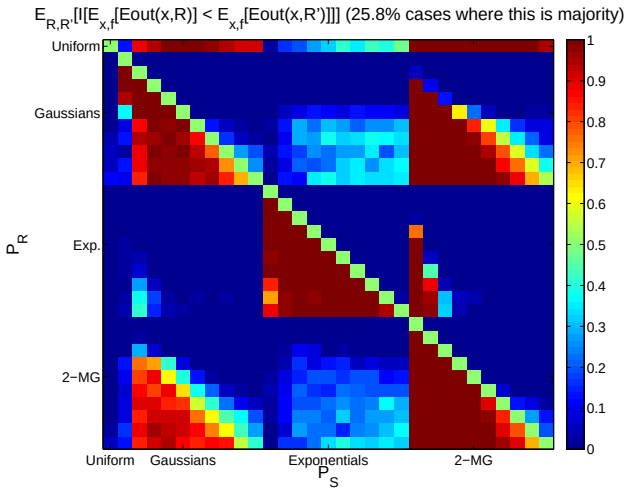


Figure 2.1: Summary of Monte Carlo Simulation. Plot indicates, for each combination of probability distributions $\mathbb{E}_{R\sim P_R, R'\sim P_S}[I[\mathbb{E}_{x\sim P_S, f}[E_{\text{out}}(x, R, f)] < \mathbb{E}_{f,x\sim P_S}[E_{\text{out}}(x, R', f)]]]$.

point $x \in \mathcal{X}$ depends not only on the point itself, but also on the training data set R used for learning which in turn affects the learned hypothesis $g \in \mathcal{H}$, and also depends on the target function f. We compute E_{out} using the misclassification 0-1 loss, that is

$$\mathbb{E}_{x,R}[E_{\text{out}}(x, R, f)] = \mathbb{E}_{x,R}[I[f(x) \neq g(x)]], \tag{2.4}$$

where $I[a]$ denotes the indicator function of expression a, $x \sim P_S$, and R is generated according to P_R.

2.1.1 Fixing the training distribution

Figure 2.1 summarizes the result of the simulation to answer the question in the first direction: for a given training distribution, is the best test distribution different? Each entry in the matrix corresponds to a pair of distributions P_R and P_S. We fix P_R, and evaluate the percentage of runs where using $P_S \neq P_R$ yields better out-of-sample performance than if $P_S = P_R$. That is, each entry corresponds to

$$\mathbb{E}_{R\sim P_R, R'\sim P_S}[I[\mathbb{E}_{f,x\sim P_S}[E_{\text{out}}(x, R, f)] < \mathbb{E}_{f,x\sim P_S}[E_{\text{out}}(x, R', f)]]]. \tag{2.5}$$

These results correspond to the case where $N_R = 100$.

The matrix is organized placing families of distributions together, with increasing order of standard deviation/time constant. The result that immediately stands out is that there is a significant number

of entries where more than 50% of the runs have better performance when mismatched distributions are used, as indicated by the yellow, orange, and red regions, which constitute 25.8% of all combinations of the probability distributions used.

A number of interesting patterns are worth noting in this plot. The first row, which corresponds to $P_R = U(-1, 1)$, falls under the category of better performance for mismatched distributions for almost any other P_S used. There is also a block structure in the plot, which is no accident due to the way the families of distributions are grouped. Among these blocks, the lower triangular part of the blocks in the diagonal corresponds to cases where the distributions are mismatched but out-of-sample performance is better. We also note that the blocks in the upper-right and lower-left corner show the same pattern in the lower triangular part of the blocks.

Perhaps it is already clear to the reader why this direction of our result is not particularly surprising, and in fact it is not all that significant in practice either. In the setup depicted in this part of the simulation, if we are able to choose a test distribution, then we might as well choose a distribution that concentrates on the region that the system learned best. Such regions are likely to correspond to areas where large concentrations of training data are available. This can be expressed in terms of lower-entropy test distributions, which are over-concentrated around the areas of higher density of training points. Such concentration results in a better *average* out-of-sample performance than that of $P_S = P_R$.

Figure 2.2 illustrates the entropy of different distributions. We plot $H(X_R)$ versus $H(X_S)$, where $H(\cdot)$ is the entropy of the discretized probability distributions and $X_R \sim P_S$ and $X_S \sim P_S$, marking the cases where using $P_S \neq P_R$ resulted in better out-of-sample performance of the algorithm. As it is clear from the plot, these cases occur when $H(X_S) < H(X_R)$.

A simple way to think of the problem is to see that if we could freely choose a test distribution, and our learning algorithm outputs θ^* as the learned parameters that minimizes some loss function $l(x, y, \theta)$ on a training data set $R = \{(x_i, y_i)\}$, then to minimize the out-of-sample error we would choose $P_S(x) = \delta(x - x^\star)$, where δ is the delta-dirac function and $x^\star = \arg\min_R(l(x, y, \theta^*)))$, the point in the input space where the minimum out-of-sample error occurs.

Similar results as those shown in Figure 2.1 are found when $N_R = 300$.

2.1.2 Fixing the test distribution

Figure 2.3 shows the result of the simulation in the other direction. Each entry in the matrix again corresponds to a pair of distributions P_R and P_S. However, this time we fix P_S and evaluate the percentage of runs where using $P_R \neq P_S$ yields better out-of-sample performance than if $P_R = P_S$. More precisely, once again each entry computes the quantity in Equation 2.5.

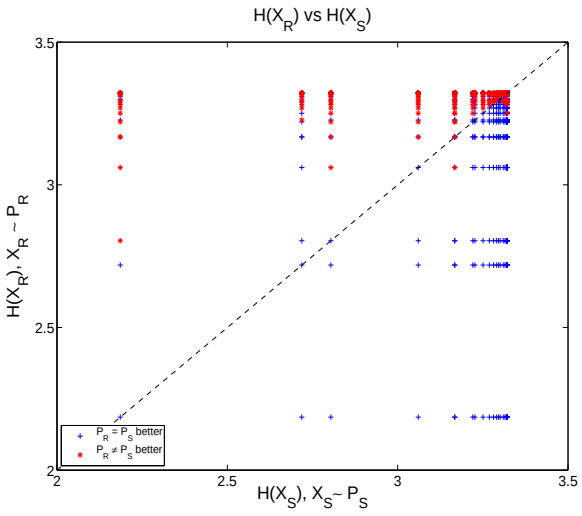


Figure 2.2: $H(X_R)$ vs $H(X_S)$: Characterization of why out-of-sample performance is better if there is a mismatch in distributions when P_R is fixed, using entropy.

This is the case that occurs in practice, where the distribution the system will be tested on is fixed by the problem statement. However, the training set might have been generated with a different distribution, and we would like to determine if training with a data set coming from P_S would have resulted in better out-of-sample performance. If the answer is yes, then one can consider the matching algorithms that we mentioned to transform the training set into what would have been generated using the alternate distribution.

The simulation result is quite surprising, as once again *there is a significant number of entries where more than 50% of the runs have better performance when mismatched distributions are used.* For 14% of the entries, a mismatch between P_R and P_S results in lower out-of-sample error, as indicated by the light green, yellow, orange, and red entries in the matrix.

In this case, although the block structure is still present, there is no longer a clear pattern relating the entropies of the training and test distributions that allows explaining the result easily as in the previous simulation. Notice that there are cases where the mismatch is better if we choose P_R of both lower and higher entropy than the given P_S. This is clear in the plot since the indicated regions in the block structure are no longer lower-triangular but occupy both sides of the diagonal. We look at this result further in the following section, when we analyze the other learning setting: regression.

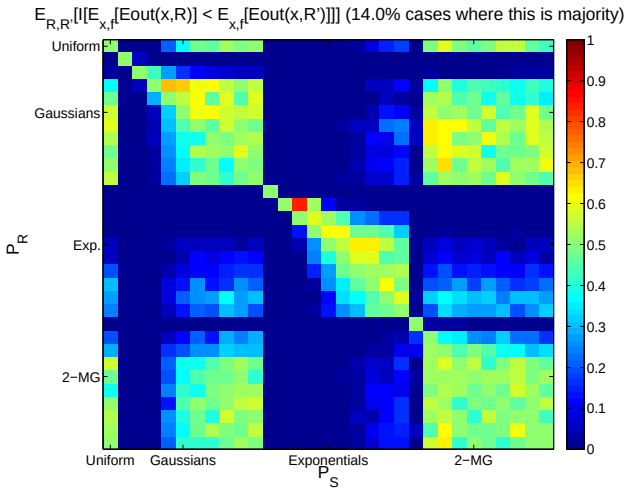


Figure 2.3: Summary of Monte Carlo Simulation. Plot indicates, for each combination of probability distributions, $\mathbb{E}_{R\sim P_R, R'\sim P_S}[I[\mathbb{E}_{f,x\sim P_S}[E_{\text{out}}(x,R,f)] < \mathbb{E}_{f,x\sim P_S}[E_{\text{out}}(x,R',f)]]]$.

2.2 Empirical and analytic results in the regression setting

We have shown empirical evidence that a mismatch in distributions can lead to better out-of-sample performance in the classification setting, and now we focus on the regression setting to cover the other major class of learning problems. In this section, we use the expressions for the expected out-of-sample error as a function of x, a general test point in the input space $\mathcal{X}$, and R, the training set, averaging over target functions and noise realizations. These expressions are derived in detail in Appendix A, for the case where we use a squared loss function and a linear model with non-linear transformations for the hypothesis set. This correspond to the choice of linear model and loss function of the simulations shown in the previous section.

The difference now is that although we choose again $\mathcal{X} = [-1, 1]$, in the regression setting $\mathcal{Y} = \mathbb{R}$. To analyze the most general regression case, we also introduce both "stochastic" and "deterministic" noise [2]. We take $y_i = f(x_i) + \epsilon_i$, where ϵ_i represents the stochastic noise, and where f is more complex than the elements of $\mathcal{H}$, so $f \notin \mathcal{H}$, hence the deterministic noise. We make the usual assumption about the stochastic noise, which is that it has zero mean and is iid. That is, $\mathbb{E}[\epsilon] = 0$, and $\mathbb{E}[\epsilon\epsilon^T] = \sigma_N^2 I$, where I is the identity matrix and σ_N is the standard deviation of the noise. We also make the assumption that the coefficients of the target function that are not included in the model, θ_C, have covariance matrix $\mathbb{E}[\theta_C \theta_C^T] = \sigma_C^2 I$.

As introduced in Appendix A, for simplicity we let

$$z = \phi(x). \tag{2.6}$$

We reorganize the features in z and elements of θ as

$$z^T = [z_M^T \; z_C^T], \quad \theta^T = [\theta_M^T \; \theta_C^T] \tag{2.7}$$

so that the first M features of z correspond to the features in the linear transformation that $\mathcal{H}$ can express. The matrix Z is the "transformed data matrix", with

$$Z = [Z_M \quad Z_C]^T. \tag{2.8}$$

These matrices are precisely defined in Appendix A.

Taking the expected value with respect to the noise, the out-of-sample error at a point $x \in \mathcal{X}$ is given by

$$\mathbb{E}_{f,\epsilon}[E_{\text{out}}(x, R, f, \epsilon)] = \sigma_C^2 \|z_C^T - z_M^T Z_M^\dagger Z_C\|^2 + \sigma_N^2 z_M^T (Z_M^T Z_M)^{-1} z_M + \sigma_N^2 \tag{2.9}$$

Notice that the above expression is independent of θ (i.e., the target function), as well as of the noise. The only remaining randomness in the expression comes from generating R, and from z, the point chosen to test the error, making the analysis very general.

Now, we are interested in minimizing the expected out-of-sample error. Let R denote a training data set generated according to P_R, while R' a data set generated according to P_S. Can we find $P_R \neq P_S$ such that

$$\mathbb{E}_{R,x,\theta_C,\epsilon}[E_{\text{out}}(x, R, f, \epsilon)] < \mathbb{E}_{R',x,\theta_C}[E_{\text{out}}(x, R', f, \epsilon)]? \tag{2.10}$$

The simulation shown in Section 2.1.2, although in a classification setting, suggests that this is the case. We run the same Monte Carlo simulation in this regression setting. The advantage is that the closed-form expression in Equation 2.9 already averages over target functions and noise, allowing us to run in a shorter time more combinations of P_R and P_S. This expression only requires running Monte Carlo simulations for the matrix Z and hence the two terms involving it, $Z_M^\dagger Z_C$ an $(Z_M^T Z_M)^{-1}$. The expectation over $x \sim P_S$ can be done using numerical integration, which is faster than the Monte Carlo simulation in this one-dimensional setting. In this case, we consider the same families of distributions, but we vary the standard deviation of the distribution in smaller steps to obtain a finer grid.

Figure 2.4 indicates that the question posed in Equation 2.10 has an affirmative answer in 21% of the $P_R \neq P_S$ combinations that we considered. This particular simulation used the Fourier harmonics

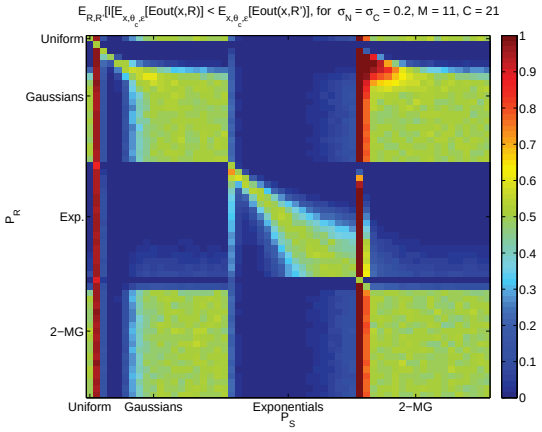


Figure 2.4: Monte Carlo simulation for $\mathbb{E}_{R\sim P_R, R'\sim P_S}\left[I[(\mathbb{E}_{x,\theta_C,\epsilon}[E_{\text{out}}(x,R,\theta,\epsilon)] < \mathbb{E}_{x,\theta_C,\epsilon}[E_{\text{out}}(x,R',\theta,\epsilon)]]\right]$, $M = 11$, $C = 21$, $N = 500$, and $\sigma_N = \sigma_C = 0.2$.

for the non-linear transformation up to order 5, so that $M = 11$. That is,

$$\phi_M(x) = [1 \quad \cos(\pi x) \quad \sin(\pi x) \quad \cdots \quad \cos(5\pi x) \quad \sin(5\pi x)]^T. \tag{2.11}$$

On the other hand, the target functions were generated using harmonics up to order 10, so that $C = 21$, with random Fourier coefficients. Both $\sigma_C = \sigma_N = 0.2$, and $N = 500$. Each entry in the matrix computes

$$\mathbb{E}_{R\sim P_R, R'\sim P_S}\left[I[(\mathbb{E}_{x,\theta,\epsilon}[E_{\text{out}}(x,R,\theta,\epsilon)] < \mathbb{E}_{x,\theta,\epsilon}[E_{\text{out}}(x,R',\theta,\epsilon)]]\right], \tag{2.12}$$

which is the same quantity as that of Equation 2.5, except that now f is determined by θ.

Notice that, as shown in Figure 2.3, the cases where mismatched distributions outperform matched ones cannot be explained using an entropy argument, as was the case in Section 2.1.1. Notice also that there are now combinations for P_R and P_S where almost 100% of the simulations returned lower out-of-sample error for mismatched distributions. In particular, this happened when P_S was a truncated Gaussian with small standard deviation ($\sigma = 0.2$), and when P_S was a mixture of two Gaussians with $\sigma = 0.2$. In addition, we note the similarity between this simulation and the one shown for the classification setting in Figure 2.3.

We varied the size of N in order to see the effect of the sample size. We see very little variation in the results. Holding the other parameters constant, we obtain a very similar result. For $N = 1000$ and for $N = 3000$, we obtain an affirmative answer to the question posed in Equation 2.10 in 21% and 20% of the cases where $P_R \neq P_S$ respectively, so the result does not change from what we obtained in

the $N = 500$ case. For $N = 100$, the percentage is even higher, at 30%. Hence, there is clear evidence that although the number of combinations of distributions for which a mismatch between training and test distributions is larger for smaller N, the result still holds as N grows. Notice that in the simulations, the target function has 21 parameters. Hence, roughly for $N = 100$ there are effectively 5 samples per parameter, while for $N = 3000$ there are 150 samples per parameter. This covers a wide range, from small to large sample sizes, given the complexity of the target function.

Going back to the derived expressions, a closed-form solution for the expected out-of-sample error is given by

$$\mathbb{E}[E_{\text{out}}(x, R, \theta, \epsilon)] = \mathbb{E}_R \int_{-\infty}^{\infty} \sigma_C^2 \|z_C^T - z_M^T Z_M^\dagger Z_C\|^2 P_S(x)dx + \int_{-\infty}^{\infty} \sigma_N^2 z_M^T (Z_M^T Z_M)^{-1} z_M P_S(x)dx + \sigma_N^2. \tag{2.13}$$

It cannot be further reduced analytically due to the inverse matrix terms. Yet, if we assume $C = M$ so that only stochastic noise is present, the expression reduces to

$$\begin{aligned} \mathbb{E}_{\epsilon,R,x,\theta}[E_{\text{out}}(x, R, \theta, \epsilon)] &= \sigma_N^2 + \mathbb{E}_R \int_{-\infty}^{\infty} \sigma_N^2 z^T (Z^T Z)^{-1} z P_S(x)dx \\ &\geq \sigma_N^2 \left(1 + \int_{-\infty}^{\infty} z^T (\mathbb{E}_R[Z^T Z])^{-1} z P_S(x)dx\right), \end{aligned} \tag{2.14}$$

where we use the result in [37] for the expected value of the inverse of a matrix. With this expression, we can find a specific example of a mismatched training distribution that leads to better out-of-sample results. Again, without loss of generality, we pick the linear transformation consisting of Fourier harmonics, namely

$$z = [1 \quad \cos(\pi x) \quad \sin(\pi x) \quad \cdots \quad \cos(m\pi x) \quad \sin(m\pi x)]^T \tag{2.15}$$

as this allows a vast representation of target functions. Here, $M = 2m + 1$. A few examples of the variety of the target functions that can be achieved with this model are shown in Figure 2.5.

If P_R is a Uniform distribution over $\mathcal{X}$, or a Gaussian distribution truncated to this interval, then

$$\begin{aligned} \mathbb{E}_R[Z^T Z] &= \mathbb{E}_R \sum_{i=1}^{N} z_i z_i^T \\ &= N\text{diag}(1, 0.5, 0.5, \ldots, 0.5) \end{aligned} \tag{2.16}$$

The above result is trivial for the uniform distribution case, and can be easily evaluated with numerical

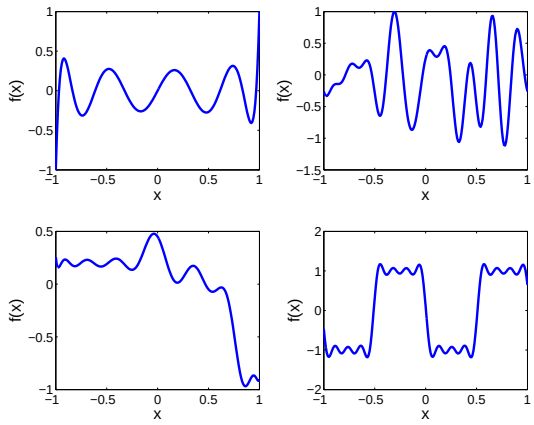


Figure 2.5: Sample realizations of targets generated with a truncated Fourier Series of 10 harmonics.

integration for the truncated Gaussians. This implies that

$$\begin{aligned}\mathbb{E}_{\epsilon,R,x,\theta}[E_{\text{out}}(x,R,\theta,\epsilon)] &\geq \sigma_N^2\left(1+\mathbb{E}_x\left[\frac{2m+1}{N}\right]\right)\\ &= \sigma_N^2\left(1+\frac{M}{N}\right)\end{aligned} \tag{2.17}$$

Now instead, pick R to be distributed according to Uniform$[-a,a]$. In this case,

$$\mathbb{E}_R[Z^TZ]_{ij} = \begin{cases} \text{sinc}(ja) & \text{if } i=1, j \text{ is even}\\ \text{sinc}(ia) & \text{if } j=1, i \text{ is even}\\ 1/2\left(1+(-1)^i\,\text{sinc}(ia)\right) & \text{if } i=j\neq 1\\ 1/2\left(\text{sinc}((i+j)a)+\right. & \text{if } i\neq j, \text{and}\\ \quad\left.\text{sinc}((i-j)a)\right. & i \text{ and } j \text{ odd}\\ 1/2(\text{sinc}\left((i+j)a\right)- & \text{if } i\neq j, \text{and}\\ \quad\text{sinc}((i-j)a) & i \text{ and } j \text{ even}\\ 0 & \text{else}\end{cases} \tag{2.18}$$

Figure 2.6 shows the closed-form bound for various choices of a and $M=10$, choosing P_S to be a truncated Gaussian with $\sigma=0.4$. The dotted line shows the bound for the case $P_R=P_S$. As it is clear from the plot, there are various choices for a so that equation 2.10 is satisfied in terms of the bound.

Since this is only a lower bound on the error, we verify that the minimum suggested by the bound does correspond to a superior mismatched distribution. We Monte-Carlo the value for both

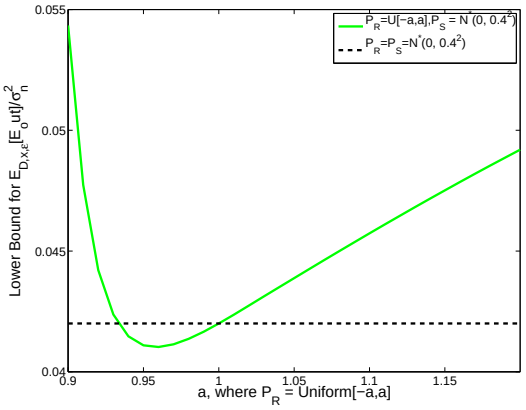


Figure 2.6: Bound for $\mathbb{E}_{R,x,\epsilon}[E_{\text{out}}(x,R)] - \sigma_N^2$ when R is generated with $P_R = P_S = \mathcal{N}^*(0, 0.4^2)$ and for $P_R \neq P_S$ with $P_R = \text{Uniform}[-a, a]$.

cases considered: we choose $P_S = \mathcal{N}^*(0, 0.4^2)$ and generate R' according to P_S, while R is generated according to $U[-0.97, 0.97]$. Notice that we use $a = 0.97$ as this choice results in the lowest error bound from Figure 2.6. Using $m = 10$, $N = 500$ and averaging over 10^8 realizations of R and R' we obtain

$$\mathbb{E}_{R,x,\theta,\epsilon}[E_{\text{out}}(x,R,\theta,\epsilon)] = 1.0429\sigma_N^2 \quad < \quad \mathbb{E}_{R',x,\theta,\epsilon}[E_{\text{out}}(x,R',\theta,\epsilon)] = 1.0440\sigma_N^2 \tag{2.19}$$

Hence, we have a concrete example of a distribution P_R that is different from P_S (Figure 2.7) that leads to better out-of-sample performance, averaging over noise realizations and target functions. The existence of such distributions leads to the concept of a dual distribution which we examine in the next chapter.

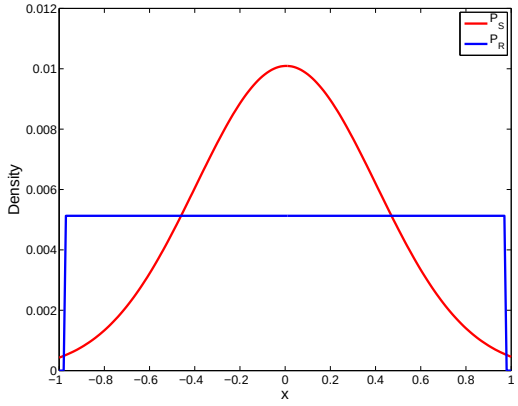


Figure 2.7: Pair of distributions $P_R \neq P_S$ such that expected out-of-sample error is lower when R is generated according to P_R rather than according to P_S for a regression problem in the domain $\mathcal{X} = [-1, 1]$.

Chapter 3

The dual distribution

As shown in the previous chapter, contrary to the common presumption, the optimal distribution from which to sample training data is not necessarily the test distribution P_S. Instead, we call the optimal training distribution the *dual distribution*. This distribution only depends on the test distribution and not in the particular target function in question. In this chapter, we define the dual distribution precisely and then show how to obtain it in the general case, as well as in a practical scenario. We end the chapter with a comparison of the dual distribution approach and a related concept in active learning.

Given a distribution P_S, we define a *dual distribution* $P_R^\star$ to be a distribution that achieves

$$\min_{P_R} \mathbb{E}_{R,x,f,\epsilon}[E_{\text{out}}(x, R, f, \epsilon)] \tag{3.1}$$

where R is a data set generated according to P_R and $x \sim P_S$. The above minimization problem of course has the constraint that P_R must be non-negative and should be normalized, so that the solution yields a valid probability distribution.

3.1 Discrete input spaces

We first find the dual distribution in the case where the input space $\mathcal{X}$ is a discrete set. Let $\mathcal{X} = \{x_j\}_{j=1}^d$, so that P_R and P_S become probability mass functions on d points. Hence, in this setting, finding the dual distribution becomes an optimization problem in $d-1$ dimensions. We only optimize with respect to $d-1$ elements of P_R, since the last element can be determined from the normalization constraint.

For simplicity, we illustrate the solution for a regression problem where only stochastic noise is present. Given R, from Equation 2.9 we can compute the expected out-of-sample error with respect

to P_S, the noise, and the target function as

$$\mathbb{E}_{x,\epsilon,\theta}[E_{\text{out}}(x,R,\epsilon,\theta)] = \sigma_N^2 \sum_{i=1}^{d} z_i^T (Z^T Z)^{-1} z_i P_S(x_i). \tag{3.2}$$

In this case, there are $\sum_{i=1}^{N} \binom{d}{i}$ possible data sets of size N (allowing for repetition of points in the data set) that could be obtained for any given P_R. To simplify the notation, since $\mathcal{X}$ is finite, we assign each of the points a number, from 1 to d, and we denote the out-of-sample error for each of these data sets as $E_{i_1,i_2,\cdots,i_N}$, where i_k indicates the element number in $\mathcal{X}$ that corresponds to the k'th data point in R.

Hence, we can find the expected out-of-sample error with respect to P_R as

$$\mathbb{E}_{R,x,\epsilon,\theta}[E_{\text{out}}(x,R,\epsilon,\theta)] = \sum_{i_1,i_2,\ldots,i_N} p_{i_1} p_{i_2} \cdots p_{i_N} E_{i_1,i_2,\ldots,i_N}, \tag{3.3}$$

where all the $E_{i_1,\ldots,i_N}$ can be found using Equation 3.2. Therefore, $P_R^\star$ is the solution to the following optimization problem:

$$\begin{aligned} \min_{p_1,p_2,\ldots,p_d} \quad & \sum_{i_1,i_2,\ldots,i_N} p_{i_1} p_{i_2} \cdots p_{i_N} E_{i_1,i_2,\ldots,i_N} \\ \text{subject to} \quad & \sum_{i=1}^{d} p_i = 1 \\ & p_i \geq 0 \end{aligned} \tag{3.4}$$

Let us look at a concrete example, with $N = 3$,

$$\begin{aligned} z = \Phi(x) &= [\cos(\pi x) \ \ \sin(\pi x)]^T \\ \mathcal{X} &= \{-3/4, -1/4, 0, 1/4, 3/4\} \\ P_S &= [1/3, 0, 1/3, 1/3, 0] \\ [x_1, x_2, x_3, x_4, x_5] &= [-3/4, -1/4, 0, 1/4, 3/4] \end{aligned} \tag{3.5}$$

Solving the optimization problem given in Equation 3.4 yields $P_R^\star \neq P_S$, with

$$P_R^\star = [0.4672, 0.1140, 0.1140, 0.000, 0.3048]. \tag{3.6}$$

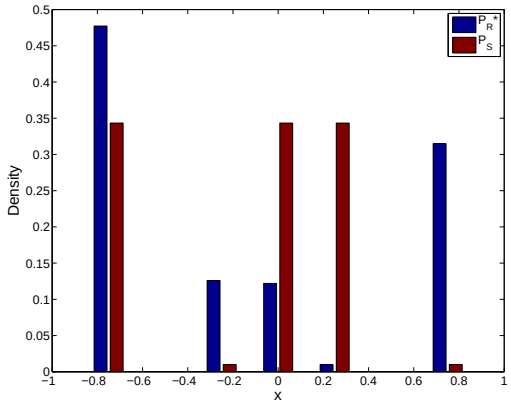


Figure 3.1: Probability mass functions for a given P_S and its dual $P_R^\star$, in a regression problem with stochastic noise, discrete input space $\mathcal{X} = \{-3/4, -1/4, 0, 1/4, 3/4\}$, and $N = 3$.

For this example,

$$\mathbb{E}_{R,x,\epsilon,\theta}[E_{\text{out}}(x, R, \theta, \epsilon)] = 1.1391\sigma_n^2 < \mathbb{E}_{R',x,\epsilon,\theta}[E_{\text{out}}(x, R, \theta, \epsilon)] = 1.5778\sigma_n^2, \tag{3.7}$$

where R' is generated according to P_S and R according to $P_R^\star$. Clearly there is a gain by training with the dual distribution, in this case. When running the optimization, for data sets that have repeated points that result in undefined out-of-sample error as the matrix $(Z^T Z)^{-1}$ is singular, we conservatively take their error to be the maximum finite out-of-sample error over all combinations of possible data sets. Figure 3.1 shows the dual distribution found, along with the given P_S.

Notice that if a different loss function is chosen and no closed form solution exists for $E_{\text{out}}(x, R)$, the dual distribution can still be found using the same procedure as above. The only difference is that $E_{\text{out}}(x, R)$ must be estimated, using a held-out set for instance, for each possible dataset R, so that the corresponding $E_{i_1,\ldots,i_N}$ can be computed and given as inputs to the optimization problem of Equation 3.4

A very important property of the optimization problem formulated in Equation 3.4 is that it is a convex optimization program. In fact it is a Geometric Program, although different from a standard Geometric Program, since the equality constraint is not a monomial. Yet, the problem is still convex. To illustrate this, let

$$\psi_i = \log(p_i) \tag{3.8}$$

$$\Lambda_{i_1,\ldots,i_N} = \log(E_{i_1,\ldots,i_N}). \tag{3.9}$$

This change of variables implicitly makes $p_i > 0$ so that the inequality constraints can be removed. Also, the problem can be rewritten as

$$\min_{\psi_1,\psi_2,\ldots,\psi_d} \quad \sum_{i_1,i_2,\ldots,i_N} e^{\sum_{k=1}^{N}\psi_{i_k}+\Lambda_{i_1,i_2,\ldots,i_N}} \tag{3.10}$$

$$\text{subject to} \quad \sum_{i=1}^{d} e_i^{\psi} = 1 \tag{3.11}$$

Notice that the objective function is a sum of exponential functions of affine functions of ψ_i. Since exponential functions are convex, affine transformations of convex functions are also convex, and sums of convex functions result in a convex function, the objective in Equation 3.10 is convex [19]. Following the same argument, the equality constraint is also convex, so that the optimization problem is a convex program.

Hence, if a minimum is found, this is the global optimum with a corresponding dual distribution. This problem can be solved with any convex optimization package. Furthermore, in most applications, P_S is generally unknown and is estimated by binning the data, which leads to a discrete version of P_S. Therefore, this discrete formulation is appropriate to find dual distributions in such settings. Solving the Geometric Program described by Equation 3.10 thus allows us to find the dual distribution in various practical settings.

Nevertheless, we need to address the more general case of continuous input spaces. The following section describes how to find the dual distribution in that case, as well as how to implement it in a practical scenario.

3.2 The continuous case

When the input space $\mathcal{X}$ is continuous, as it is the case in most applications, the optimization problem in Equation 3.1 is a functional optimization problem, since we are interested in finding the full distribution P_R. We denote the corresponding probability density function by p_R, and optimize with respect to this density. The objective function of the optimization problem can be written as the functional $J : \mathcal{P} \to \mathbb{R}$

$$J(p) = \int_{x_N} \cdots \int_{x_1} L(x_1, \ldots, x_N) \prod_{i=1}^{N} p(x_i) dx_1 \cdots dx_N, \tag{3.12}$$

where

$$L(x_1, \ldots, x_N) = \mathbb{E}_{x\sim P_S, f, \epsilon}[E_{\text{out}}(x, R, f, \epsilon)], \tag{3.13}$$

and $\mathcal{P}$ is the set of all probability density functions with $\mathcal{P} \subset L^1$. (Recall an L^p space over X is defined as the space of functions f for which $\int_X |f(x)|^p < \infty$. Since probability density functions integrate to unity, they are elements of L^1). In the following subsection, we use functional calculus to arrive at the analytic condition that the dual distribution must satisfy.

3.2.1 Analytic condition for the dual distribution

To minimize the functional $J(p)$, we first transform the variables, as we did in Section 3.1. Let

$$\psi(x) = \log p(x) \tag{3.14}$$

$$\Lambda(x_1, \ldots, x_N) = \log(L(x_1, \ldots, x_N)). \tag{3.15}$$

The optimization problem becomes

$$\begin{aligned} \min_{\psi} &= J(\psi) \\ \text{subject to} \quad & \int e^{\psi(x)} dx = 1 \end{aligned} \tag{3.16}$$

where

$$J(\psi) = \int_{x_N} \cdots \int_{x_1} e^{\Lambda(x_1,\ldots,x_N)) + \sum_{i=1}^{N} \psi(x_i)} dx_1 \cdots dx_N, \tag{3.17}$$

and where the positivity constraints are implicit, given the domain of the logarithm.

Now, recall that the gradient of a functional $J(\psi)$, denoted as $\nabla_\psi J$, is given by [28]

$$J(\psi + \delta\zeta) = J(\psi) + \delta \langle \nabla_\psi J, \zeta \rangle + \mathcal{O}(\delta^2), \tag{3.18}$$

where $\delta \in \mathbb{R}, \delta > 0$, and $\zeta \in \mathcal{P}$ is an arbitrary function. Consider the Lagrangian

$$\mathcal{L}(\psi) = J(\psi) + \lambda \left(\int e^{\psi(x)} dx - 1 \right). \tag{3.19}$$

Then, the dual distribution must satisfy

$$\nabla_\psi (\mathcal{L}(\psi(x))) = 0. \tag{3.20}$$

In fact, we can use the Euler-Lagrange theorem [30] to show that if there is a function ψ that satisfies Equation 3.20, then it is the global minimizer. The theorem states that for a function $f \in \mathcal{C}^2$, with $f : [a, b]^d \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ where $\mathcal{C}^2$ denotes continuously twice differentiable functions, and we have

the functional

$$\inf_{\mathbb{X}} I(u) = \inf_{\mathbb{X}} \int_{\mathcal{X}} f(x, u, u')dx \tag{3.21}$$

where $\mathbb{X} = \{u \in \mathcal{C}^1, u : [a, b]^d \to \mathbb{R}, |u|_{\partial_1\Omega} = u_0\}$, u_0 are the boundary conditions, and $\mathcal{X} = [a, b]^d$, then if $I(u)$ admits a minimizer $\bar{u} \in \mathcal{C}^2$, then $\bar{u}$ satisfies the Euler-Lagrange (EL) equation:

$$\sum \frac{\partial}{\partial x_i} \frac{\partial}{\partial u'} f(x, \bar{u}(x), \bar{u}'(x)) - \frac{\partial}{\partial u} f(x, \bar{u}(x), \bar{u}'(x)) = 0. \tag{3.22}$$

Conversely, if $\bar{u}$ satisfies the EL equation and the mapping $M(u, u') \to f(x, u, u')$ is convex for every $x \in [a, b]^d$, then $\bar{u}$ minimizes $I(u)$.

In our case, $u = \psi$, and $J(\psi) = I(\psi)$. Also, in our case, u' does not appear, so that $f(x, \psi, \psi') = f(x, \psi)$. Hence, having the gradient of the Lagrangian with respect to ψ equal to 0 is equivalent to satisfying the EL equation. This is the necessary condition.

Now, we can show that it is in fact a sufficient condition by using the converse. Notice that in our case $\mathbb{X} = \mathcal{P}$ is a convex set, as convex combinations of density functions are also convex. Hence, all that remains to show is that the mapping M is convex, that is, show that for $0 \leq \alpha \leq 1$, $\alpha \in \mathbb{R}$,

$$M(\alpha\psi + (1 - \alpha)\phi) < \alpha M(\psi) + (1 - \alpha)M(\phi). \tag{3.23}$$

Substituting, we have in the left hand side,

$$e^{\Lambda(x_1,\ldots,x_N)+\alpha\sum_i \psi(x_i)+(1-\alpha)\sum_i \phi(x_i)}. \tag{3.24}$$

On the right hand side we have

$$\alpha e^{\Lambda(x_1,\ldots,x_N)+\sum_i \psi(x_i)} + (1 - \alpha)e^{\Lambda(x_1,\ldots,x_N)+\sum_i \phi(x_i)}. \tag{3.25}$$

Now, we notice that due to the strict convexity of the exponential function

$$e^{\alpha\theta_1+(1-\alpha)\theta_2} < \alpha e^{\theta_1} + (1 - \alpha)e^{\theta_2}. \tag{3.26}$$

Hence, dividing both sides of Equation 3.23 by $e^{\Lambda(x_1,\ldots,x_N)}$ and substituting $\theta_1 = \sum_i \psi(x_i)$ and $\theta_2 = \sum_i \phi(x_i)$ shows that the mapping M is strictly convex.

This implies that if the dual distribution exists, that is, if we find ψ that satisfies Equation 3.20, then it is the unique, and is the global minimizer of J, and by constraint satisfaction, also the minimizer of $\mathcal{L}$. Note the theorem assumes continuous differentiability of u, but the theorem can be generalized for functions that are continuously differentiable, except at sets of measure zero.

We now compute the gradient of the Lagrangian. For simplicity, let dR denote $dx_1 \cdots dx_N$, and let $\mathcal{R}$ denote the support of the set $\{x_1, \ldots, x_N\}$ then

$$\begin{aligned} J(\psi + \delta\xi) &= \int_{\mathcal{R}} e^{\sum_{i=1}^N \psi(x_i) + \delta\xi(x_i) + \Lambda(x_1,\ldots,x_N)} dR \\ &= \int_{\mathcal{R}} e^{\sum_{i=1}^N \psi(x_i) + \Lambda(x_1,\ldots,x_N)} \left(1 + \delta \sum_{i=1}^N \xi(x_i) + \mathcal{O}(\delta^2)\right) dR \\ &= J(\psi) + \delta \int_{\mathcal{R}} e^{\sum_{i=1}^N \psi(x_i) + \Lambda(x_1,\ldots,x_N)} \sum_{i=1}^N \xi(x_i) dR \\ &= J(\psi) + \sum_{i=1}^N \left\langle \int_{x_n, n\neq i} e^{\sum_{i=1}^N \psi(x_i) + \Lambda(x_1,\ldots,x_N)} dx_{n_{n\neq i}}, \xi(x_i) \right\rangle, \end{aligned} \tag{3.27}$$

where the simplification follows from using a Taylor expansion of the exponential. Finally, since the loss functions we are interested in are independent of the order of the points in the training set, then the logarithm of the loss function $\Lambda(x_1, \ldots, x_N)$ is symmetric with respect to x_i. Therefore,

$$\nabla_\psi (J(\psi(x_n))) = N \mathbb{E}_{\substack{x_i \sim e^\psi \\ i\neq n}} [L(x_1, \ldots, x_N)]. \tag{3.28}$$

Following a similar procedure for the second term in the Lagrangian, we obtain that at point x_n

$$\nabla_\psi (\mathcal{L}(\psi(x_n)) = \left(N \mathbb{E}_{\substack{x_i \sim e^\psi \\ i\neq n}} [L(x_1, \ldots, x_N)] + \lambda \right) e^{\psi(x_n)} \tag{3.29}$$

We can now use the constraint to find λ by integrating the above equation over x_n. We obtain

$$\lambda = -N e^{\psi(x_n)} \mathbb{E}_{\substack{x_i \sim p \\ i=1,\ldots,N}} [L(x_1, \ldots, x_N)] \tag{3.30}$$

Substituting for λ we obtain the optimality condition that the dual distribution needs to satisfy:

$$\boxed{p(x_n) \left(\mathbb{E}_{\substack{x_i \sim p \\ i\neq n}} [L(x_1, \ldots, x_N)] - \mathbb{E}_{\substack{x_i \sim p \\ i=1,\ldots,N}} [L(x_1, \ldots, x_N)] \right) = 0.} \tag{3.31}$$

This condition applies to the dual distribution in the general case, without making assumptions about the target class or the learning model. Now, all that remains is to find p that satisfies this condition, which can be done, for example, using functional gradient descent [49].

The functional gradient descent step is given by

$$p(x) := p(x) - \eta \nabla (\mathcal{L}(p(x)) \tag{3.32}$$

where η is the learning rate, hence

$$p(x_n) := p(x_n) - \eta N \left(\mathbb{E}_{\substack{x_i \sim p \\ i \neq n}}[L(x_1, \ldots, x_N)] - \mathbb{E}_{\substack{x_i \sim p \\ i=1,\ldots,N}}[L(x_1, \ldots, x_N)] \right) p(x_n). \tag{3.33}$$

Notice that the integral of the update over x_n is 0. Hence, this update guarantees that the normalization constraint is satisfied at each step, so that gradient descent works in this case as in an unconstrained problem. Therefore, if the initial condition is a valid probability density function (pdf), all subsequent p's will also be valid pdf's.

The interpretation of this update is very intuitive: If a point x_n is included in the training set, and the resulting out-of-sample error is lower than the expected out-of-sample error with N points, that is $\mathbb{E}_{\substack{x_i \sim p \\ i \neq n}}[L(x_1, \ldots, x_N)] < \mathbb{E}_{\substack{x_i \sim p \\ i=1,\ldots,N}}[L(x_1, \ldots, x_N)]$, then $p(x_n)$ should be increased. If including the point leads to a higher out-of-sample error, then the density at this point should be decreased.

In the following subsection, we introduce a concrete example of how the condition of Equation 3.31 can be used computationally to derive the dual distribution.

3.2.2 Dual distribution examples

As shown in the previous subsection, finding the dual distribution reduces to performing functional gradient descent. However, the update rule depends on being able to compute the expected out-of-sample error $\mathbb{E}_{\substack{x_i \sim p \\ i \neq n}}[L(x_1, \ldots, x_N)]$. Computing the expected value with respect to the training set can be readily done using Monte Carlo (MC) simulation. This can be slow unless a closed form for $L(x_1, \ldots, x_N)$ exists.

If a squared loss function is used for ℓ, and the hypothesis class $\mathcal{H}$ is chosen to be a linear model (which can include non-linear transformations of the inputs), then a closed-form solution for $L(x_1, \ldots, x_N)$ exists. This solution is independent of the specific target function. Hence, in this setting, the dual distribution can readily be found. The closed-form solution, as derived in Appendix A is given by

$$L(x_1, \ldots, x_N) = \sigma_C^2 \|\phi_C(x)^T - \phi_M(x)^T \Phi_{MM}^{-1} \Phi_{MC}\|^2 + \mathbb{E}_{x \sim P_S}\left[\sigma_N^2 \phi_M(x)^T \Phi_{MM}^{-1} \phi_M(x)\right] + \sigma_N^2, \tag{3.34}$$

where $\phi : \mathcal{X} \to \mathcal{Z}^{M+C}$ denotes the transformation of the input, with

$$\phi(x) = [\phi_M(x)^T \quad \phi_C(x)^T]^T, \tag{3.35}$$

so that $\phi_M : \mathcal{X} \to \mathcal{Z}^M$ represents the part of the target function that can be captured by the model, and $\phi_C(x) : \mathcal{X} \to \mathcal{Z}^C$ is the part of the target that cannot be captured by the model. The matrices

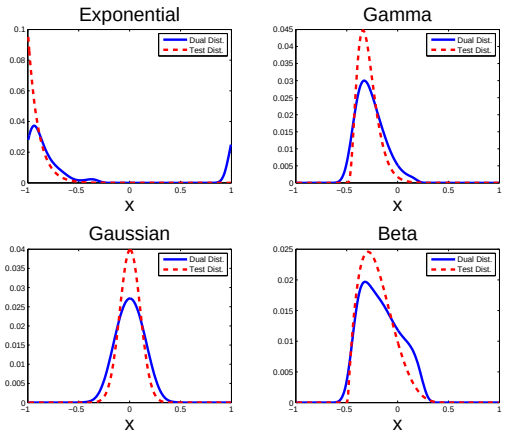


Figure 3.2: Examples of dual distributions, for 1-D test distributions, in a linear regression problem.

Table 3.1: Out-of-sample (OoS) performance improvement when training the learning algorithm with data coming from the dual distribution rather than from the test distribution

TEST DISTR.	PARAMETERS	OOS ERROR IMPROVEMENT
EXPONENTIAL	$\lambda = 5$	46.3%
GAMMA	$\alpha = 4,\ \beta = 0.05$	32.0%
GAUSSIAN	$\mu = 0,\ \sigma = 0.1$	21.4%
BETA	$\alpha = 2,\ \beta = 5$	10.0%
F	$\nu_1 = 100,\ \nu_2 = 80$	5.7%
WEIBULL	$\lambda = 1,\ k = 5$	2.2%
UNIFORM	[-1,1]	0.5%
2-D GAUSSIAN	$\Sigma = [0.1^2\ 0.08; 0.08\ 0.1^2]$	22.6%
2-D MG	$\Sigma = [0.1^2\ 0.06; 0.06\ 0.1^2]$	5.71%

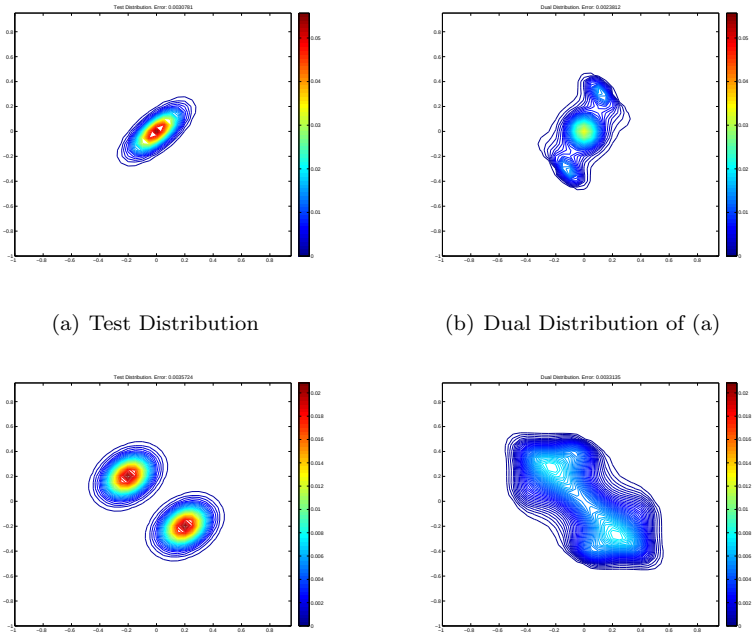
$\Phi_{MM} \in \mathcal{Z}^{M\times M}$ and $\Phi_{MC} \in \mathcal{Z}^{M\times C}$ defined for the training input points $x_1, \ldots, x_N$ are given by

$$\Phi_{MM} = Z_M^T Z_M = \sum_{i=1}^{N} \phi_M(x_i)\phi_M(x_i)^T, \tag{3.36}$$

$$\Phi_{MC} = Z_M^T Z_C = \sum_{i=1}^{N} \phi_M(x_i)\phi_C(x_i)^T. \tag{3.37}$$

Finally σ_N^2 and σ_C^2 characterize the energy of the stochastic noise and 'excess' target complexity as explained before.

Figure 3.2 shows the dual distributions for various one-dimensional test distributions for the regression setup. The learning model uses Fourier harmonics of the input, while the target functions are constructed by considering functions that include Fourier harmonics higher than those that belong to



(a) Test Distribution

(b) Dual Distribution of (a)

(c) Test Distribution

(d) Dual Distribution of (c)

Figure 3.3: Examples of dual distribution, for 2-D test distributions. (a) 2-D Gaussian; (b) Dual distribution for (a); (c) Mixture of 2-D Gaussians; (d) Dual distribution for (c).

the model. The simulation parameters were set to $N = 100$, $M = 3$, $C = 5$, $\sigma_N = \sigma_C = 0.2$. The input domain is $\mathcal{X} = [-1, 1]$, so the distributions were zeroed out outside this domain and renormalized. Table 3.1 shows the parameters of the test distributions and also indicates the improvement in out-of-sample performance when the learning algorithm is trained with samples coming from the dual distribution, rather than from the test distribution. Figure 3.3 shows the dual distribution for two-dimensional test distributions.

As it is clear from Table 3.1, the gains in using the dual distribution can be significant. For these examples, N was chosen so that there were enough samples to estimate the three parameters in the model ($M = 3$), and the target was more complex than the model.

The reader may be wondering how the sample size (N), the excess target complexity with respect to the model ($C - M$) and its magnitude (σ_C), and the stochastic noise level (σ_N) affect the dual distribution. We address this question in the following section.

3.3 Variability of the dual distribution

The definition of the dual distribution is based on some specific aspects of the learning problem, such as the training set size, the target complexity, and the model complexity. In this section, we explore

the change in dual distribution due to these factors.

3.3.1 Asymptotic behavior

The first factor we analyze is the dependence of the dual distribution on N, the training set sample size. In particular, consider the case where $N \to \infty$. Recall that the dual distribution is the distribution P that minimizes the quantity $\mathbb{E}_{x_i \sim P}[L(x_1, \ldots, x_N)]$. Using the closed-form expression for $L(x_1, \ldots, x_N)$ in the squared loss and linear model with non-linear transformations case, we can separate the impact of the stochastic and deterministic noise terms. The stochastic term, that is, the term proportional to σ_N^2 from Equation 3.34, is $\mathcal{O}(1/N)$. Notice that:

$$\mathbb{E}_{x_i \sim P}\left[\Phi_{MM}^{-1}\right] = \frac{1}{N}\mathbb{E}_{x_i}\left[\left(\frac{1}{N}\sum_{i=1}^{N}\phi_M(x_i)\phi_M(x_i)^T\right)^{-1}\right]$$

As $N \to \infty$,

$$\frac{1}{N}\sum_{i=1}^{N}\phi_M(x_i)\phi_M(x_i)^T \xrightarrow{P} \mathbb{E}_{x_i \sim P}[\phi_M(x_i)\phi_M(x_i)^T] \tag{3.38}$$

where $\xrightarrow{P}$ denotes convergence in probability. Substituting, the stochastic noise term simplifies to

$$\frac{1}{N}\mathbb{E}_x\left[\sigma_N^2\phi_M(x)^T\mathbb{E}_{x_i}\left[\phi_M(x_i)\phi_M(x_i)^T\right]^{-1}\phi_M(x)\right]. \tag{3.39}$$

Therefore, this term vanishes as $N \to \infty$.

The remaining term, on the other hand, is $\mathcal{O}(1)$, so this is the term that must be minimized. Following a similar analysis as above, it follows that

$$\lim_{N\to\infty}\mathbb{E}_{x_i \sim P}\left[L(x_1, \ldots, x_N)\right] = \sigma_N^2 + \sigma_C^2\mathbb{E}_x\left[\|\phi_C(x)^T - \phi_M(x)^T\Phi\|^2\right], \tag{3.40}$$

where

$$\Phi = \left(\mathbb{E}_{x_i}\left[\phi_M(x_i)\phi_M(x_i)^T\right]\right)^{-1}\mathbb{E}_{x_i}\left[\phi_M(x_i)\phi_C(x_i)^T\right]. \tag{3.41}$$

Notice that if the collection of features $\{\phi_i(x)\}_{i=1}^{M+C}$ (the components of $\phi(x)$) form an orthonormal set under P, then by definition

$$\mathbb{E}_{x_i \sim P}[\phi_i(x)\phi_j(x)] = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}. \tag{3.42}$$

Therefore, $\mathbb{E}_{x_i}\left[\phi_M(x_i)\phi_M(x_i)^T\right] = I$, and $\mathbb{E}_{x_i}\left[\phi_M(x_i)\phi_C(x_i)^T\right] = \mathbf{0}$, so that $\Phi = \mathbf{0}$. This orthonormality condition holds, for example, when P is a Gaussian distribution and the features are Hermite

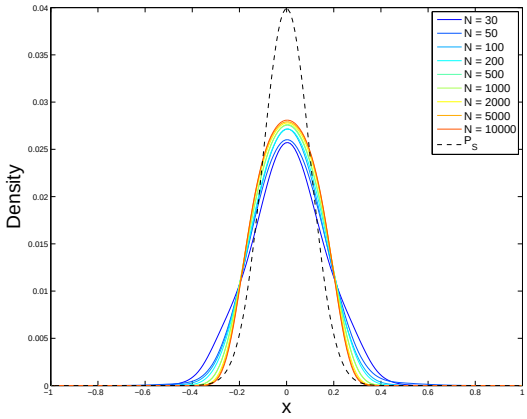


Figure 3.4: Example Dual Distributions in 1-Dimension when the training set size N changes, for $P_S = \mathcal{N}(0, 0.1^2)$, $M = 3$, $C = 5$, using a linear model with Fourier harmonics and a squared loss function.

polynomials, or when P is a uniform distribution and the features are Fourier harmonics, among other cases. In this case, the error would reduce to $\sigma_N^2 + C\sigma_C^2$, and hence there would be no dependence of the error on the training distribution.

However, when the features are not orthonormal under P, the out-of-sample error still changes with the training distribution in the limit as $N \to \infty$. We can minimize Equation 3.40 with respect to Φ. This optimization problem is strictly convex, as it is a quadratic program in the entries of Φ. Finding the gradient and setting it to zero, we find that the necessary and sufficient condition for the minimum is to satisfy the equation

$$\Phi^T \mathbb{E}_{x \sim P_S}[\phi_M(x)] = \mathbb{E}_{x \sim P_S}[\phi_C(x)]. \tag{3.43}$$

The solution to this equation will depend on the type of features chosen. For example, if the features outside the model have mean zero, making the right hand side vanish, then the distribution P that makes the features orthogonal will be the solution.

Figure 3.4 shows the effect of N on the dual distribution in a specific example. For this example $P_S = \mathcal{N}(0, 0.1^2)$, $M = 3$, $C = 5$, $\sigma_N = \sigma_C = 0.2$ and N varies. We used a linear model with Fourier harmonics and a squared loss function. Notice that the variability of the dual distribution is small as N changes.

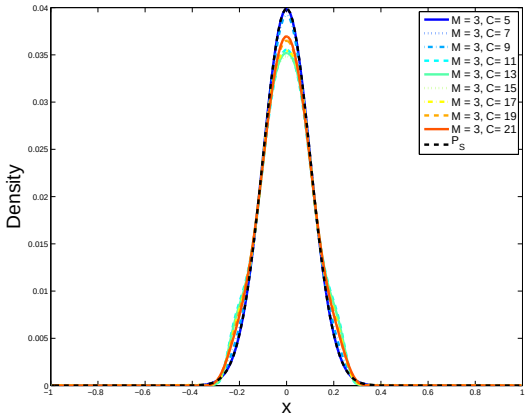


Figure 3.5: Example Dual Distributions in 1-Dimension when the deterministic noise changes, for $P_S = \mathcal{N}(0, 0.1^2)$, $N = 100$, $M = 3$, using a linear model with Fourier harmonics and a squared loss function.

3.3.2 Effect of noise and complexity

We now look at the effect of target complexity. Notice that as the target complexity grows, the deterministic noise term dominates $L(x_1, \ldots, x_N)$. Hence, although the stochastic noise term does not vanish as it is the case when $N \to \infty$, it is still the deterministic noise term that drives the minimization. Figure 3.5 shows the dual distributions for the same test distribution, as the target complexity increases. As the figure shows, there is little variability with respect to the change in target complexity. The variability actually disappears completely if Hermite polynomials are chosen for the features. In this case, for all values of C, $P^\star = P_S$, and Figure 3.6 exemplifies this behavior.

If we now look at the case where only stochastic noise is present in the data, we notice that for finite N, the error becomes

$$\sigma_N^2 \mathbb{E}_{x \sim P_S} \left[\phi_M(x)^T \mathbb{E}_{x_i \sim P} \left[\Phi_{MM}^{-1} \right] \phi_M(x) \right]. \tag{3.44}$$

Again we have a quadratic form, but this time in terms of the matrix Φ_{MM} rather than in term of the matrix Φ. This objective function has a minimum of zero, which is achieved at $\mathbb{E}[\Phi_{MM}] = \mathbf{0}$. However, Φ_{MM} follows the particular form defined in Equation 3.36, which constrains the quadratic program so it yields a different solution.

Figure 3.7 illustrates the effect of increasing the stochastic noise in a concrete example, where the dual distribution is calculated for the same test distribution, and σ_N is increased while holding N, M, and C constant. Notice that for small values of σ_N, $P_R^\star = P_S$.

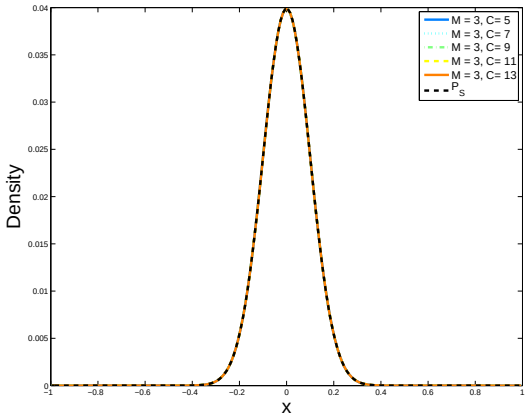


Figure 3.6: Example Dual Distributions in 1-Dimension when the deterministic noise changes, for $P_S = \mathcal{N}(0, 0.1^2)$, $N = 100$, $M = 3$, using a linear model with Hermite polynomial features and a squared loss function.

As it can be seen from the above analysis, the dual distribution is fairly robust with respect to the different components of the learning problem. Namely, the sample size, the noise, and the target complexity. This property allows using the dual distribution in a practical setting where components like the level of noise and target complexity might not be exactly known. The following subsection describes how to find the dual distribution in a practical setting.

3.4 Using the dual distribution in a practical setting

The dual distribution can be applied in two different settings. The first is the population-based active learning setting. This is a special case of active learning, in which contrary to supervised learning where the training data set is fixed, it is possible to sample points according to a desired distribution. This active learning setting is common in applications of experiment design, where the idea is precisely to design the distribution from which points will be sampled. In this case, the design distribution plays the role of the dual distribution, and is chosen by searching within a class of distributions [63].

The second setting where the dual distribution can be used, is in the supervised learning setting case that we have been discussing. In this section, we will show how to use the dual distribution even though the data has already been generated using a fixed distribution. We describe in detail how to do this, and show results on benchmark datasets.

The supervised learning setting poses two challenges for the use of the dual distribution method.

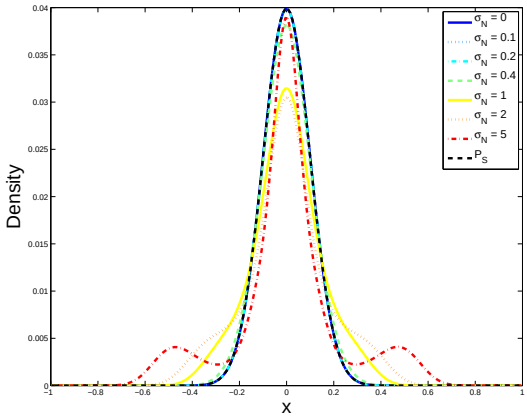


Figure 3.7: Example Dual Distributions in 1-Dimension when the stochastic noise changes, for $P_S = \mathcal{N}(0, 0.1^2)$, $N = 100$, $M = 3$, $C = 5$, using a linear model with Fourier harmonics and a squared loss function.

First, the data set is fixed, so the expected values in Equation 3.33, which are taken with respect to data set generations, cannot be evaluated. Hence, there is a problem in computing the dual distribution itself. The second problem is how to use the dual distribution, since the data is already fixed. It is now necessary to make the data set appear as if it came from a different distribution. Here we describe how to approach both problems.

In order to get the dual distribution with only one data set sample, we make use of the fact that in this setting, the dual distribution only needs to be computed at the positions x_i, $i = 1, \ldots, N$. The reason for this is that we will use matching algorithms to make the sample look as if it was distributed according to the dual distribution. As we explain shortly, matching algorithms only need to compute weights for each of the samples. Hence, it is no longer necessary to compute the full function, but simply its values at N locations. Also, we notice that the gradient at a point x_n is given by the difference in the expected loss with respect to $N - 1$ training points, having x_n present in the training set, and the expected loss with respect to N training points (Equation 3.31). So, given that only one data set is available, $\mathbb{E}_{\substack{x_i \sim p \\ i=1,\ldots,N}}[L(x_1, \ldots, x_N)]$ is approximated by the estimate of the loss using a single sample (i.e. a single data set). On the other hand, $\mathbb{E}_{\substack{x_i \sim p \\ i \neq n}}[L(x_1, \ldots, x_N)]$ is estimated by increasing the weight of point x_n and finding the resulting loss. The difference of the two terms will approximate the effect of this point on the loss, and hence determine an approximate value of the gradient at this point.

Once the dual distribution is computed, we need to make the training dataset look as if it was distributed according to this new distribution. To do this, we make use of the available methods from

the covariate shift literature. These methods are described in Section 1.2. All these methods are variants of importance weighting [64], and their goal is to estimate the weights $w(x) = p_S(x)/p_R(x)$, where p_R and p_S are the training and test densities, respectively. They do this, in order to match P_R to P_S. Some of the methods, like KMM [38], KLIEP [67], and LSIF [42] usually perform better in the covariate shift correction problem, as they try to estimate the ratio directly, rather than computing the numerator and denominator separately. This can be done when there are unlabeled samples available, coming from both the training and test distributions. In our case, the importance weights are given by $w(x) = p_R^\star(x)/p_R(x)$. Here, the numerator is found directly through functional gradient descent, having no available samples distributed according to it. Hence, it is necessary to use methods that actually compute the training density p_R. This can be done either by finding a histogram of the training set with the adequate resolution, or using other non-parametric methods like Kernel Density Estimation (KDE) [56] and [53]. Chapter 5 proposes an alternative method that can be used to match the training distribution to any other distribution, such as the dual. We call this algorithm Soft Matching.

Therefore, the dual distribution can be used in the supervised learning setting, using the mentioned approximation for the functional gradient descent, and making use of importance weighting to change the distribution of the training data. Table 3.2 shows the average out-of-sample error, in both classification and regression tasks on 17 benchmark datasets [7] and [68], when the training set is transformed so that it appears distributed as the dual distribution. The values are compared to the case where no changes are made to the training set.

The results are averaged over 1,000 different splits of the data into training and test sets. The training set is also split further, as a validation set is needed to compute the expected loss for the functional gradient descent. The reported errors are on the test set which is not used at all during training, nor during computation of the dual distribution. For all datasets 25% of the data was left aside for testing, 25% was part of the validation set, and the remaining 50% was used for training. For classification problems, weighted SVMs with Gaussian kernels were used, choosing the kernel width as in [38], with the `libsvm` implementation [23]. Ridge regression was used for the remaining data sets, with regularization parameter $\lambda = 0.1$.

As can be seen from the table, in all of the classification problems, the use of the dual distribution led to a lower out-of-sample error (classification error percentage). Numbers in boldface indicate that the improvement is statistically significant. For the regression problems, the improvements in normalized mean-squared error (NMSE) were smaller but still present. Since the use of weights can lead to an increase in variance or equivalently a sample size reduction [61], it is not surprising that the improvement in performance is lower than in the examples where direct sampling from the dual

Table 3.2: Generalization error in benchmark datasets under the supervised learning paradigm, with and without the use of the dual distribution. 0/1 classification error is reported for the classification tasks; normalized mean-squared error (NMSE) is shown for regression tasks. N is the size of the full data set. All numbers are multiplied by 100.

Dataset	N	Dual	No Dual
(Classif.)		0/1 Error	
Breast C.	278	$\mathbf{25.70 \pm 0.14}$	27.53 ± 0.15
Breast WI	683	$\mathbf{4.38 \pm 0.04}$	4.45 ± 0.04
German Cred.	768	$\mathbf{23.96 \pm 0.09}$	25.08 ± 0.09
Haberman	306	$\mathbf{25.56 \pm 0.13}$	26.10 ± 0.13
Diabetes	768	$\mathbf{24.09 \pm 0.09}$	25.08 ± 0.09
Ionosphere	351	$\mathbf{6.28 \pm 0.07}$	6.41 ± 0.07
(Regression)		NMSE	
Abalone	4177	$\mathbf{50.25 \pm 0.10}$	50.75 ± 0.10
Ailerons	13750	$\mathbf{18.63 \pm 0.02}$	18.65 ± 0.02
Bank8FM	8192	$\mathbf{6.70 \pm 0.01}$	6.72 ± 0.01
Bank32NH	8192	46.84 ± 0.06	46.87 ± 0.06
Bos. Housing	606	36.74 ± 0.27	36.94 ± 0.27
CA Housing	20650	36.15 ± 0.04	36.19 ± 0.04
Cpu-act	8192	$\mathbf{25.97 \pm 0.08}$	26.39 ± 0.08
Cpu-small	8192	$\mathbf{30.11 \pm 0.09}$	30.47 ± 0.09
δ-Ailerons	9129	49.81 ± 0.10	49.81 ± 0.10
Kin8nm	8192	58.83 ± 0.05	58.83 ± 0.05
Puma8nh	8192	61.73 ± 0.05	61.73 ± 0.05

distribution is possible as in Table 3.1. However, the key takeaway is that there is empirical evidence that shows that using the dual distribution does improve out-of-sample performance in a supervised learning setting, in both classification and regression problems.

Since all datasets considered have a multidimensional test distribution, the dual distribution was found for each of the projections of the test distribution, along its original coordinates, and the distribution that led to the lowest error in the validation set was chosen in each run. As we discuss in Section 3.5, finding the dual distribution for a multi-dimensional test distribution is computationally more difficult, as it is necessary to compute numerically a function at every point in a high-dimensional grid. Also, sampling from arbitrary distributions in high-dimensional spaces is less accurate when p is saved in a grid, and this is required at every step of gradient descent.

Some important details regarding the implementation of the algorithm that produces the full dual distribution, as well as the implementation of the algorithm that focuses on the training points, are presented in the following section.

3.5 Computational and implementation details

The previous sections describe how to obtain the dual distribution in two cases: the case where it is possible to compute the expected values in Equation 3.31, so that the full density of the dual

Algorithm 1 Exact dual distribution

Input: P_S, $L(\cdot)$, learning rate η
Discretize domain $\mathcal{X} \to \mathcal{X}_D$
Initialize $p(x_n)$ for $x_n \in \mathcal{X}_D$
repeat
  for all $x_n \in \mathcal{X}_D$ **do**
    $\nabla(\mathcal{L}(p(x_n)) := \left(\mathbb{E}_{\substack{x_i \sim P \\ i \neq n}}[L(x_1, \ldots, x_N)] - \mathbb{E}_{\substack{x_i \sim P \\ i=1,\ldots,x_N}}[L(x_1, \ldots, x_N)] \right) p(x_n)$
    $p(x_n) := p(x_n) - \eta \nabla_\psi(\mathcal{L}(p(x_n))$
    if $p(x_n) < 0$ **then** $p(x_n) = 0$
  end for
  Normalize p
until $(\nabla(\mathcal{L}(p)) = 0)$

distribution can be found, and the practical supervised learning case, where an approximation of the dual distribution is found at the training points. Some important implementation details of both algorithms are described in this section.

Algorithm 1 describes the procedure to obtain the dual distribution in the exact case, that is, when we are allowed to sample data from a desired distribution. This algorithm is the one used to obtain the dual distributions in the examples of Section 3.2.2.

We first discuss a significant speed up that can be applied in this case. In order to obtain the full function numerically, it is necessary to discretize the domain, and obtain the distribution at the desired resolution. Assume the chosen resolution is δ, then the number of times that $\mathbb{E}_{x_i \sim P, i \neq n}[L(x_1, \ldots, x_N)]$ must be computed is proportional to $(1/\delta)^d$, at each step of gradient descent, in d dimensions. This value is computed via MC simulation, and hence it can be a very computationally expensive operation.

When we use a squared loss function and a linear model with non-linear transformations, there is a closed form solution for the loss, given in Equation 3.34. However, it is still necessary to find $\mathbb{E}_{x_i}[\Phi_{MM}^{-1}]$. This matrix must be found through MC simulation for each value of x_n in the grid, while randomizing the remaining $N-1$ points generated according to P. However, there can be a significant saving in computation if we apply the Sherman-Morrison identity [60]:

$$\Phi_{MM}^{-1} = \Phi_{MM,n}^{-1} - \frac{\Phi_{MM,n}^{-1} \phi_M(x_n) \phi_M(x_n)^T \Phi_{MM,n}^{-1}}{1 + \phi_M(x_n)^T \Phi_{MM,n}^{-1} \phi_M(x_n)}, \tag{3.45}$$

where

$$\Phi_{MM,n} = \sum_{\substack{i=1 \\ i \neq n}}^{N} \phi_M(x_i) \phi_M(x_i)^T. \tag{3.46}$$

Hence, $\mathbb{E}_{x_i}[\Phi_{MM,n}^{-1}]$ can be computed once via MC simulation, and the value of $\mathbb{E}_{x_i}[\Phi_{MM}^{-1}]$ can be approximated using the identity, and substituting for $\Phi_{MM,n}^{-1}$ by its expected value. This allows us to

Algorithm 2 Approximate dual distribution for supervised learning

Input: P_S, p_R, $R = \{x_i\}_{i=1}^N$, $L(\cdot)$, learning rate η
Initialize $p(x_i)$ for $x_i \in R$
repeat
  $w := p./p_R$ (element-wise division)
  for all $x_i \in R$ **do**
    $w' := w$
    $\mathbb{E}_{\substack{x_i \sim P \\ i=1,\ldots,x_N}}[L(x_1, \ldots, x_N)] := L(w; x_1, \ldots, x_N)$
    $w'(x_i) := w(x_i) + 1$
    Normalize w' so that $\sum_i w'_i = N$
    $\mathbb{E}_{\substack{x_i \sim P \\ i \neq n}}[L(x_1, \ldots, x_N)] := L(w'; x_1, \ldots, x_N)$
    $\nabla(\mathcal{L}(p(x_n)) := \left(\mathbb{E}_{\substack{x_i \sim P \\ i \neq n}}[L(x_1, \ldots, x_N)] - \mathbb{E}_{\substack{x_i \sim P \\ i=1,\ldots,x_N}}[L(x_1, \ldots, x_N)] \right) p(x_n)$
    $p(x_n) := p(x_n) - \eta \nabla_\psi(\mathcal{L}(p(x_n))$
    if $p(x_n) < 0$ **then** $p(x_n) = 0$
  end for
  Normalize p
until $(\nabla(\mathcal{L}(p)) = 0)$

compute, with a single MC simulation, the value of $\mathbb{E}_{x_i \sim P, i \neq n}[L(x_1, \ldots, x_N)]$ at every desired x_n.

Another computational consideration that should be taken into account regards the constraint satisfaction at each step. Although the update at each step, given by Equation 3.33, guarantees that p integrates to 1 if the initial p is a proper pdf, numerically there might be small errors that can make the resulting density add up to a value slightly different from 1. Hence, in the implementation we normalize p at each step to avoid instability issues.

We also noticed that carrying out the minimization in the p space rather than in the ψ space was much quicker and usually led to solutions that yield the lowest out-of-sample error. The only drawback is that the positivity constraint must be also forced at each step. We did this by using the heuristic of setting to zero at each step any values that become negative.

Finally, for all experiments, p was initialized to be a uniform distribution in the finite domain. Another possible initialization point is $p = p_S$. If an alternative initialization is used, p must be a smooth function, and $p(x) > 0$ at every x. Otherwise, since the updates are proportional to $p(x)$, the points initialized at zero will not change throughout the descent.

Algorithm 2 describes the procedure to obtain an approximate dual distribution, in the supervised learning setting. The same considerations regarding the constraints of the minimization problem are taken into account.

3.6 Differences with active learning

The concept of a dual distribution in supervised learning is somewhat related to similar ideas in active learning and experimental design. Especially, the methods of 'batch' active learning, where a 'design' distribution is found in order to minimize the error, seems to be solving a similar problem to the dual distribution. However, the fundamental difference is that active learning finds such optimal distribution *given a particular target function*. Hence, most methods rely on the information given by the target function in order to find a better training distribution. A common example is when distributions give more weight to points around the boundaries of the target function. Yet, the problem of finding the dual distribution is *independent* of the specific target function. The Monte Carlo simulations presented in Chapter 2, as well as the bounds shown, average over different realizations of target functions.

For example, [43] describes an algorithm to find an appropriate 'design' distribution that will lower the out-of-sample error. In the algorithm proposed, a first parameter is estimated with s data points, and with this parameter the optimal design distribution is found. Having a new design distribution, $T - s$ points are sampled from it and a final parameter is then estimated. Notice, however, that the optimal design distribution is dependent on the target function. In the results we present, if a dual distribution is found given a particular test distribution, such distribution is optimal independently of the specific target function.

Other papers in the active learning community that focus on linear regression, like [63], seem closely related to our work. In the mentioned paper, the results apply to linear regression only, and consider the out-of-sample error conditioned on a given training set. The nice property of the out-of-sample error in linear regression is that it is independent of the target function. This is the reason why even in the active learning setting, the dependence of the target function disappears in this case and the mathematical analysis looks similar to the one we presented in Section 2.2. Yet, even though our analysis in Chapter 2 is done with linear regression and hence uses similar mathematical formulas, our approach is based on averaging over realizations of training sets and of targets functions in the supervised learning scenario, rather than in the cases addressed in the mentioned papers. Furthermore, the problem of finding the dual distribution and the results presented can be applied to other learning algorithms besides linear regression, both for classification and regression problems in the supervised learning setting as shown in the previous section.

Another difference that may stand out to the reader is the way the 'design' distribution is used once it is found in the active learning papers, as opposed to how we propose to use the dual distribution here. In the active learning scenario, points are sampled from the design distribution, but in order to avoid obtaining a biased estimator, as shown in [61], the loss function is weighted for these points with

$w(x) = q(x)/p(x)$, following their notation, where $q(x)$ is the test distribution ($P_S(x)$) and $p(x)$ is the 'design' distribution found. Notice that in the results presented in the simulations of Section 2.1 and in Tables 3.1 and 3.2, we do not re-weight the points but instead explicitly allow a mismatch between P_S and P_R. Furthermore, in the supervised learning setting, where the training set is fixed and we are not allowed to sample new points, we propose that matching algorithms, as the ones described in Section 1.2, be used to match the given training set to the dual distribution. In this case, the objective is to have weights $w(x) = p_R^\star(x)/p_S(x)$, so that the training set appears distributed as the dual distribution. These weights are actually inverse to those used in the active learning algorithms described. Although we are aware that the estimator computed in the linear regression setting will be biased when we use the dual distribution, we are concerned with minimizing the out-of-sample error, which takes into account both bias and variance, and hence we may obtain a biased estimator but improve the mean-squared error performance as the results show in Tables 3.1 and 3.2.

Furthermore, the results shown in [61] hold only in the asymptotic case, and since we are dealing with the supervised learning scenario where only a finite training sample is available, the same assumptions are not valid. Thus, it is no longer optimal to use the mentioned weighting mechanism when N is not sufficiently large, as also shown in [61]. In the active learning setting, it is desirable that as more points are sampled, the proposed algorithms have performance guarantees. Hence, the algorithms are designed to satisfy conditions such as consistency of the estimator, unbiasedness, etc., in the asymptotic case, which explains why the active learning algorithms use the above-mentioned weighting mechanism. In our setting, minimizing the out-of-sample performance with a fixed-size training set is our main objective, which is why the two approaches differ. As it is clear, the dual distribution serves a different purpose in the supervised learning setting than that of the active learning algorithms.

Having answered the first fundamental question posed in Chapter 1, is it better to have $P_R = P_S$, and having concluded that in fact P_R^* is the optimal training distribution to generate the dataset for training the learning algorithm, we move on to answer the second question. Is it advantageous to use weights to make the training set look like $P_R^\star$? We answer this question in the following chapter.

Chapter 4

To weight or not to weight

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

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

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

4.1 What makes weighting work sometimes only?

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

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

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

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

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

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

4.1.1 The effective sample size

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1.2 Weighting vs sampling

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

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

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

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

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

and the variance is given by

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

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

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

where we used the fact that

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

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

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

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

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

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

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

Notice that

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

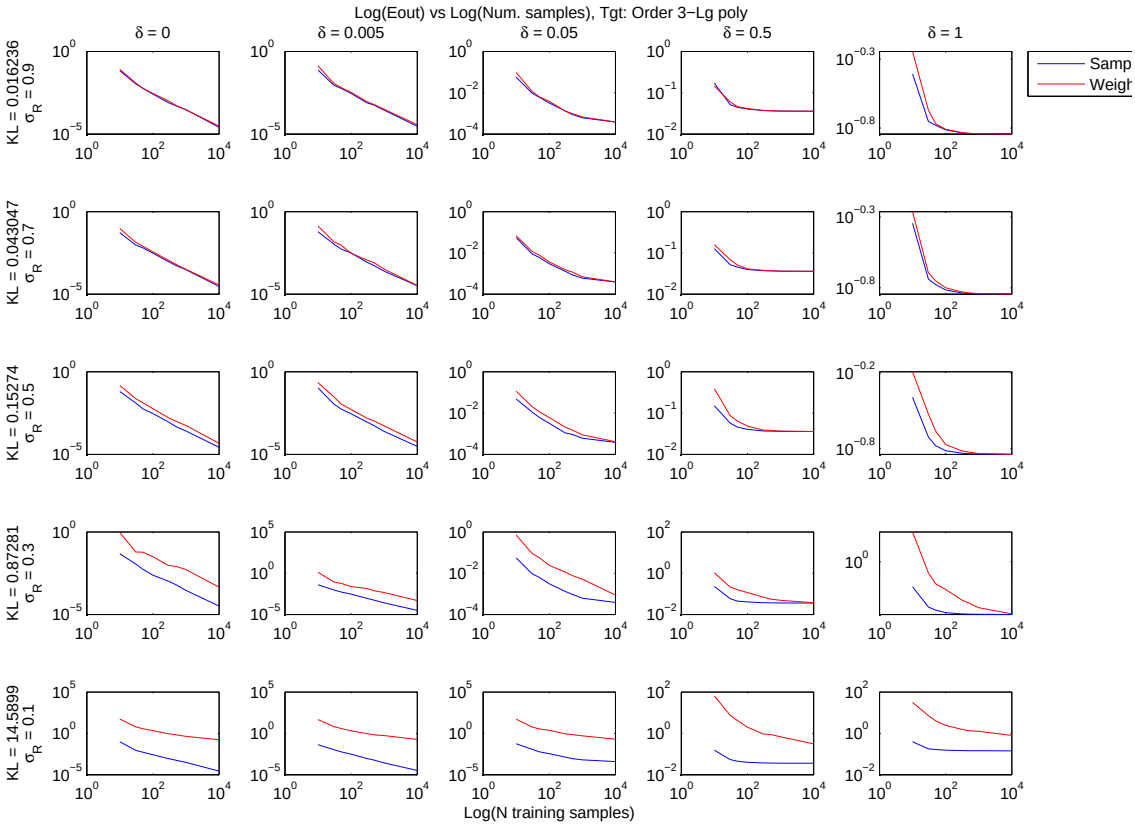


Figure 4.1: Simulations illustrating the out-of-sample error difference vs the number of training samples, when sampling data form a distribution P' and matching a sample distributed as P using importance weights $w(x) = p'(x)/p(x)$. The figure illustrates the effect as P is further from P' in KL-divergence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

4.1.3 Decomposition of the weighting effect

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

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

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

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

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

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

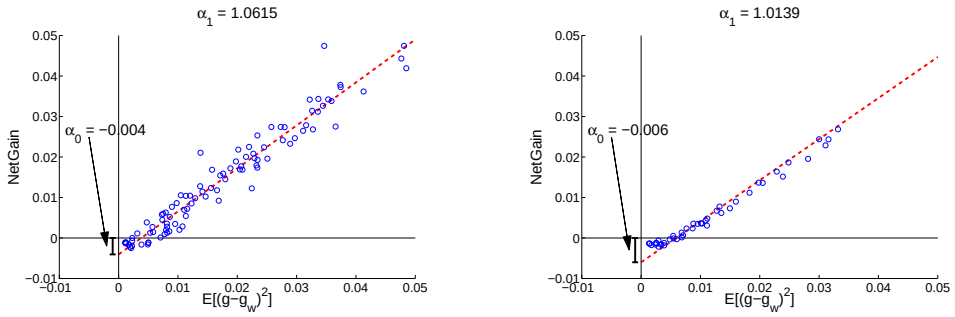
$$MatchGain = \mathbb{E}_{x\sim P_S}[\ell(g(x), f(x))] - \mathbb{E}_{x\sim P_S}[\ell(g_{P'}(x), f(x))] \tag{4.19}$$

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

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

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

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



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

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

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

Using the terms defined, it is clear that:

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

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

4.2 The algorithm: Targeted matching

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

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

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

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

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

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

Hence,

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

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

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

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

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

Algorithm 3 Targeted Weighting algorithm,

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

observation is that the problem reduces to estimating α_0.

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

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

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

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

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

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

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

4.3 Experimental results

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

4.3.1 Results on the Netflix dataset

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

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

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

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

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

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

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

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

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

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

4.3.2 Results on further benchmark datasets

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

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

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

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

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

Chapter 5

A novel class of matching algorithms

We have answered the first two fundamental questions posed in Chapter 1. Namely, we found that there is an optimal distribution from which data should be drawn to train the learning algorithm, we called it the dual distribution and described how to find it. We then established an algorithm that can determine if a weighting scheme used to change the distribution of the training data will yield out-of-sample improvement. The remaining question is: how to find such weighting scheme? This question has received great attention in the covariate shift literature, in which various algorithms have been proposed to match the training distribution to a test distribution. The idea behind these methods is to estimate the weights $w(x) = p_S(x)/p_R(x)$, in order to correct for the covariate shift bias. In the following section, we review some of the most popular methods. We then propose a new class of algorithms that allow finding the weights efficiently and show their performance on real data sets.

5.1 Previous algorithms

The problem that instance weighting algorithms for covariate shift correction solve consists of estimating $w(x) = p_S(x)/p_R(x)$. The following methods find this ratio in different ways.

5.1.1 Indirect ratio estimation via KDE

The first approach used to find the importance weight was Kernel Density Estimation (KDE). The method finds the training and test densities by modeling each distribution as a linear combination of

kernel functions around the existing samples. That is

$$\hat{p}(x) = \frac{1}{N(2\pi\sigma^2)^{d/2}} \sum_{i=1}^{N} K_\sigma(x, x_i). \tag{5.1}$$

The method is very simple to implement, only requiring a choice for the size σ of the kernels. However, the method suffers in the case of noisy data sets as the errors in estimation get amplified when the ratio is taken between two estimated quantities.

Hence, the following methods propose estimating the ratio directly, rather than trying to first solve the hard problem of estimating full densities in order to find the ratio.

5.1.2 Logistic regression methods

The next group of methods propose to build a classifier that would discriminate data coming from the training and test distributions. The model assumes that all data is drawn from a probability distribution $p(x)$, and that there is a variable η that determines if a point comes from one or the other distribution. With this assumption and using Bayes' theorem, it is possible to estimate the ratio. Specifically, let

$$p_R(x) = p(x|\eta = -1) \quad \text{and} \quad p_S(x) = p(x|\eta = 1). \tag{5.2}$$

Then, by Bayes' theorem

$$w(x) = \frac{p_S(x)}{p_R(x)} = \frac{p(\eta = 1|x)p(\eta = -1)}{p(\eta = -1|x)p(\eta = 1)} \tag{5.3}$$

To find the quantities in the right hand side, the method assumes further that

$$p(\eta = -1)/p(\eta = 1) \approx N_R/N_S \tag{5.4}$$

and finds the remaining ratio via logistic regression. The optimization problem is given by

$$\theta^* = \arg\min_\theta \frac{1}{N_R + N_S} \sum_{x \in R \cup S} \log(1 + \exp(-\eta\theta^T x)) \tag{5.5}$$

and the final estimate for the weights is therefore

$$\hat{w}(x) = \frac{N_R}{N_S} \exp(x^T\theta^*). \tag{5.6}$$

Variations of this idea are described in [13], [14], and [71]. The problem with this approach is that the assumptions made by the model are quite strong.

5.1.3 Kernel mean matching (KMM)

One of the methods that has been used in practical applications most often is Kernel Mean Matching [38]. The success of this algorithm is based on a theorem that states that two distributions are the same, if and only if the mean in all dimensions of a Reproducing Kernel Hilbert Space (RKHS) are equal. Formally, let $\Phi : \mathcal{X} \to \mathcal{F}$ be a map into a feature space and let $\mu : \mathcal{P} \to \mathcal{F}$ denote the expectation operator,

$$\mu(P) := \mathbb{E}_{x \sim P(x)}[\Phi(x)]. \tag{5.7}$$

where $\mathcal{P}$ is a probability space. Then the theorem proved in [38] states that the operator μ is bijective if $\mathcal{F}$ is an RKHS with a universal kernel $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$. In other words, for two distributions P and P', $P = P'$ iff $\mu(P) = \mu(P')$.

Hence, the authors propose to solve the following optimization problem to match the means of the samples in a RKHS:

$$\begin{aligned} \text{minimize}_w \quad & ||\mu(P_S) - \mathbb{E}_{x \sim P_R}[w(x)\phi(x)]|| \\ \text{s.t.} \quad & w(x) \geq 0,\ \mathbb{E}_{x \sim P_R(x)}[w(x)] = 1. \end{aligned} \tag{5.8}$$

In practice, the objective function of the above problem is approximated with empirical estimates:

$$\left|\left| \frac{1}{N_R} \sum_{i=1}^{N_R} w_i \phi(x_i) - \frac{1}{N_S} \sum_{j=1}^{N_S} \phi(x'_j) \right|\right| = \frac{1}{N_R^2} w^T K w - \frac{2}{N_R^2} \kappa^T w + const \tag{5.9}$$

The optimization problem becomes:

$$\begin{aligned} \text{minimize}_w \quad & \frac{1}{N_R^2} w^T K w - \frac{2}{N_R^2} \kappa^T w, \\ \text{s.t.} \quad & w \in [0, B], \quad \left| \sum_{i=1}^{N_R} w_i - N_R \right| \leq N_R \epsilon. \end{aligned} \tag{5.10}$$

Hence, the problem reduces to solving a quadratic program with inequality constraints. However, the main drawback of the method is that it is very sensitive to both the choice of the size of the kernel and the constants B and ϵ. Before Targeted Weighting was introduced, there was no method available to cross-validate different weighting schemes, and hence the choice of these parameters was problematic.

5.1.4 Parametric models for the ratio: KLIEP, LSIF, RuLSIF, etc.

Although KMM proved successful in some applications, as discussed in Chapter 4, weighting may worsen performance, and hence it was sometimes not a useful method as cross-validation was not

available (See Section 4.2). A few more methods were proposed based on the idea of estimating the ratio of the importance weight directly, by using a parametric model for the ratio. The first of these methods was Kullback-Liebler Importance Estimation Procedure (KLIEP) [67]. In this method, the ratio is modeled by the equation

$$\hat{w}(x) = \frac{p_S(x)}{p_R(x)} = \sum_{t=1}^{T} \alpha_t \phi_t(x). \tag{5.11}$$

With this model, the idea is to minimize the KL-divergence between the training and test distributions, given by

$$KL(\alpha) = \mathbb{E}_{P_S}\left[\log \frac{p_S(x)}{\hat{w}(x) p_R(x)}\right] = \mathbb{E}_{P_S}\left[\log \frac{p_S(x)}{p_R(x)}\right] - \mathbb{E}_{P_S}[\log \hat{w}(x)]. \tag{5.12}$$

Since the first term is constant with respect to the weights, the method maximizes the second term, adding the constraint that $\mathbb{E}_{x\sim P_R}[\hat{w}(x)] = \int p_S(x)dx = 1$. Using empirical estimates for the above quantities, the problem becomes:

$$\begin{aligned} \text{maximize}_\alpha \quad & \sum_{j=1}^{N_S} \log\left(\sum_t \alpha_t \phi_t(x'_j)\right) \\ \text{s.t.} \quad & \frac{1}{N_R}\sum_t \alpha_t \sum_{i=1}^{N_R} \phi_t(x_i) = 1, \quad \alpha \geq 0 \end{aligned} \tag{5.13}$$

This optimization problem is convex and the authors propose to solve it using gradient descent with constraint satisfaction at each step. The authors suggest using Gaussian kernels centered at the test points for the functions, that is $\hat{w}(x) = \sum_{j\in S} \alpha_t K_\sigma(x, x_j)$. The authors also claim that KLIEP has a model selection procedure, by cross validating the choice of kernel width, T, etc., with respect to the value of the objective function. Nevertheless, this cross-validation procedure is a measure of how good the match is between the two distributions, but is not a measure of the bottom line out-of-sample performance. As argued in Chapter 4, matching distributions using weights may actually worsen performance in some cases, depending on the tradeoff between the $MatchBenefit$ and $WeightLoss$. Targeted Weighting, on the other hand, could be used exactly for this cross-validation purpose in conjunction with KLIEP, or any of the matching methods proposed so far.

A variant of this method was proposed, in which the KL divergence was replaced by a least squares error approach. This led to least squares importance fitting (LSIF) [42]. In this method, the following

objective function is minimized:

$$\begin{aligned} J(\alpha) &:= \frac{1}{2}\mathbb{E}_{P_R}[(\hat{w}(x) - w(x))^2] \\ &= \frac{1}{2}\mathbb{E}_{P_R}[\hat{w}(x)^2] + \mathbb{E}_{P_R}[\hat{w}(x)w(x)] + \frac{1}{2}\mathbb{E}_{P_R}[w(x)^2] \\ &= \frac{1}{2}\mathbb{E}_{P_R}[\hat{w}(x)^2] + \mathbb{E}_{P_S}[\hat{w}(x)] + const. \end{aligned} \tag{5.14}$$

Once again, expectations are approximated using empirical averages, so that the optimization problem becomes:

$$\begin{aligned} \text{minimize}_\alpha \quad & \frac{1}{2}\alpha^T \hat{H}\alpha - \hat{h}\alpha \\ \text{s.t.} \quad & \alpha > 0 \end{aligned} \tag{5.15}$$

with $\hat{H}_{jk} = \frac{1}{N_R}\sum_{i=1}^{N_R}\phi_j(x_i)\phi_k(x_i)$, and $\hat{h}_i = \frac{1}{N_S}\sum_{j=1}^{N_S}\phi_i(x'_j)$. Nevertheless, the authors point out that the method is numerically unstable, so they use L2-regularization with an unconstrained problem (uLSIF). Once again, they use the objective function value in order to cross-validate the choice of kernel size

Finally, RuLSIF was proposed in which the weights are regularized, by using

$$w(x) = \frac{p_S(X)}{\beta p_R(x) + (1-\beta)p_S(x)}. \tag{5.16}$$

The need for this regularization is precisely the tradeoff pointed out in Chapter 4. Nevertheless, the authors propose finding β using importance weighted cross-validation (IWCV) [65]. However, IWCV requires knowledge of the ratio $p_S(x)/p_R(x)$. Since this ratio is unknown, cross-validation is done assuming that the estimate of the weights, found through one of the above methods, is actually accurate.

5.1.5 Discrepancy minimization

The matching algorithms discussed so far make no assumption about the learning algorithm to be used. That is, they make no assumption of what loss function is to be minimized or what the hypothesis set is. One of the most recent algorithms, introduced in [26], uses the notion of *discrepancy* [47], which leads to a tractable algorithm when we use a squared loss function, a linear model, and $\mathcal{Y} = \mathbb{R}$.

The intuition behind this approach is to find weights such that the training distribution is matched to a distribution in which the hypothesis learned will yield the same out-of-sample error as if the hypothesis had been learned with training points sampled from the test distribution. Notice that this

notion does not imply that the weights found will exactly match the training and test distributions. It will only match the out-of-sample error, which in fact is what matters to a practitioner. The hope is that by doing this, the negative effect of weighting will be minimized, as the perturbation of the weights from unity should be smaller than when we try to match exactly the two distributions.

Formally, the discrepancy is defined as follows. Let $\mathcal{H}$ be a set of functions with $h : \mathcal{X} \to \mathcal{Y}$ for every $h \in \mathcal{H}$, and let $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ define a loss function over $\mathcal{Y}$. The discrepancy distance disc_L between two distributions P and Q over $\mathcal{X}$ is given by

$$\text{disc}_L(P, Q) = \max_{h,h' \in \mathcal{H}} |L_P(h', h) - L_Q(h', h)| \tag{5.17}$$

where $L_P(h, h') = \mathbb{E}_{x \sim P}[\ell(h(x), h'(x))]$. That is, the discrepancy finds the maximum out-of-sample error difference that could result if we trained with points distributed as P instead of Q. It is not hard to see that this notion can give a bound for the error made by training with a different distribution. In fact, in [26], assuming that kernel ridge regression is used (model that uses a squared loss function, a linear model with a kernel for the non-linear transformation, ℓ_2 norm regularization for the parameters, and $\mathcal{Y} = \mathbb{R}$), the authors prove the following bound:

$$|\ell(h'(x), y) - \ell(h(x), y)| \leq 2r\sqrt{M\lambda \text{disc}(P', P)/\lambda}, \tag{5.18}$$

where $K(x, x) \leq r^2$, $L(h(x), y) \leq M$, and λ is the regularization parameter. Hence, the authors propose minimizing the discrepancy between the training and test sets. It turns out that the optimization problem is equivalent to the following semi-definite program:

$$\begin{aligned} \text{minimize}_{w,\lambda} \quad & \lambda \\ \text{s.t} \quad & \lambda I - M(w) \succeq 0 \\ & \lambda I + M(w) \succeq 0 \\ & 1^T w = 1, w \geq 0 \end{aligned} \tag{5.19}$$

where $M(w) = \sum \hat{P}_S(x) x x^T - \sum_{i=1}^{N_R} w_i x_i x_i^T$, and $A \succeq 0$ denotes matrix A is positive semi-definite. Solving this problem requires, therefore, using convex optimization packages. Since the notion of discrepancy allows regularizing the weights, the algorithm is effective. Nevertheless, its main drawback is that it is only valid for regression problems with the learning model specified above.

5.2 A new class of algorithms

We propose a new class of algorithms for matching. These algorithms were conceived for very large datasets, as it is the case with the Netflix dataset, where computational complexity is a key issue. Furthermore, as the algorithms were conceived in the recommender system setting, we wanted to be able to choose particular coordinates along which to match distributions. In recommender systems, the data usually includes only an item ID, a user ID, and the rating which is the value that the system will try to predict. The time of the rating may also be available. Under this scenario, matching training and test sets is not meaningful as both user ID and movie ID are not relevant coordinates. Rather, matching certain coordinates that can be derived from the data, such as the item popularity (known as item support), or the amount of ratings by a particular user (user support), can be desired. We begin by introducing the hard matching algorithm which is the basic building block of the algorithms presented here. A "soft" version is also presented, and finally different variants of the initial algorithm are discussed.

5.2.1 Hard matching

We recall that a sampled version of the loss function yields a variance with approximately $N_{\text{eff}} = \left(\sum_i w_i\right)^2 / \sum_i w_i^2$. Hence, as there are many schemes that can match the training dataset to a desired distribution, we pick the one that minimizes the ℓ_2-norm of w, since this will maximize N_{eff}. The reason for this is that the the numerator of the expression for N_{eff} is a constant. To see this, notice we can scale all weights by a constant and this would not have any effect on the learning algorithm. Hence we pick to normalize the sum of the weights to N_R. Here we use N_R to denote the number of points in the training set, while we use N_S for the points in the test set. To illustrate the matching condition, first consider a one-dimensional distribution. One way to match is to divide the input space into bins and match the fraction of points appearing in each of the bins for the training set and for data coming from the desired distribution.

To formalize this, let T denote the number of bins that we use to divide the input space. Let $b : \mathcal{X} \to \{1, \ldots, T\}$ be a function that indicates the bin number into which each data point x_i falls. Let $\mu, \nu \in \mathbb{R}^T$ be vectors that hold the frequency of points in each bin for the training set and for the distribution we desire to match to. That is

$$\mu_j = \frac{1}{N_R} \sum_{i=1}^{N_R} I[b(x_i) = j], \tag{5.20}$$

and similarly for ν, except that the summation is over points in the test set if $P' = P_S$. In case P' is a different distribution, such as the dual distribution, ν can be obtained by integrating the density of

P' for each of the bins. With this notation, the idea described above consists of solving the following optimization problem:

$$\begin{aligned} \text{minimize}_{w_i} \quad & \frac{1}{2}\sum_{i=1}^{N_R}(w_i - 1)^2 \\ \text{s.t.} \quad & \frac{1}{N_R}\sum_{i:b(x_i)=\tau} w_i = \nu(\tau), \quad \text{for } \tau = 1, \ldots, T \\ & w_i \geq 0, \end{aligned} \tag{5.21}$$

where $\nu(\tau)$ indicates the τ'th component of the vector ν. This is a quadratic program with linear constraints, and hence it is convex and has a unique solution. In fact we can solve for the weights analytically by constructing the Lagrangian and setting its gradient to 0. We ignore for now the positivity constraints. Notice further that since

$$\sum_{i=1}^{N_R} w_i = \sum_{\tau}\sum_{i:b(x_i)=\tau} w_i = N_R \sum_{\tau} \nu(\tau) = N_R, \tag{5.22}$$

it is not necessary to include normalization constraints for the weights as this constraint is implicitly satisfied.

The Lagrangian is given by

$$\mathcal{L}(w, \lambda) = \sum_{i=1}^{N_R}(w_i - 1)^2 + \frac{1}{N_R}\sum_{\tau=1}^{T}\lambda_\tau \sum_{i:b(x_i)=\tau} w_i - \nu(\tau), \tag{5.23}$$

thus

$$\frac{\partial \mathcal{L}(w, \lambda)}{\partial w_i} = 0 \Longrightarrow w_i = 1 - \frac{\lambda_\tau}{N_R}. \tag{5.24}$$

Notice that this means that the weights of all points that fall in the same bin must be equal. Hence, let $w_i = w_\tau$ for all $x_i \in R$ with $b(x_i) = \tau$. Then, substituting in the constraint,

$$\frac{1}{N_R}\sum_{i:b(x_i)=\tau} w_i = \frac{1}{N_R}\sum_{i:b(x_i)=\tau} N_R \mu(\tau) w_\tau = \mu(\tau) w_\tau = \nu(\tau). \tag{5.25}$$

Therefore, the solution for the weights is given by

$$w_\tau = \frac{\nu(\tau)}{\mu(\tau)}. \tag{5.26}$$

Clearly, this approach gives an approximation for the importance weighs $w(x) = p'(x)/p(x)$. Also notice that having ignored the non-negativity constraints in the problem did not change the solution,

as clearly $w_\tau \geq 0$.

Now that the intuition is clear, we move on to multiple coordinates. We want the distributions to be matched along each of the desired coordinates, and we consider projections into these coordinate independently. This allows the algorithm to grow linearly in complexity with the number of coordinates chosen, rather than exponentially in d, the number of dimensions of the input space. Generalizing the previous notation, let C be the number of coordinates we want to match along, and let $T_1, \ldots, T_C$ be the number of bins chosen along each of the coordinates. Let $b_c : \mathcal{X} \to \{1, \ldots, T_c\}$ for $c = 1, \ldots, C$ be functions that map each training point x_i to the corresponding bin number along coordinate c. Also, let $\mu_c, \nu_c \in \mathbb{R}^{T_c}$, for $c = 1, \ldots, C$ be the corresponding frequency vectors along coordinate c. The optimization problem we want to solve is given by

$$\begin{aligned} \text{minimize}_{w_i} \quad & \frac{1}{2}\sum_{i=1}^{N_R}(w_i - 1)^2 \\ \text{s.t.} \quad & \frac{1}{N_R}\sum_{i:b_c(x_i)=\tau} w_i = \nu_c(\tau), \quad \text{for} \quad \begin{matrix}\tau = 1, \ldots, T, \\ c = 1 \ldots, C\end{matrix} \\ & w_i \geq 0 \end{aligned} \tag{5.27}$$

The Lagrangian in this case is given by

$$\mathcal{L}(w, \lambda) = \frac{1}{2}\sum_{i=1}^{N_R}(w_i - 1)^2 + \frac{1}{N_R}\sum_{c=1}^{C}\sum_{\tau=1}^{T_c}\lambda_{c\tau}\sum_{i:b(x_i)=\tau} w_i - \nu_c(\tau). \tag{5.28}$$

Setting the gradient of the Lagrangian with respect to w_i to 0 yields

$$w_i = 1 - \sum_{c=1}^{C}\frac{\lambda_{cb_c(x_i)}}{N_R}. \tag{5.29}$$

We now solve for the Lagrange multipliers. We substitute in the constraint and obtain

$$\frac{1}{N_R^2}\sum_{i:b_c(x_i)=\tau}\sum_{k=1}^{C}\lambda_{kb_k(x_i)} = \mu_c(\tau) - \nu_c(\tau). \tag{5.30}$$

Notice further that for coordinate c, the above equation becomes

$$\frac{1}{N_R^2}\sum_{i:b_c(x_i)=\tau}\lambda_{cb_c(x_i)} = \mu_c(\tau) - \nu_c(\tau) - \frac{1}{N_R^2}\sum_{i:b_c(x_i)=\tau}\sum_{\substack{k=1 \\ k\neq c}}^{C}\lambda_{kb_k(x_i)}. \tag{5.31}$$

But for the left hand side, since the outer sum is over the x_i that satisfy $b_c(x_i) = \tau$, we are simply adding $\mu_c(\tau)N_R$ times the value of $\lambda_{c\tau}$. Hence,

$$\frac{\lambda_{c\tau}}{N_R} = \frac{1}{\mu_c(\tau)} \left(\mu_c(\tau) - \nu_c(\tau) - \frac{1}{N_R} \sum_{i:b_c(x_i)=\tau} \sum_{\substack{k=1 \\ k\neq c}}^{C} \frac{\lambda_{kb_k(x_i)}}{N_R} \right). \tag{5.32}$$

Rescaling the Lagrange multipliers to absorb the N_R constant, we have the following system of equations:

$$w_i = 1 - \sum_{k=1}^{C} \lambda_{kb_k(x_i)} \tag{5.33}$$

$$\lambda_{c\tau} = 1 - \frac{\nu_c(\tau)}{\mu_c(\tau)} - \frac{1}{N_R\mu_c(\tau)} \sum_{i:b_c(x_i)=\tau} \sum_{\substack{k=1 \\ k\neq c}}^{C} \lambda_{kb_k(x_i)}. \tag{5.34}$$

To solve for the Lagrange multipliers, we initialize them at 0, and use Equation 5.34 iteratively to update their values. In our experiments, the values converge in a few iterations. With these values, the weights can be readily found.

We applied this algorithm to the Netflix dataset, as this dataset suffers from the covariate shift problem. The dataset consists of around 100 million points that included the user ID, movie number, time of a rating and rating (the value to predict). That is, $R = \{(u_i, m_i, t_i, r_i)\}$. The dataset was designed in such a way that the training set included ratings from historical data, but the test set only included the most recent ratings available to Netflix. This inherently created a difference in distributions along various coordinates of the data. The coordinates included: absolute time of rating (t); time since first rating of movie (dt); number of ratings per user or 'user support' (us); number of ratings for a movie or 'movie support' (ms); time since first rating of user (ut); and order of rating among the user's ratings (un).

We ran one of the most popular algorithms used in the Netflix competition, the SVD [44], and noticed that performance actually worsened if we used hard matching. This occurred both if we matched along single coordinates, or all coordinates at once. Table 5.1 summarizes these results. This was actually not very surprising as we were aware that weights could have a negative effect on the sample 'size', and in this particular dataset, some weights needed to become very large in order to achieve the matching constraints, which worsens this effect. To alleviate this problem, we introduce a regularized version of this method. We call this method "Soft Matching", which we explain in the next subsection.

A remaining question to be answered is how to choose the free parameters T_c, the number of bins in each of the coordinates. Sometimes, the data will have a natural division for the different coordinates.

Coordinates	RMS error improvement (basis points)
t	−71
dt	−14
us	−121
ms	−147
ut	−41
un	−40
all coords.	−93

Table 5.1: RMS error improvement in basis points when the given coordinates are matched between training and test sets using the hard matching algorithm.

For example, in the Netflix dataset, movies were rated over a period of 2,243 days. Therefore, binning by day seemed a natural thing to do. Nevertheless, it is not necessarily clear from the data how many bins should be chosen. Yet, the higher the number of bins, the higher the chance that some bins will have very few points, which could lead to inaccurate estimates of $p_S(x)/p_R(x)$ for those bins. In the other extreme, if very few bins are chosen, the estimated ratio will also be inaccurate because it captures a large part of the input space. In the extreme case, the ratio will always be 1. Therefore, as a rule of thumb, in the experiments ran on UCI classification datasets and LIACC Regression datasets, we chose the number of bins to be such that on average, every bin had at least 10 points. Hence, for datasets containing in the order of 10^2 points, 10 bins were used, while for datasets in the order of 10^3, 100 bins were used. For the Netflix dataset, the bins were either: the number of days, which meant about 50,000 points per bin for the t, dt, ut, and un coordinates; the number of unique movies (17,771) for the movie support ms, which yields about 5,000 items per bin; and 3,000 bins for the user support, each one corresponding to the number of movies rated by each user, with the last bin grouping all users that rated more than 3,000 movies. This yields about 33,000 points per bin.

5.2.2 Soft Matching

In order to reduce the negative effect or $WeightLoss$ that hard matching can lead to, we alleviate the effect by softening the constraints. One way to do this is to pull the constraints into the objective function. By using free parameters that allow trading off the amount of matching desired, we can get control how close to unity we want the weights to be. We call this problem Soft Matching, which is

described as follows:

$$\begin{aligned} \text{minimize}_w \quad & \frac{1}{2}\sum_{i=1}^{N_R}(w_i-1)^2 + \frac{1}{2}\sum_{c=1}^{C}\lambda_c\sum_{\tau=1}^{T_c}\left(\sum_{i:b(x_i)=\tau}\frac{w_i}{N_R} - \nu_c(\tau)\right)^2 \\ \text{s.t.} \quad & w \geq 0. \end{aligned} \tag{5.35}$$

Notice the similarity of the above objective function with the Lagrangian of Equation 5.28. In this case, the parameters λ_c control the level of matching, rather than being Lagrange multipliers. If the λ_c are set to infinity, then the constraints must be matched exactly in order for the objective function to be finite. If on the other hand the λ_c are set 0, all weights remain equal to 1. We solve again for the weights analytically by finding the gradient with respect to the weights and setting it to 0. As it was the case for the hard matching algorithm, we initially ignore the inequality constraints. Setting the gradient of the objective function to 0 yields

$$w_i = 1 - \frac{1}{N_R}\sum_{c=1}^{C}\lambda_c\left(\sum_{j:b(x_j)=\tau}\frac{w_j}{N_R} - \nu_c(\tau)\right). \tag{5.36}$$

Define further the auxiliary variables

$$\omega_c(b_c(x_i)) = -\sum_{x_j:b_c(x_j)=b_c(x_i)}\frac{w_j}{N_R} - \nu_c(b_c(x_i)) \quad \text{for } c = 1,\ldots,C. \tag{5.37}$$

Thus,

$$w_i = 1 + \sum_{c=1}^{C}\lambda_c\omega_c(b_c(x_i)), \tag{5.38}$$

where the N_R constant has been absorbed into the λ_c's. Substituting this equation in the definition for ω_c, we can solve for this auxiliary variables:

$$\omega_c(\tau) = \frac{1}{1+\lambda_c\mu_c(\tau)}\left(\nu_c(\tau) - \mu_c(\tau) - \frac{1}{N_R}\sum_{b_c(x_i)=\tau}\sum_{\substack{k=1\\k\neq c}}^{C}\lambda_k\omega_k(b_k(x_i))\right) \tag{5.39}$$

Once again, notice the solution is parallel to the one obtained in the hard matching procedure, except that the old Lagrange multipliers, $\lambda_{c\tau}$ are now $-\omega_c(\tau)\lambda_c$. To verify that indeed in the case that λ_c goes to infinity we recover the hard matching solution, we take Equation 5.39 and multiply it

by $-\lambda_c$. We have,

$$-\lambda_c\omega_c(\tau) = \frac{-\lambda_c}{1+\lambda_c\mu_c(\tau)}\left(\nu_c(\tau) - \mu_c(\tau) - \frac{1}{N_R}\sum_{b_c(x_i)=\tau}\sum_{\substack{k=1\\k\neq c}}^{C}\lambda_k\omega_k(b_k(x_i))\right)$$
$$= \frac{1}{1/\lambda_c+\mu_c}\left(\mu_c(\tau) - \nu_c(\tau) - \frac{1}{N_R}\sum_{b_c(x_i)=\tau}\sum_{\substack{k=1\\k\neq c}}^{C}-\lambda_k\omega_k(b_k(x_i))\right). \tag{5.40}$$

Now, taking the limit of the equation as $\lambda_c \to \infty$ we obtain

$$\lambda_{c\tau} = 1 - \frac{\nu_c(\tau)}{\mu_c(\tau)} - \frac{1}{N_R\mu_c(\tau)}\sum_{b_c(x_i)=\tau}\sum_{\substack{k=1\\k\neq c}}^{C}\lambda_{k\tau}, \tag{5.41}$$

where we have set

$$\lim_{\lambda_c\to\infty} -\lambda_c\omega_c(\tau) := \lambda_{c\tau}. \tag{5.42}$$

As it is clear, we have recovered the Hard Matching systems of equations (Eq. 5.33).

We now apply the Soft Matching algorithm to the Netflix dataset and obtain the results shown in Table 5.2. As it is clear from the table, choosing certain coordinates, and for low values of λ, there is an improvement in the RMS error on the test data. This had not occurred when we applied the hard matching algorithm. It is important to note that in this dataset, it is extremely hard to obtain improvements over available solutions, as practitioners tried for two years to improve upon solutions in order to win the Netflix Prize competition. An improvement of a few basis points, as the ones shown in Table 5.2, could have meant being ahead of the pack by a significant amount.

Picking the value for the λ_c can be done through cross-validation, which can be achieved using the Targeted Weighting algorithm introduced in Chapter 4. Once we observed which values of λ_c worked well for each of the coordinates, we matched simultaneously the subset of coordinates that TW indicated were helpful. We then tried different values of λ_c and matched simultaneously the favorable coordinates, namely the user support us, the movie support ms, and the absolute time t.

Having successfully applied Soft Matching to a real dataset that suffered from covariate shift, we explore alternative formulations of the hard matching problem, on the one hand with the idea of reducing the number of free parameters to be adjusted via cross-validation, and on the other, expressing the "softening" of the initial algorithm in a principled way. We discuss these variants in the following subsection.

Table 5.2: Soft Matching applied to the Netflix dataset, with the SVD++ learning model using 50 factors. RMS improvement is given in basis points.

Coord.	λ	RMS improvement	λ	RMS improvement
t		−71		**9**
dt		−14		**0.1**
us	1000	−121	100	**5**
ms		−147		−5
ut		−41		−20
un		−40		−17
t		**7**		**2**
dt		**0.1**		**0.1**
us	50	**9**	10	**3**
ms		−0.3		**0.4**
ut		−14		−5
un		−13		−5
t		**1**		**0.3**
dt		**0.1**		**0.01**
us	5	**2**	1	**0.4**
ms		**0.2**		**0.01**
ut		−2		−0.01
un		−2		−0.01
t, us, ms	100, 100, 10	**10**	100, 50, 10	**15**

5.2.3 Hard matching with slack variables

The second variation of hard matching we consider consists of using slack variables for the constraints, rather than including a weighted version of the constraints in the objective function. This approach accounts for the fact that even if two samples come from the same distribution, their realizations will yield slightly different histograms along the desired coordinates. For this and the following problems we state the optimization problem considering $C = 1$. Natural extensions for $C > 1$ only involve summations over the number of coordinates. The optimization problem is:

$$\text{minimize}_{w_i} \quad \frac{1}{2}\sum_{i=1}^{N_R}(w_i - 1)^2 \tag{5.43}$$

$$\text{s.t} \quad \sum_{x_i : b(x_i)=\tau} \frac{w_i}{N_R} - \nu(\tau) = \xi_\tau \quad \text{for } \tau = 1, \ldots, T$$

$$\frac{1}{2} \quad \sum_{\tau=0}^{T} \xi_\tau^2 \leq K \tag{5.44}$$

The Lagrangian of this problem is given by

$$\mathcal{L}(w,\lambda,\alpha) = \frac{1}{2}\sum_{i=1}^{N_R}(w_i-1)^2 + \sum_{\tau=1}^{T}\lambda_\tau \sum_{x_i:b(x_i)=\tau}\frac{w_i}{N_R} - \nu(\tau) - \xi_\tau + \alpha\left(\frac{1}{2}\sum_{\tau=0}^{T}\xi_\tau^2 - K\right) \tag{5.45}$$

The Karush-Kuhn-Tucker (KKT) conditions of this problem yield

$$w_i = 1 - \frac{\lambda_\tau}{N_R} \tag{5.46}$$

$$\lambda_\tau = \alpha N_R \epsilon_\tau \tag{5.47}$$

$$0 = \alpha\left(\frac{1}{2}\sum_{\tau=1}^{T}\epsilon_\tau^2 - K\right) \tag{5.48}$$

$$\alpha \geq 0. \tag{5.49}$$

The KKT system also includes the constraints. Here, Equations 5.46 and 5.47 are the result of taking the partial derivative with respect to the weights and to the Lagrange multipliers of the Lagrangian. The remaining two are the complementary slackness conditions and the positivity constraint of Lagrange multipliers for inequality constraints. We eliminate λ_τ by combining Equations 5.46 and 5.47, and then, substituting in the constraint obtain

$$w_i = \frac{\nu(\tau) + \frac{1}{\alpha}}{\mu(\tau) + \frac{1}{\alpha}}, \tag{5.50}$$

where $\tau = b(x_i)$. Notice that we arrive at the same solution we had for the Soft Matching algorithm, except that now the regularization parameters, λ, are replaced by the Lagrange multiplier α. In this problem however, α is found through a different method, by using the complementary slackness condition given in Equation 5.48. Notice that if $\alpha = 0$, then the constraints are trivially satisfied. Therefore, for complementary slackness to hold, it is necessary that

$$\frac{1}{2}\sum_{\tau=1}^{T}\epsilon_\tau^2 = K. \tag{5.51}$$

Expressing this equation in terms of α, we obtain

$$\frac{1}{2}\sum_{\tau=1}^{T}\left(\frac{\nu(\tau)-\mu(\tau)}{1+\alpha\mu(\tau)}\right)^2 - K = 0. \tag{5.52}$$

We can solve for α in the above equation numerically, for example, using the bisection method. Notice that for large enough α the first term tends to 0, so that the left hand side is a negative number. On

the other hand, for $\alpha = 0$ the first term is larger than K, otherwise the constraints would already be satisfied. Hence, the left hand side, as a function of α, goes from negative to positive in the interval $[0, \infty)$. Thus, a solution can be found through numerical methods.

This formulation trades off setting the regularization parameter λ, for the parameter K. Once again K must be determined. Yet, perhaps more information is available to find this K. For example, one can generate two samples from an estimate of the distribution P_S, and compute the expected value of K through Monte Carlo simulations. Hence, this approach gives the advantage of determining the free parameters through a method different than cross-validation.

The drawback, however, is that when multiple coordinates are considered, once again we require solving iteratively an equation for the α_c's. However, this time each equation is solved through a numerical method like the bisection method, rather than by simple substitution of values as in the Soft Matching case. This slows down considerably the algorithm. Since the solutions given by this algorithm and Soft Matching are practically the same, it is not surprising that the out-of-sample improvement is the same as the one obtained when using Soft Matching. Hence, we prefer to use Soft Matching due to the computational advantage.

5.2.4 Statistical approach

An alternative formulation similar to the slack variable approach, is to use the conditions from statistics. The Kolmogorov-Smirnov test that determines if two samples come from the same distribution can be used to specify the matching condition that two samples must satisfy. This procedure tests the null hypothesis H_0, that states that two samples come from the same distribution. Let $F_m(x)$ and $G_m(x)$ be the empirical cdf's, of the two samples with m and n points, respectively. The test accepts the null hypothesis H_0 if

$$\sqrt{\frac{mn}{m+n}} \sup_x (F_m(x) - G_n(x)) \leq D_{mn}, \tag{5.53}$$

where D_{mn} is a value that depends on the statistical significance of the test. Hence, we can set up the following optimization problem:

$$\text{minimize}_{w_i} \quad \frac{1}{2} \sum_i (w_i - 1)^2 \tag{5.54}$$

$$\text{s.t.} \quad \left| \sum_{\tau=1}^{t} \frac{w_i}{N_R} - \nu(\tau) \right| \leq D_{N_R N_S}, \quad \text{for } t = 1, \ldots, T, \tag{5.55}$$

where $D_{N_R N_S}$ is given by Kolmogorov-Smirnoff tables. Hence, in the limit where T is $\max(N_R, N_S)$, this is equivalent to carrying out the Kolmogorov-Smirnoff test. We rewrite the problem to have twice

the constraints, eliminating the absolute value, and it becomes

$$\begin{aligned} \text{minimize}_{w_i} \quad & \frac{1}{2}\sum_i (w_i - 1)^2 \\ \text{s.t.} \quad & \sum_{\tau=1}^{t} \frac{w_i}{N_R} - \nu(\tau) \leq D_{N_R N_S}, \quad \text{for } \tau = t, \ldots, T, \\ & \sum_{\tau=1}^{t} \nu(\tau) - \frac{w_i}{N_R} \leq D_{N_R N_S}, \quad \text{for } \tau = t, \ldots, T, \end{aligned} \tag{5.56}$$

The Lagrangian of this problem is given by

$$\begin{aligned} \mathcal{L}(w, \alpha, \beta) = \frac{1}{2}\sum_i (w_i - 1)^2 + \sum_{t=1}^{T} \alpha_t \left(\sum_{\tau=1}^{t} \left(\sum_{i:b(i)=\tau} \frac{w_i}{N_R} - \nu(\tau) \right) - D_{N_R N_S} \right) + \\ \sum_{t=1}^{T} \beta_t \left(\sum_{\tau=1}^{t} \left(\sum_{\tau=1}^{T} \nu(\tau) - \sum_{i:b(i)=\tau} \frac{w_i}{N_R} \right) - D_{N_R N_S} \right). \end{aligned} \tag{5.57}$$

The KKT conditions yield

$$w_i = 1 + \sum_{\tau=t}^{T} \frac{\alpha_\tau - \beta_\tau}{N_R} \tag{5.58}$$

$$0 = \alpha_t \sum_{\tau=1}^{t} \left(\left(\sum_{i:b(i)=\tau} \frac{w_i}{N_R} - \nu(\tau) \right) - D_{N_R N_S} \right) \quad \text{for } t = 1, \ldots, T \tag{5.59}$$

$$0 = \beta_t \sum_{\tau=1}^{t} \left(\left(\nu(\tau) - \sum_{i:b(i)=\tau} \frac{w_i}{N_R} \right) - D_{N_R N_S} \right) \quad \text{for } t = 1, \ldots, T \tag{5.60}$$

$$\alpha_\tau \geq 0 \tag{5.61}$$

$$\beta_\tau \geq 0, \tag{5.62}$$

where $t = b(x_i)$, and the system is completed by the constraints. We are now interested in finding α_τ and β_τ in order to determine the weights. We substitute the value of the weights given by Equation 5.58 into Equation 5.59. We obtain that either $\alpha_k = 0$ or

$$\sum_{t=1}^{k} \sum_{\tau=t}^{T} \frac{\alpha_\tau - \beta_\tau}{N_R} = \sum_{\tau=1}^{k} \mu(\tau) - \nu(\tau) - D_{N_R N_S} \quad \text{for } k = 1, \ldots, T. \tag{5.63}$$

Hence, for $k = 1$

$$\sum_{\tau=1}^{T} \frac{\alpha_\tau - \beta_\tau}{N_R} = \frac{\mu(1) - \nu(1) - D_{N_R N_S}}{\mu_1}. \tag{5.64}$$

Similarly, using Equation 5.60, we obtain that either $\beta_k = 0$ or

$$\sum_{t=1}^{k}\sum_{\tau=t}^{T}\frac{\alpha_\tau - \beta_\tau}{N_R} = D_{N_R N_S} - \sum_{\tau=1}^{k}\mu(\tau) - \nu(\tau) \quad \text{for } k = 1, \ldots, T. \tag{5.65}$$

Evaluating for $t = 1$ yields

$$\sum_{\tau=1}^{T}\frac{\alpha_\tau - \beta_\tau}{N_R} = \frac{D_{N_R N_S} - (\mu(1) - \nu(1))}{\mu_1}. \tag{5.66}$$

Since we obtain for each inequality a value for $\sum_{\tau=1}^{T}\frac{\alpha_\tau - \beta_\tau}{N_R}$ with different sign, we simply check which constraint is being violated. Violation of one constraint will imply α_1 or β_1 equal to 0, and one of the above equations is not valid. Hence, we can decide what the correct value is for $\sum_{\tau=1}^{T}\frac{\alpha_\tau - \beta_\tau}{N_R}$. Once this value is known, we can uniquely determine w_1. Subsequently, w_2 can be obtained, and so on until we have w_T.

Hence, this is another method that can be used to carry out matching. The free parameter now, rather than the regularization parameters λ as in Soft Matching, or the value of K as in hard matching with slack variables, is the value of $D_{N_R N_S}$, which must be determined using Kolmogorov-Smirnoff test values. The drawback of this method is once again the extension to multiple coordinates. In this case, the system of equations given by the KKT conditions cannot be solved in such a straightforward way. Finally, we present a different approach based on a probabilistic assumption.

5.2.5 Probabilistic approach

The final variation we present uses a similar idea as the previous methods, except that it uses a probabilistic formulation. Letting ν' and ν'' represent the frequency vectors of two samples, with each component corresponding to the frequency count in each bin, we think of these as two realizations of points sampled from a distribution P. Let N' and N'' be the number of points in each sample. We ask the question, how different can these two vectors be, given that they were generated according to the same distribution. This condition will become our new matching criterion.

Specifically, we are concerned with the quantity

$$\begin{aligned} \text{maximize}_p \quad & \frac{Pr(\mu, \nu|p)}{\mathbb{E}_{\rho', \rho''}[Pr(\rho', \rho'')|p} \\ & \sum_{i=1}^{T} p_i = 1 \end{aligned} \tag{5.67}$$

where $Pr(X)$ denotes the probability of event X occurring. In this case, we are concerned about

finding the probability that μ and ν were samples generated from p, a quantity that we normalize by its expected value.

We first simplify this expression. Notice that

$$P(\nu'|p) = \frac{N!}{(N'p_1)!\cdots(N'p_T)!} p_1^{N'p_1}\cdots p_T^{N'p_T}. \tag{5.68}$$

Further, recall Stirling's approximation, namely

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n = e^{n\log n - n - \frac{1}{2}\log(2\pi n)}. \tag{5.69}$$

Substituting and canceling out terms, we obtain

$$P(\nu'|p) \approx \frac{\sqrt{2\pi N'}}{\prod_{i=1}^T \sqrt{2\pi N' p_t}} \tag{5.70}$$

We now make the following simplifying assumption

$$\mathbb{E}_\rho[\rho|p] = \left(\frac{2\pi N}{\prod_{t=1}^T 2\pi N p_t}\right)^\gamma \tag{5.71}$$

Rewriting Equation 5.67, the problem is:

$$\text{maximize}_p \frac{\left(\frac{N'!\prod_{i=1}^T p_i^{N'\nu_i'}}{(N'\nu_1')!\cdots(N'\nu_T')!}\right)\left(\frac{N''!\prod_{i=1}^T p_i^{N''\nu_i''}}{(N''\nu_1'')!\cdots(N''\nu_T')!}\right)}{\left(\frac{2\pi N'}{\prod_{i=1}^T 2\pi N' p_i}\right)^\gamma \left(\frac{2\pi N''}{\prod_{i=1}^T 2\pi N'' p_i}\right)^\gamma} \tag{5.72}$$

Grouping all constant terms into the constant K, the objective function is

$$\text{maximize}_p\, K \prod_{i=1}^T p_i^{N'\nu' + N''\nu'' + 2\gamma}. \tag{5.73}$$

Writing the Lagrangian for this problem, setting its gradient to 0, and solving for p, we obtain

$$p_i^* = \frac{N'\nu_i' + N''\nu_i'' + 2\lambda}{N' + N'' + 2\gamma T}. \tag{5.74}$$

With this value of p^*, we can now compute the maximum value of the objective function. This will in turn be a criterion for determining if two samples came indeed from the same distribution. We

denote this value L_{max}, and it is given by

$$L_{\max} = \frac{N'!N''!(2\pi N')^{\gamma(T-1)}(2\pi N'')^{\gamma(T-1)}}{\prod_{i=1}^{T}(N'\nu_i')!(N''\nu_i i'')!}\prod_{i=1}^{T}\left(\frac{N'\nu_i' + N''\nu_i'' + 2\gamma}{N' + N'' + 2\gamma T}\right)^{N'\nu_i' + N''\nu_i'' + 2\gamma}. \tag{5.75}$$

Now, this probabilistic approach to the problem reduces to finding weights such that the weighted sample and the sample from the desired distribution have a normalized probability of coming from the same distribution above certain threshold. This threshold can be set to a fraction of $L_{\max}$. That is, the optimization problem becomes

$$\min_{w} \quad \frac{1}{2}\sum_{i}(w_i - 1)^2 \tag{5.76}$$

$$\text{s.t.} \quad \frac{N_R!N'!(4\pi^2 N_R N')^{\gamma(T-1)}}{\prod_{i=1}^{T}\left(\sum_{j:b(x_j)=i} w_j\right)!(N'\nu(i)')!}\prod_{i=1}^{T}\left(\frac{\sum_{j:b(x_j)=i} w_j + N'\nu(i)' + 2\gamma}{N' + N'' + 2\gamma T}\right)^{\sum_{j:b(x_j)=i} w_j + N'\nu(i)' + 2\gamma} \geq L \tag{5.77}$$

We can take the logarithm of the constraint, and we realize that the constraint is neither convex nor concave, as it involves sums of entropies and negative entropies. To see this, ignoring terms that do not involve w, and setting $\gamma = 0.5$, we obtain

$$\sum_{\tau=1}^{T}\left(\sum_{i:b(i)=\tau} w_i + N'\nu(\tau)' + 1\right)\log\left(\frac{\sum_{i:b(i)=\tau} w_i + N'\nu(\tau)' + 1}{N_R + N_S + T}\right) - \sum_{i:b(x_i)=\tau} w_i \log\left(\frac{\sum_{i:b(i)=\tau} w_i}{N_R}\right) \geq \ell_0. \tag{5.78}$$

Hence, although we begin with a more principled criterion to determine the extent of matching, the resulting optimization problem is harder to solve. Particularly, the extension to multiple coordinates is not as straightforward as for the previous problems. Secondly, due to the non-convexity of the problem, we cannot guarantee that once we find a minimum, it is the global one. Finally, we see that once again the free parameter ℓ_0 has to be chosen, although the order of it can be determined by computing $\log(L_{\max})$, if we use an estimate for p.

This concludes our exploration of matching methods, answering the third fundamental question we posed in Chapter 1. In practice we use Soft Matching as it not only gives tangible improvements in real datasets, but it is also the least demanding in terms of computation. Since we were precisely interested in finding a suitable method for large datasets, this gives Soft Matching the edge over these algorithms.

Part II

Behavior Analysis with Machine Learning

Chapter 6

Supervised behavior classification

Shifting gears, we now move on to an application of Machine Learning in ethology, or behavior analysis. Biologists study behavior in animals with various objectives. Among others, understanding the human brain has become a primary objective not only for science but also for the US government with the BRAIN initiative [9]. In order to understand such a complex organ as the brain, biologists begin this endeavor by studying the brain of simpler organisms, such as fruit flies (Drosophila megalonaster). To do this, scientists use a large number of techniques in genetics and neuroscience that allow them to identify particular neurons, connections, substances, etc., that are responsible for animal behavior.

Nevertheless, to analyze the behavior of flies, biologists must record thousands of videos of animals to quantify desired behaviors. Analyzing these videos is extremely time consuming, as biologists need to determine how many times certain behavior occurs, as well as extract other properties of the behavior such as duration, latency, etc. Annotating videos can take almost three times as much as the actual duration of the videos, as these have to be viewed in slow motion [5]. This is when Machine Learning comes in handy, as classifiers can be trained to detect specific behaviors from videos of animals.

In this chapter, we describe a classifier that we developed in order to detect a particular fly behavior, "unilateral wing extensions" (UWE). Figure 6.1 shows extracts of a sequence that is classified by biologists as a UWE. The classifier was needed in order to analyze thousands of videos that would take more than few years to annotate by hand. With an automated classifier, the time to analyze the videos became negligible. The work required only involved labeling by hand a small fraction of the videos in order to train and test the system. The resulting classifier was used in the study described in [6]. In that paper, biologists discovered the specific neuron and substance that make male flies aggressive, in situations where there is no competition for any resources, such as food, females, or territory. The hope of this investigation is that aggressive behavior in humans might be understood, especially to be able to design appropriate pharmacological products that could be used in mental

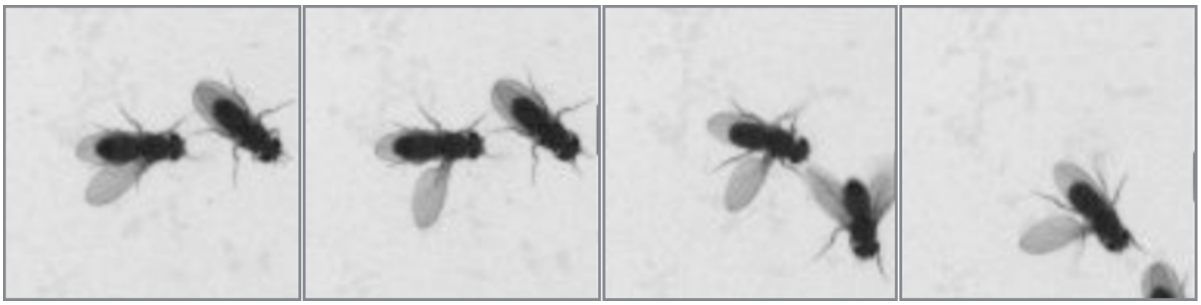


Figure 6.1: Sequence depicting a Unilateral Wing Extension

disorders such as Post Traumatic Stress Disorder, Hyperirritability, etc.

Previous work in the area is described in [31], where different behaviors of flies were detected, mainly through the use of manual filters. The method, however, was not yielding reliable performance in this dataset, reason why a new method was called for. Other work in behavior classification included behavior classification in mice as described in [21] and [39]. However, as we will describe shortly, these methods based on structured output classification were not suited to this specific task where very few labeled positive samples were available, and the quality of the videos varied greatly across the experiment. Some more recent work in behavior classification was published after our work was done, which describes methods for general purpose behavior classifiers (See [33] and [41]).

In the following sections we describe the input used for the system, the learning algorithm chosen and implementation details, the post-processing stage, and the resulting performance of the classifier.

6.1 Pre-processing stage

The original format of the data was video recordings in grayscale, on movies filmed either at 30 or 200Hz. In order to be able to apply any learning algorithm to it, the videos had to be processed first using Computer Vision techniques to extract the trajectories of the flies. Working with the gray scale pixel values would be hopeless, given the number of input dimensions and the small amount of labeled positives. Instead, the Computer Vision system extracts the time series that describe the position of the fly and its various parts. To do this, we used the tracker software described in [33].

This tracker fits an ellipse to the body of each of the flies, and also it detects the position of the tip of the wings and legs. Hence, the position of the fly is uniquely determined by the centroid of the fly, the length of the two axes of the ellipse, the ellipse orientation, and the position of wing and leg tips. Finally, invariant features with respect to position and orientation are derived. These include the minimum wing angle, maximum wing angle, axis ratio, among others, which are crucial for detecting wing extensions. The resulting features and derived features, that included first and

second time derivatives resulted in 36 features that describe each frame of the input.

6.2 Learning algorithm

The data set available for training consisted of five twenty minute movies filmed at 200Hz, which were annotated manually for this task. These movies contained only about 4,000 frames labeled as wing extensions, in a dataset that included 480,000 frames. To add complexity to the problem, videos in which the system was going to be tested on had been filmed across several years, and most of them consisted of lower resolution videos filmed at 30Hz with different lighting conditions.

Hence, given the few positives and the level of noise of the data, we decided to take the simplest approach which involved using a frame-by-frame classifier. By taking this approach we were able to use the 4,000 frames as positives, rather than only about 100 positive bouts of wing extensions, the number available in a more complex structured output method. We also chose the simplest learning algorithm to avoid overfitting. We used a squared loss function for the error, weighting positives heavily to account for class imbalance, and used ℓ_2 regularization for the parameters. The model was a linear hypothesis set with non-linear transformation of the features. For the linear transformation, we used a polynomial kernel of degree 2. A more complex kernel was avoided to avoid overfitting to the data. This learning model is the same one as the one described in Appendix A, with the specific non-linear transformation described.

6.3 Post-processing stage

The result of the linear classifier was extremely successful in terms of false negatives, as it detected every wing extension in the test set. Nevertheless, the false positive rate was relatively high. The first post-processing stage that was done was to smooth the result of the classifier with an averaging kernel over the time series. Biologists had determined that the UWE should take at least 130 milliseconds (4 frames in 30 Hz) movies, so that discontinuous frames that were marked as positives were not counted as such. This simple hint allowed reducing the false positive rate significantly.

After watching the remaining false positives that the classifier was giving, it was clear that there were some wing threats (bilateral wing extensions) that were being labeled as positives. Also, events where the fly was grooming itself with the wings were being erroneously labeled as positives. Another case occurred when the flies performed wing extension towards their reflection on the wall rather than towards the other fly. Finally, some mistakes were also due to errors in the tracker when the flies stood up against the wall of the dish, so that the model used by the tracker confused the wings with other parts of the fly. Since the false positives could be grouped and understood easily, further filters

to the data were applied. A threshold for the minimum wing angle was set, to avoid bilateral wing extensions (30 degrees). To avoid the grooming events, a minimum threshold for the maximum wing angle was also set (62 degrees). The facing angle between the flies was set to at least 35 degrees to avoid UWE not directed to the other fly, and the axis ratio of the fly was maintained at a minimum of 0.8 to avoid tracker errors from standing flies. This approach does not lead to overfitting in this particular case, as we chose a model with very few parameters compared to the number of frames in the training set.

Applying these filters that were determined by biologists, the final classifier had a recall of 89.4% and a precision of 87.1%. (The recall is the fraction of the true positives detected by the algorithm, while the precision is the fraction of true positives detected divided by both the true detected positives and the false positives). The final test was done on previously unseen movies by the system, which included 113 manually scored UWE, out of which 101 were detected, and 15 false positives were reported [6].

Hence, this simple Machine Learning approach allowed analyzing thousands of videos. Approaches like this have been improved upon in the last few years, in which classifiers are developed for general purpose behaviors, as in [33] and [41]. We decided to explore further the problem of behavior classification, by considering a slightly different setup. Could we discover behaviors from videos of animals in an unsupervised way, that is, without searching for behaviors previously labeled as such by biologists? Answering this question led to the results presented in the following chapter.

Chapter 7

CUBA: Caltech Unsupervised Behavior Analysis

Up to now, Machine Learning has permeated behavioral biology with methods as the one described in the previous chapter, where the task of classifying fractions of videos into specific behaviors fits perfectly under the supervised learning scenario of learning systems. Yet, unsupervised methods had not been widely applied to this problem. A recently published method analyzes the locomotion of fly larvae using unsupervised methods [69]. The study aims to form a taxonomy of the types of behaviors that can be observed in these larvae. Another method analyzes Caenorhabditis elegans worms and the study introduced the so-called eigen-worm shapes. These shapes constitute a dictionary of behavioral motifs that are able to describe the locomotion of the worms. The description in this space constitutes a fingerprint for each of the worms. The fingerprints are then compared between different mutant genetic lines and wild types of worms.

Nevertheless, these two methods were the only studies found in the literature that apply unsupervised learning methods to the problem of behavioral biology. In this chapter, we study the problem of analyzing behavior in an unsupervised manner. The goal is to be able to analyze automatically large quantities of videos, and draw conclusions in an unbiased way, by letting data speak for itself. The goal is that the method will aid biologists in testing hypotheses, as well as discovering patterns in the data that were previously undetected.

This study led to a system which we call CUBA (Caltech Unsupervised Behavior Analysis). The method extracts basic units that describe the motion of the animals, which we denote as movemes. A moveme, as defined in [5] is the simplest pattern that is associated with a behavior. Movemes are then compounded to form actions. Finally, a concatenation of actions forms an activity. CUBA is able to extract such movemes. The abstraction can then be iterated to form actions. We then compare the behavior of animals in this space, and cluster them into meaningful groups that perform the same activity. With this abstraction, it is possible to understand the patterns that arise in biological

experiments, as well as to use this as a mathematical tool to test hypotheses and discover new patterns. One of the achievements of CUBA is that it was able to discriminate, without supervision, different genetic lines of flies.

We describe CUBA in the following sections. In the first section, we start by defining the problem more concretely, and explain the goal in the context of a few biological experiments where we applied our work. We then describe the various methods used from Machine Learning that constitute CUBA. Finally, we present the results obtained in two different datasets.

7.1 Problem statement

The goal of the method is to analyze behavior of animal videos, using unsupervised machine learning techniques. The term behavior analysis may be ambiguous, unless we specify exactly what we mean by it. Ethology, the branch of biology that studies behavior, is concerned with various aspects of behavior, such as descriptive, causal, genetic, and evolutionary [5]. The first step in the analysis is therefore to be able to have an accurate description of each behavior. With such descriptions it is then possible to test hypotheses about causality, or to find differences between genetic lines, analyze evolutionary changes, etc. Hence, we focus first on the descriptive aspect.

When describing behavior, it is important to answer basic questions such as what is happening, how it is happening, and where it is happening. These answers vary depending on the time scale used in the description. Hence the description of behavior should begin by identifying the simplest meaningful patterns or movemes, and then be able to group movemes into meaningful structured actions. At a larger time scale, actions should be grouped into activities or stories. This hierarchy is analogous to the one used in natural language processing. When we want to understand a sentence, we would like to begin by discovering phonemes, then words, then full sentences, and so on.

With this in mind, our definition of behavior analysis can be decomposed in a few steps. First, given a set of time series that represent trajectories of animals, we want to identify typical coherent, meaningful patterns at various time scale resolutions. Second, we would like to find a hierarchy that allows grouping these patterns into more complex patterns in order to understand behavior at larger time scales. Finally, having abstracted the trajectories in this new space of meaningful patterns, we want to detect common patterns across individuals, detect clusters, find outliers, etc.

To clarify our goal, we describe a biological experiment that was the subject of study in [35]. The experiment aimed to understand if flies actually showed a fear-like response, when an arousing stimulus such as a shadow was repetitively presented. The experiment varied parameters such as the number and frequency of shadows presented, as well as evaluated different conditions such as having the experiment with single and multiple flies at a time, or having an additional attractive resource

such as food in the experiment. Taking this experiment as an example, a successful unsupervised analysis would first identify the movemes which are very basic units that describe the locomotion of the fly, such as slow walking, fast walking, accelerating, resting, hopping, etc. Putting together these movemes, actions can start to emerge. For example the action of escaping can be formed with the movemes of accelerating–flying/hopping–decelerating. An action like feeding could be described by movemes of slow walking–stopping (on food)–slow walking. Finally, at a higher level, an activity or story could describe a fly feeding until shadows pass, then jumping off and moving to the open space, before finally returning to the food after spending some time in the boundary enclosing the arena.

Once we have a full description of what each fly is doing, the abstraction could allow finding similar types of flies, by looking at flies that conform to the same story. Clustering similar flies would help determining what are the typical behaviors of flies in the experiment. Also, it would be possible to identify flies that are behaving differently from the norm, or perhaps identify previously unseen behaviors. This abstraction is therefore useful in summarizing and describing what is occurring in the experiment. Furthermore, the analysis becomes a tool to both formulate and test new hypotheses to understand what is occurring in the experiment.

The above analysis can be done with any type of biological experiment where trajectories of animals can be obtained using a computer vision tracking system. The goal in each experiment might be different, but the descriptive analysis provided by the system will remain the same. Giving biologists the power to summarize and cluster data, as well as to detect patterns in the experiments, is the ultimate goal of the system. Having clarified this goal, we present in the next section the various techniques used in order to construct such a system.

7.2 The method

As described before, the system performs the behavior analysis at three different levels. First it must discover meaningful patterns in the data at the smallest time scale. Then, it must be able to group these patterns or movemes into more complex actions and stories. Finally, the system summarizes the data, clusters trajectories into meaningful groups, detects outliers, etc. We present now the methods used at each of these stages.

7.2.1 Detecting movemes

In order to approach the problem of detecting movemes, we made a simplifying assumption about the animals. We think of them as finite state machines that execute a certain action depending on their internal state. For example, a fly that moves into a “hungry state” would most likely perform actions

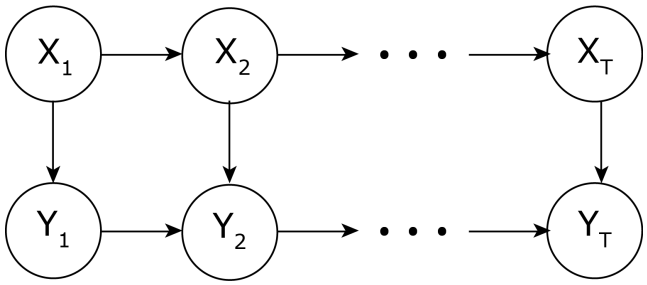


Figure 7.1: Graphical representation of a Hidden Markov Model

of finding food and eating. In a similar way, a fly in an "aroused state" would most likely try to fly and escape. Such model lends itself to a formalization into a Dynamic Probabilistic Graphical Model (PGM), where each node in the model represent a state in the animal in time, and each state has a probability distribution over the possible outcomes of the observed variable. The advantage of this probabilistic approach is that it allows modeling the inherent variability in the locomotion, actions, etc., of individuals, while allowing to group and abstract similar movemes or actions into a state that can have a semantic meaning. This approach has been taken by biologists before, for example, as in [4], where a PGM is used to try and study emotions in animals.

One PGM that lends itself particularly well to our task is a Hidden Markov Model (HMM). An HMM is a model for a stochastic process, in this case the time series. The process is described by hidden states, and at each hidden state a probability distribution determines the observed output, where the output is the time series. Hence, observing the time series allows inferring what the most likely sequence of hidden states is in this process. This fits the description of actions being the result of an internal state of the animal. The sequence of internal states or hidden states that the animal goes through, are reflected by the observed variables such as velocity, acceleration, etc., of the animal. A second important property of the HMM is that it takes into account the temporal relation of the time series. This is the case as the model assumes that the transition between states depends on the previous state of the system. Furthermore, the HMM assumes that the probability of being at any given state is independent of all previous past states given the observed variable at that state and the immediately previous state. This assumption allows us to include the time dependence, but it also simplifies the model considerably, so that inferring the parameters of the model is computationally efficient. Higher-order Markov Models can also be used but introduce a high computational cost. The graphical representation of this model is shown in Figure 7.1.

We introduce some notation that is used to specify the HMM. Let X_t be the hidden state at time t, with $X_t \in \{1, \ldots, Q\}$ where Q is the number of hidden states in the model. Let Y_t be the time series we observe, for time $t = 1, \ldots, T$. X_t can be a value or a vector of observed features,

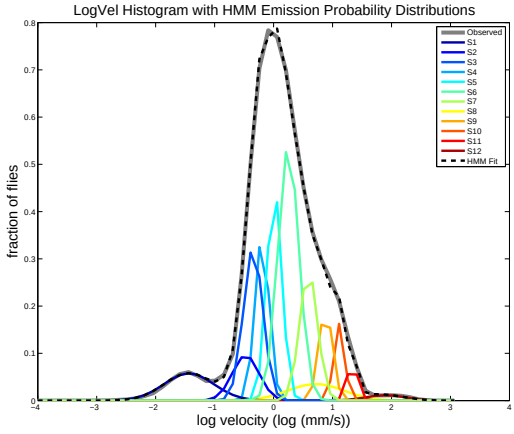


Figure 7.2: Histogram of log-velocities and emission densities for a subset of the "Fear in Flies" dataset

such as velocity, acceleration, wing position, etc. Let $A \in [0,1]^{Q\times Q}$ be the Transition Matrix, where $A_{ij} = P(X_{t+1} = j | X_t = i)$. Let B_q be the Emission Probability of state q, so that $Y_t \sim B_{X_t}$. Finally, let $\pi \in \mathbb{R}^Q$ be the initial state distribution, with $\pi_i = P(X_0 = i)$. Then, a HMM is completely specified by the transition probability matrix, the emission probabilities and the initial distribution, and we denote it by $\lambda(A, B, \pi)$.

Now, in order to learn a HMM, two parameters need to be specified: the number of hidden states Q, and the form of the emission probabilities B_q. As we will show in Section 7.3, we can determine Q through cross-validation. We learn λ for different values of Q using a subset of the data (i.e. training data). Then we use an unseen subset of the data to compute the likelihood of the trajectories given the observed model. As it will become apparent, as Q becomes larger the model no longer increases significantly the likelihood of the data but we start getting diminishing returns by adding complexity to the model. Using an Occam's razor principle, we chose the lowest value of Q that will yield a high likelihood of the data. For small datasets, in fact increasing the number of states will lead to overfitting and hence the likelihood will start decreasing.

The second choice is the parametric form of the emission probabilities. In our experiments, we used Gaussian densities, although other choices are suitable, such as t-distributions, exponential distribution, etc. This choice should be made depending on the data to be analyzed. For example, for data collected in [35], which we will refer to as the "Fear in Flies" dataset, a histogram of the log-velocities of the data yield a distribution that could be approximated by a Mixture of Gaussians, as exemplified by Figure 7.2. In this plot, the gray line indicates the distribution of the data (ignoring the time-stamp of the data), while each colored distribution represents a particular Gaussian

distribution with different parameters. Each of these are the emission probabilities of the estimated HMM for this particular subset of the data. The mixing component of the model, or weights for each component, is given by the steady-state probabilities of the transition matrix of the HMM. The dotted line represents the resulting distribution when these Gaussian components are added together. Hence, the distribution of this particular parameter in log-space is suited for Gaussian emission probabilities, while the distribution for other parameters may require a different type of emission probabilities. Once these choices are made, we proceed to fit the HMM to the data. For all our experiments that require fitting HMMs to data, we used the code in [50].

Once we fit the HMM, $\lambda(A, B, \pi)$, the movemes are the small units in the time series that get assigned to the same hidden state. These movemes are very short, usually in the order of 1 to 10 frames. The hidden states give semantic meaning to each of this movemes. For example, when fly trajectories are analyzed, the hidden states represent the following movemes: stationary, slow walking, slow walking on food, walking, walking on the boundary (usually named thigmotaxis), accelerating, among others. Depending on the number of states chosen for each subset of the data, there are more or less hidden states than the ones described above, so that there is a finer or coarser granularity to the described movemes.

Having extracted coherent meaningful snippets of the time series into what we call movemes, we move on to discover actions.

7.2.2 Detecting actions

In the context of behavior analysis, actions are defined as a combination of movemes that occur in the same order [5], and as in any other context, they can be described by a verb. In order to detect actions, we follow the same approach as the one used for detecting movemes. When movemes are found, the initial observed variables are abstracted into sequences of hidden states. This abstraction groups similar snippets of the time series into the same hidden state, and also transforms the initial continuous time series into a time series of discrete states. Yet, this abstraction does not have to stop at this level. We can further fit a HMM to the new time series in this discrete hidden-state space. The idea once again is to group similar common sequences into single states that will now represent actions.

The details of this process are exactly the same as the one described in the previous subsection, except that now there is no need to choose a parametric form for the emission probabilities. In this case, the emission probabilities are probability mass functions in $[0, 1]^Q$. That is, at state $Q_i^{(2)}$, the probability mass function will determine the probability of observing any one of the 1 to Q states of the first level HMM. Yet there is still a free parameter that needs to be set, which is the number of

hidden states that this new HMM will have. We will call this $Q^{(2)}$. The method for finding $Q^{(2)}$ is the same as the one used for finding Q, via cross-validation. In this case, not only will the likelihood of the data saturate, but it will also be evident that if the emission probabilities for these new states assign all the mass to a single state in the first level HMM, then there is no abstraction being done by the second layer. Rather, we are actually interested in emission probabilities that will have non-zero probabilities of emission in more than one of the states of the first HMM.

To illustrate this, if we fit a HMM to the output of a HMM that is fitted to log-velocity data of flies, a common state that emerges is one in which the emission probability mass function has a large mass for the hidden states that represent the movemes of accelerating/decelerating and hopping. Thus, the concatenation of an acceleration, a hop, and a deceleration are grouped together into a single action. Hence, these new hidden states will be considered as actions.

As it may be clear to the reader, this process can be repeated as many times as desired in order to abstract actions at larger resolutions in time. The experiments we present considered 2-level HMMs, but the method can be applied using as many levels as desired. Now, in order to detect an even higher level degree of abstraction, we move from analyzing single trajectories to finding similarities between trajectories. We explain this concept in the following subsection.

7.2.3 Finding stories

The last level of abstraction that we consider is what we call stories or activities. To do this, rather than simply adding more levels to our HMM model, we compare the full time series of individuals, in order to find clusters of similar individuals. Once a cluster is found, we can represent it by its medoid (the point closest to every other point in the cluster), and the concatenation of actions that constitute this trajectory will constitute a story. As will be shown in Section 7.3, these stories have clear interpretations.

The challenge of clustering the time series lies in quantifying how similar or different two trajectories are. To do this, we begin with the output of the HMM rather than with the original trajectories. The output of the first- or second-level HMMs is easier to deal with as the inherent variability and noise in the observed time series is naturally smoothed out by the HMM. We can now think of the time series in the hidden-state space as vectors in $\{1, \ldots, Q\}^T$. In principle, we could find the distance between two points by using the Euclidean norm of the difference between pairwise vectors. Nevertheless, if we are aiming to abstract a common story between a group of individuals, it is very unlikely that even if two individuals fall into the same story, that they will be at the same state at the same exact time. Not only this might not be the case, but even if two of these vectors follow the same exact sequence of states, it is also possible that the each individual spends different amounts of

time in each state. These two differences may not make the story different, and hence for this reason we want to use a distance that is more flexible than the Euclidean distance of the two vectors.

One common distance that has been used in the analysis of time series is Dynamic Time Warping (DTW) [12], introduced more than twenty years ago. Although technically this is not a distance as it does not satisfy the triangle inequality, it has been used due to the fact that it yields small distances between time series that are slightly distorted in the time axis. The technique has been successfully applied in the speech recognition community [12], in gene alignment as in [1] and [8], in medical applications such as cardiology [22], among others.

In our domain, the advantages of using DTW vs Euclidean distance are similar as those in other time-series applications. We would like to group time series that have small variations in the duration of each of the states, but that in general have a similar sequence of states. Another advantage of DTW is that the algorithm is flexible enough to allow defining a particular cost between each pair of states. In our case, since the numerical value of the hidden states may not be meaningful, we are able to design better cost matrices. The simplest of these approaches sorts the state numbers by the mean of the emission probabilities, so that the order of the states reflects the distance between two states. More sophisticated measures such as the KL distance between the emission probabilities of each pairwise state can be more descriptive.

In short, the DTW distance between two time series is found by comparing at each time step the two time series, and deciding if it is less costly to insert or delete (i.e., warp) frames in one of the time series, in order to make the corresponding cost at each step as low as possible. Dynamic programming is used in order to find this optimal trajectory efficiently, and the cost of finding this distance is $\mathcal{O}(T_1 T_2)$, where T_1 and T_2 are the lengths of each of the time series. Fast approximations to DTW have been proposed in [59] and [3] which reduce the cost to be nearly linear in the length of the trajectories rather than quadratic.

Once distances are computed, we can use any clustering algorithm such as k-means in order to find meaningful groups of trajectories. Since there is no sense for a centroid in this space, that is, an average trajectory, we use the k-medoids algorithm which finds points in the dataset to be the centers of mass of the clusters, and assigns the remaining points to the nearest cluster. As with the k-means algorithm, the solution is only a local minimum for the NP-complete problem of finding the best assignment, so we restart the algorithm many times to obtain a good enough clustering. As with every clustering algorithm, finding the right number of clusters is non-trivial, but we rely on the same methods used widely in practice. For example, plotting the average size (diameter) of the obtained clusters, versus the number of clusters allows choosing a value for k when the average size of the cluster saturates. The resulting clusters will constitute the various stories that can be found in the

data, and the medoids of the cluster will be the representatives of each of these stories.

Having described the various stages of the method, we now present results in two different datasets: the Fear in Flies dataset [35] and the Fly Bowl dataset [20].

7.3 Results on real datasets

The machinery described in the previous section can be used in different ways according to the specific dataset used. In this section we apply CUBA to two datasets that study flies. Each dataset served a different purpose in the biology community and in the same way we use the machinery of CUBA in different ways in order to come to useful conclusions concerning behavior.

7.3.1 CUBA in the Fear in Flies dataset

As described before, the Fear in Flies dataset was the result of a new setup, described in [35], that allowed recording videos of flies in a closed arena, and where a servo-motor controlled paddle produces shadows to stimulate the flies. The idea of the setup was to demonstrate fear-like behavior in fruit files. The dataset consists of recordings of movies that test various conditions of the experiment. It includes arenas with and without food, experiments with a different number of shadows presented, experiments where the interval between shadows or shadow frequency changes, experiments with different genetic lines tested at different temperatures, experiments with different levels of starvation of the flies, among others. Figure 7.3 shows a typical frame of the movies contained in the dataset. This arena shows 10 flies, the food patch in the center (darker region), and the paddle passing on top of the closed arena.

In this section we present results of applying CUBA to subsets of this data. The results are by no means exhaustive but are presented to illustrate how CUBA can be used as a tool for behavior analysis for biologists.

In order to apply CUBA to the dataset, we first need to define what are the observed variables. For this experiment, a Computer Vision tracker algorithm was developed in [35], which outputs the position of each fly at each frame of the movie. Due to the resolution of the movies, as well as to the number of flies in each preparation (usually 10), it is not possible to extract detailed pose features as the ones we describe in Chapter 6. Rather, all the information available is the position of the fly at each time step. A simple feature that can be derived from the position is the velocity of the fly. When we refer to velocity we are referring to the speed in the filmed plane. This feature has the advantage of including temporal correlation between frames, as well as being invariant with respect to the position of the fly. Figure 7.4 shows a fly trajectory as output by the tracker, and the corresponding log-velocity time series. A log-scale is used for the velocity due to the significant difference in speeds a fly can

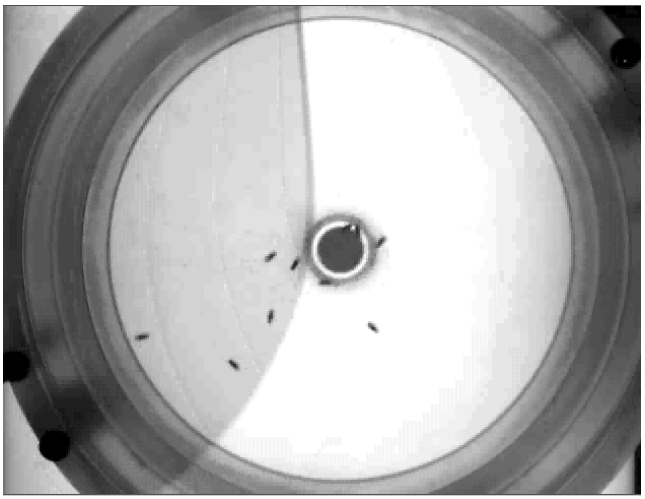


Figure 7.3: Fear in Flies dataset setup. The image corresponds to a frame of the videos captured and depicts the closed arena with a food patch at the center, 10 fruit flies, and the servo-motor controlled paddle that arouses the flies

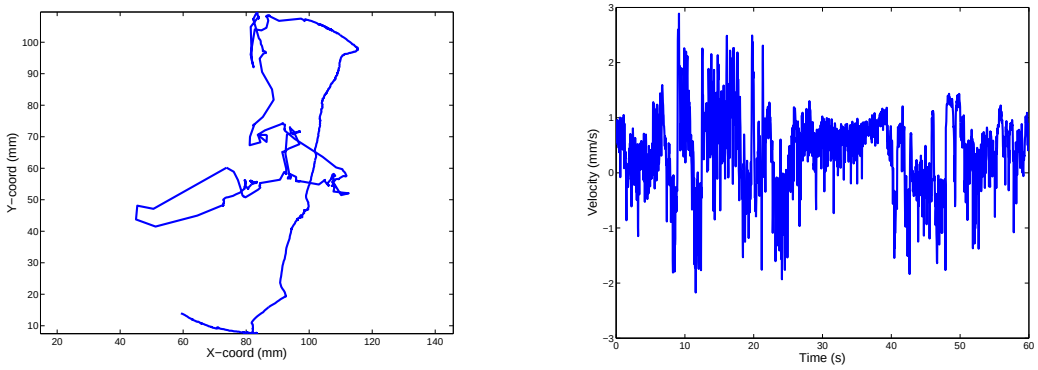
have, for example, between slow walking and flying.

As it is evident from the velocity plot for a single fly, these time series are very noise and therefore hard to understand. For this reason, extracting meaningful patterns or movemes requires the use of the techniques described in the previous section. The first step of CUBA is to learn the movemes using a HMM. Figure 7.5 exemplifies a typical model learned. This model was learned using 150 time series of flies in experiments with food, both for single and 10-fly preparations, as well as for a varying number of shadows.

The number of states was chosen through cross-validation of the likelihood on unseen data. Figure 7.6 shows the likelihood for different models used. As it is clear, the likelihood saturates around 10 to 12 hidden states. 12 states were chosen above 10 in this example, as the 12-state model was able to capture the faster state (small tail in the log-velocity distribution plot) more precisely than the model with 10 states.

With the model learned, the time series of velocities can now be transformed into time series of hidden states, where these states constitute the most probable sequence of hidden states given the model learned. Figure 7.7 shows a few sample trajectories in the hidden-state space. Each state is represented by the same color that was used for the emission densities in Figure 7.5. The plot also color codes the position of the fly by showing white, gray, and black colors on top of the time series to indicate open space, food, and boundary, respectively.

Once the number of states is chosen, it is also possible to give semantic meaning to each of the states. These correspond to movemes. By watching the movies and following the corresponding state



(a) Tracked trajectory of a single fly

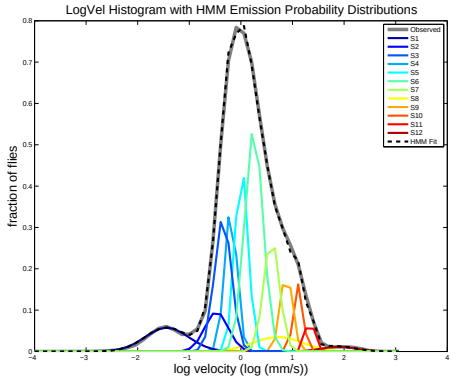
(b) Log-Velocity time series of the fly

Figure 7.4: Trajectory and log-velocity of a single fly as output by the tracker

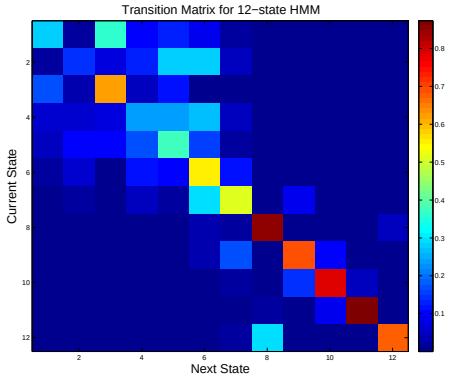
of the flies, it is clear that States 1 to 4 (dark blue states) are given to the fly when there is no visible motion. State 5 is assigned to the fly when it is moving very slowly. State 6 (green) is most frequent when flies are walking very slowly, usually in the food patch. The orange states, 10 and 11, are assigned to the flies when they perform a short fast walk. The dark red state, state 12, is assigned to the flies when they hop/fly. State 8 (yellow) corresponds to an acceleration state, as it is always assigned to flies just before and after a hop/fly event. This explains why the emission probability density function of this state has a very large standard deviation with respect to the other states.

The first major achievement of this analysis, is that it provides automatic classifiers for simple moveme/actions such as hopping or freezing. The work in [35] uses thresholds set by hand in order to detect events of hopping and freezing, but using CUBA, there is no need to fix a subjective threshold value to detect such behaviors. Rather, state 12 represents the hopping bouts, while states 1 and 2 represent the freezing states.

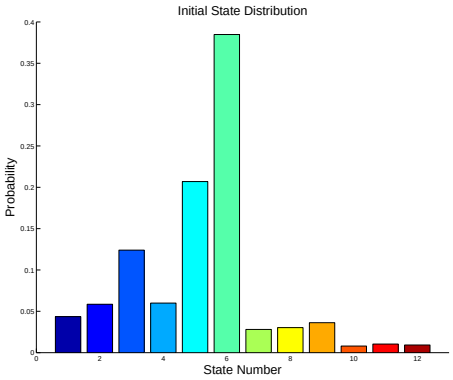
Once the movemes are found, we move on to compounding movemes by running a new HMM on the output of the first-level HMM. A typical outcome is shown in Figure 7.8. The figure shows the emission probability mass function for each of the second-level hidden states. In this example we use $Q^{(2)} = 7$. From the plot, a few actions can be inferred. State 7 has an emission probability that gives most of the weight to the hopping states and the acceleration state. Hence, this action groups acceleration - hop/fly - deceleration into a single action. State 6 has an emission probability that only gives weight to the two fast walking states. Hence, the action of fast walking groups the two previously discovered movemes into a single action. Similarly State 5 groups together the movemes of fast and slow walking. Another interesting action is that of State 3, in which the stationary states and slow walking states of the first level HMM are grouped together. This is a common action that is



(a) Log-velocity histogram of time series and emission densities B



(b) Transition matrix of HMM A



(c) Initial distribution π

Figure 7.5: Typical HMM model $\lambda(A, B, \pi)$ learned for the Fear in Flies dataset

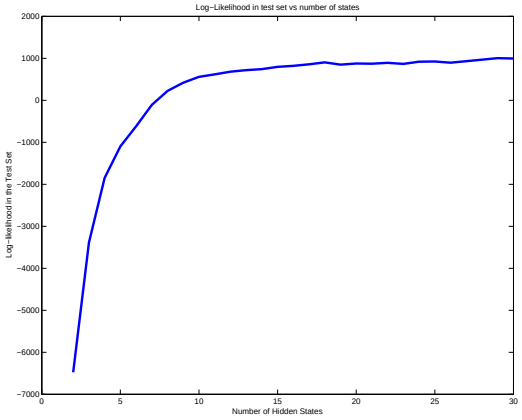


Figure 7.6: Log-likelihood of time series as the number of hidden states in the HMM vary

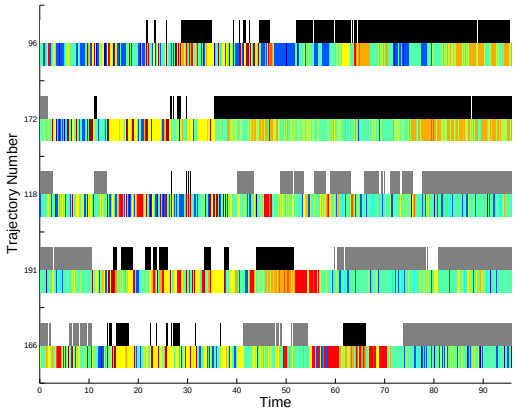


Figure 7.7: Sample time series of flies in the hidden state space with position indication. The color of the hidden states correspond to the colors used in Figure 7.5. The position of the fly is encoded with three colors: white for open space, grey for food patch, and black for the boundary.

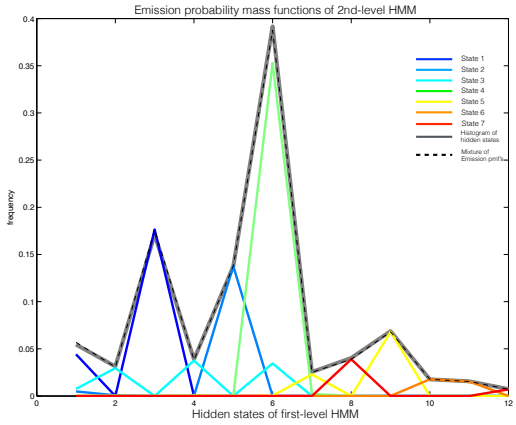


Figure 7.8: Emission probability mass functions of 2nd-level HMM

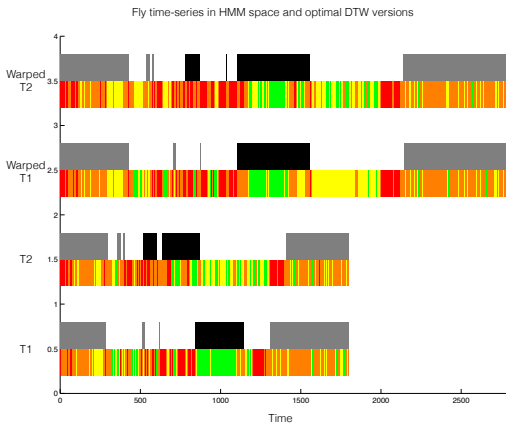


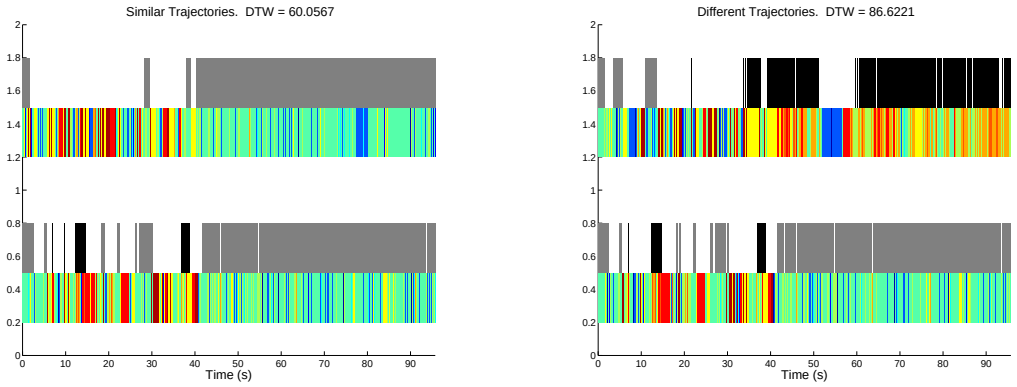
Figure 7.9: Time-series in hidden state space and warped versions found to compute the DTW distance between the pair of time series. Position is overlaid in the figure, but this information is not used in the DTW distance calculation.

mainly visible when the flies are feeding, as they walk slowly and stop continuously while they feed.

We now move on to clustering time series of individuals in order to find stories in this setup. We find the DTW pairwise distances among the individuals we want to analyze. Figure 7.9 shows two time series in a simpler 4-state HMM that are compared for illustration purposes, with the position overlaid, but not taken into account in the DTW distance calculation. The top plots show the warped versions of the time series that yield the minimum cost in a frame to frame comparison of the warped time-series. Once a matrix of distances is computed between each pair of trajectories, we can cluster them using k-medoids. Figure 7.10 shows a pair of similar time series and a pair of dissimilar time-series, when the 12-state HMM is used.

In order to visualize the clusters produced by k-medoids, we use Multidimensional Scaling (MD-scaling) [45]. The technique finds the location of points in a d-dimensional Euclidean space, given a matrix of distances or dissimilarities between points. In our case, the matrix of DTW distances is the input to the MD-scaling algorithm, and we use $d = 2$ in order to visualize the clusters. Figure 7.11 show clusters found in a subset of the data that included 240 time-series of flies on food, coming from setups with 10 flies, and where 10 forward and backward passes of the paddles were presented to the flies. We first examine the clusters by looking at the medoid time-series. Figure 7.11b shows the time series in HMM space of the three medoids of the dataset, with positions overlaid, as well as with vertical lines indicating the time at which the forward and backward passes of the paddle occurred.

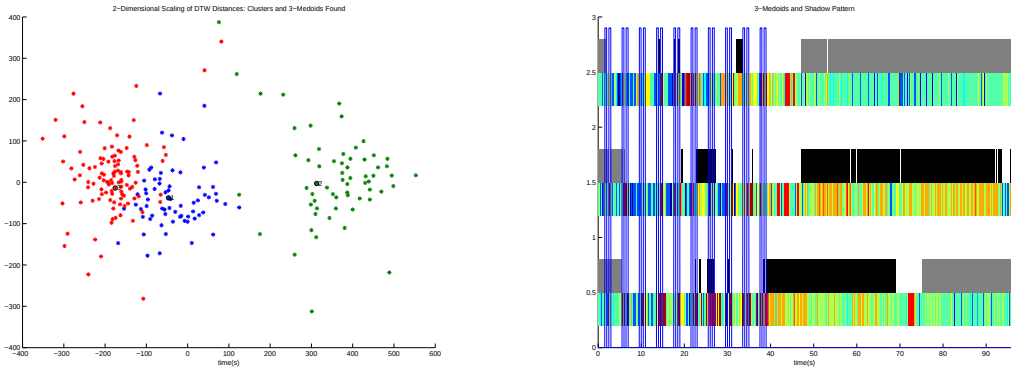
The three medoids can be interpreted as three distinct stories. The first medoid (from bottom to top) indicates a fly walking slowly on food (green state), which gets agitated when shadows pass



(a) Close time-series in DTW sense

(b) Distant time-series in DTW sense

Figure 7.10: Sample close and distant time-series in HMM space according to the DTW distance



(a) 3-medoid clustering in 2-dscaling

(b) Time-series in HMM space of medoids found

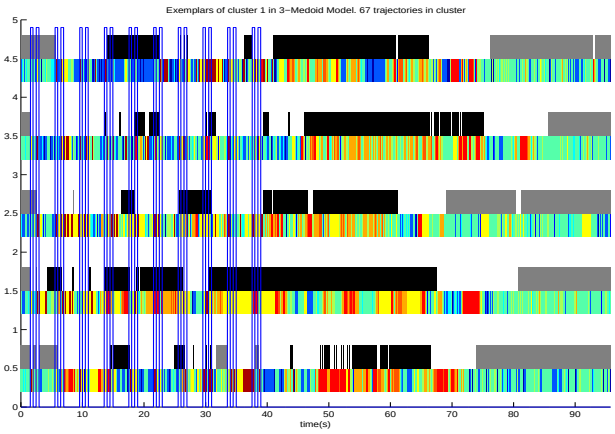
Figure 7.11: Clusters found by k-medoids using DTW distances on a 240-fly subset of the data, visualized using 2-dimensional scaling. The corresponding time-series of the medoids are also shown.

(yellow and red states) and ends up walking quickly (orange states) on the boundary before returning to the food. The second medoid shows a fly that after the shadows pass gets agitated and stays on the boundary walking quickly, as indicated by the predominant orange color of the trajectory. The third story shows instead the fly returning shortly after the shadows pass, walking slowly on food as indicated by the predominant green state. To confirm this observation, we plot five random time-series from each of the clusters. As it is clear visually in Figure 7.12, the medoids are indeed faithful representatives of the time-series that are grouped into the same cluster.

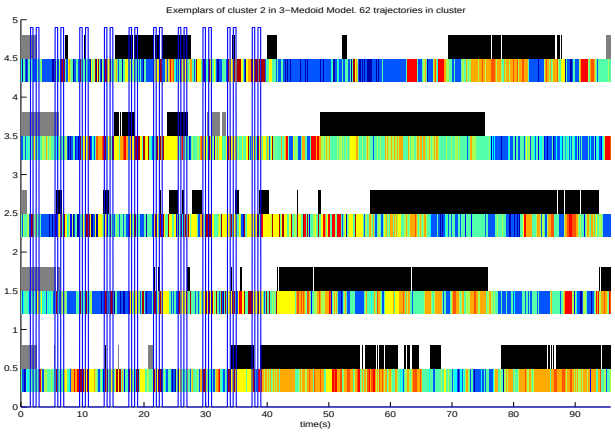
A more detailed analysis of each of the stories can be done by plotting histograms along time in hidden-state space, of each of the stories. Furthermore, as the position of the location of the flies is known, in terms of the three major regions, namely the food patch, open space, and boundary, the histograms can be further separated by location. Figure 7.13 shows these histograms. The histogram for time series in the same story is divided into three, displaying the percentage of flies on food at the top, those in the open space in the middle, and those in the boundary in the bottom. A guide for the mean log-velocity of each state and its standard deviation is plotted for reference at the rightmost part of the plots.

These histograms provide a clear summary of the data, in terms of the velocity of the flies in the experiment. It is clear that before the shadow stimuli are presented, the flies are all walking slowly or stationary on food. The flies then disperse once the shadows are presented, and most flies hop, as it is evidenced by the increase in frequency of the acceleration (yellow) state, and the hopping state (dark red). Another interesting observation is that during the shadow presentations, not only the acceleration and hopping states (yellow and dark red) increase, but it is also the case that the dark blue state increases and decreases periodically with the shadows. This shows evidence of a freezing behavior. As evidenced in other animals such as rodents, when an arousal event such as the presence of a predator occurs, animals freeze before attempting escape. This may be a mechanism to avoid detection, for example. *Yet, before this experiment had been carried out, clear evidence of freezing in flies had not been presented.* CUBA detects evidence of this behavior.

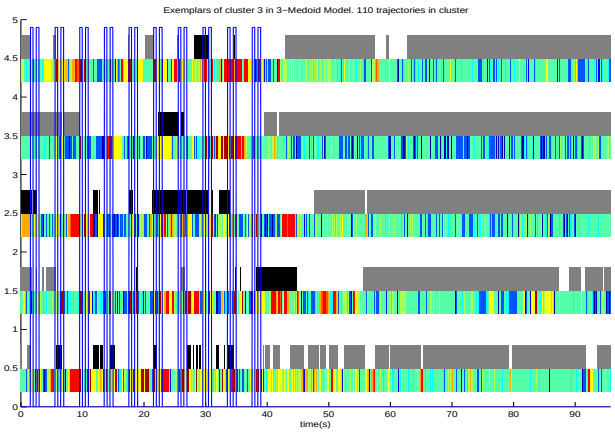
Across all clusters, it is clear that the first two shadows are enough to disperse all flies from the food patch. After the third shadow, it is now common to find flies in the boundary. Three possibilities divide the time series into separate stories. Either flies are very quick to return to food, after settling down in open space, as shown in story 3, and therefore flies will predominantly remain in the slow walking state (green) once they settle down. It may also be the case that flies remain very agitated as shown by the orange states in story 2. Yet, notice that the orange states are mostly predominant when the flies are in the boundary. Therefore this raises an important point, which is that flies walk faster when they are on the boundary. Similarly, when flies are on food, the fast walking states (orange) are



(a) Time-series from cluster 1

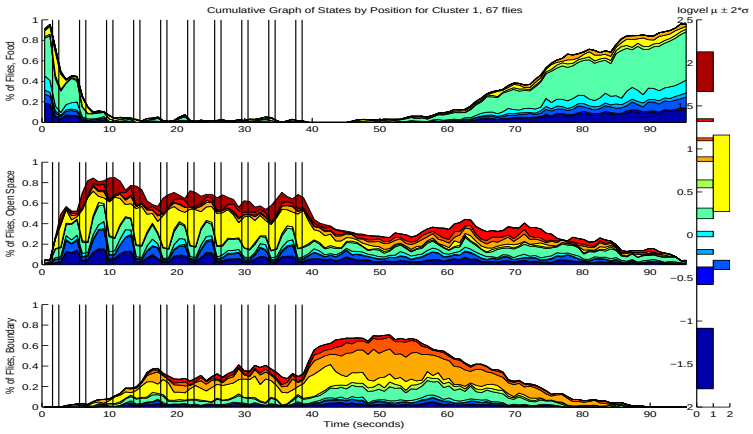


(b) Time-series from cluster 2

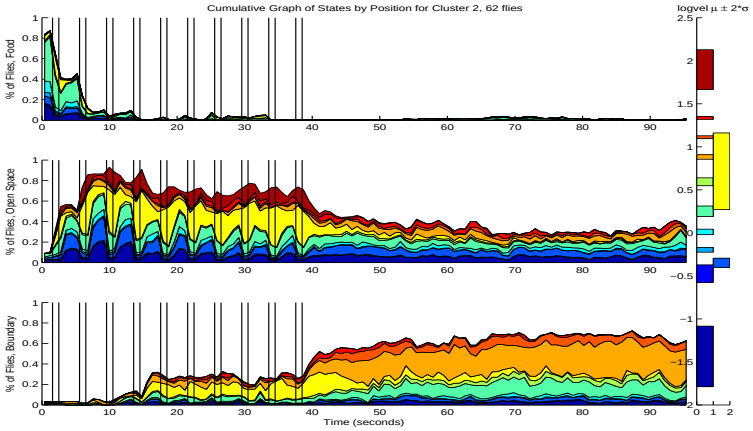


(c) Time-series from cluster 3

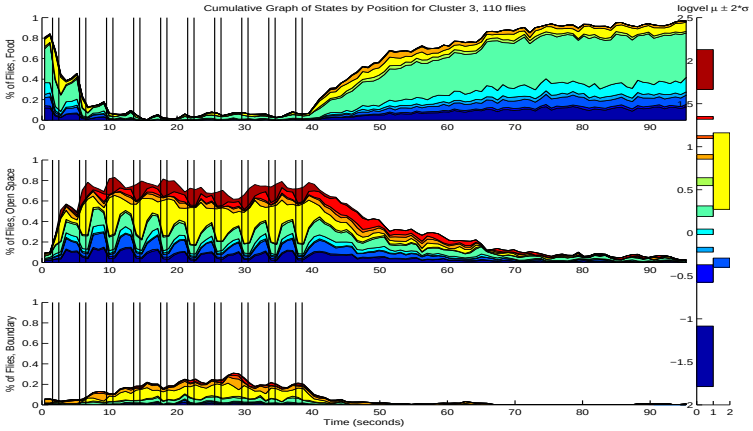
Figure 7.12: Time series in HMM space with overlaid position from the three clusters found



(a) Cluster 1: 67 flies



(b) Cluster 2: 62 flies



(c) Cluster 3: 110 flies

Figure 7.13: Histograms of states occupied by flies by cluster (story) and position (food patch, open space, and boundary)

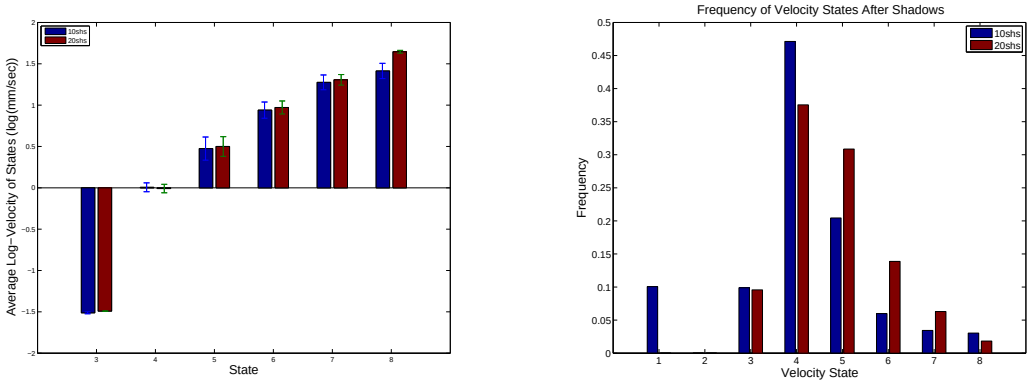
almost not present. This observation could be subject to causal testing, that is, biologists could test if flies walk faster because they are in the boundary, or if it is the case that agitated flies spend more time in the boundary. Finally, there is an intermediate story, in which flies disperse to open space, go to the boundary, but then do return to food during the length of the video.

With this visualization, it is clear how applying CUBA to the dataset allows summarizing and identifying the different behaviors present in the experiment. The detection of movemes as well as automatic classifiers for behaviors like freezing and hopping, present a major advantage as the classifiers are a byproduct of the method that require no manual annotations. Also, clustering trajectories into similar stories allows identifying the different types of behaviors elicited by the stimuli presented, as well as having a concise but detailed understanding of what flies are doing. This methodology, compared to usual methods that compare simple statistical quantities, such as mean or median velocities as done in [35], allows a deeper understanding of the biological phenomena observed. A simple value like the mean can be deceiving in the case of outliers, while the description given by CUBA gives a clear picture of the phenomena observed.

Another application of CUBA to a biological experiment like the Fear in Flies dataset, is that it allows using the mathematical models obtained to test hypotheses. For example, a simple experiment that was carried out in the original investigation was to understand how the number of shadows presented to the flies affected the flies. If more shadows elicited higher levels of agitation this could imply an integration model for "fear" in flies. To illustrate how CUBA can be used to show this, we apply the method comparing two scenarios: groups of 10-flies where 10 shadows are presented, and groups of 10-flies subject to 20 shadows. We now learn two HMM models: one for the 10-shadow data, and another for the 20-shadow experiment. We keep the number of hidden states Q equal for both models.

We ran this experiment in two sets of data: 240 flies subject to 10 shadows, and 240 flies subject to 20 shadows. Figure 7.14 shows the mean of the emission distributions found for both models, with blue bars representing the model for the 10-shadow data and red for the 20-shadow data. The standard deviation of the emission density function means are found by fitting the model multiple times with different splits of the data into training and validation sets. For this subset of the data, the chosen model using cross-validation was $Q = 8$. As it is clear from Figure 7.14a, state number 8, which constitutes the hopping/flying state, has a significantly higher value for the mean in log-velocity space. For the remaining states, there is no significance difference between the mean of the states. This shows clear evidence that the arousal caused by 20 shadows leads to higher observed velocities than in the case of 10-shadow presentations.

Figure 7.14b also shows evidence for the higher arousal in the 20-shadow experiment as it shows



(a) Means of emission densities for the HMM models for the10 vs 20 shadow experiments

(b) Time-series in HMM space of medoids found

Figure 7.14: Comparisson of HMMs learned for the 10 vs 20 shadow experiments

the distribution of flies in each of the hidden states after the shadows are presented. As it is clear from the plot, for the 20-shadow experiment, the flies spend more time in states 5 through 7 than the flies in the 10-shadow experiment, which spend most of their time in the slow walking state (state 4). This again shows evidence of a scalable response by the shadows, as the number of shadows increase.

This simple experiment exemplifies how hypothesis can be tested using the CUBA machinery. We now move on to an application to CUBA to the Fly Bowl dataset, which was created in order to understand interactions of flies in a closed bowl, when the wings have been removed.

7.3.2 CUBA in the Fly Bowl dataset

The Fly Bowl dataset is composed of videos of flies in a planar enclosed arena. There are usually 20 flies in each arena, and the wings of the flies have been removed. The software developed in [20] uses Computer Vision techniques to extract the position of the flies, as well as their orientation. Various techniques are used in order to avoid identity swapping when flies are close to each other. The work presented in that paper detects and analyzes specific behaviors in flies: walk, stop, sharp turn, backup, crab walk, touch, and chase. The high-throughput system developed in that study allowed analyzing various genetic lines. The vectors describing behavior frequency and duration allowed predicting genetic lines as well as gender.

In this context, we use CUBA to analyze the data without previously defining behaviors. For a subset of the data consisting of 415 fly trajectories filmed during 13 minutes, we run the CUBA machinery, once again on the log-velocity time series, and obtain a HMM with 15-states or movemes. We then run a second HMM and choose through cross-validation $Q^{(2)} = 6$ states. The inferred trajectories in HMM space are then clustered, obtaining this time 4 clusters. Figure 7.15 shows the

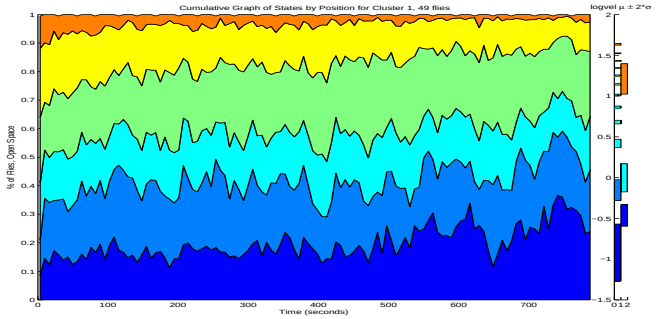
histograms of states in time for each of the clusters. In this case the fly bowl does not allow flies to walk in the border, and since there is no food, there is no spatial differentiation in the flies.

The actions represented by the 6 hidden states of the second-level HMM are the following: state 1 and state 2 (dark blue states) represent stationary states; state 3 (cyan) represents a slow walk–stop–slow walk repetition; state 4 (green state) represents a continuous slow walking state; state 5 (yellow) is a fast walking state; and state 6 (orange) involves accelerating–fast walking–decelerating repetition. The histograms of Figure 7.15 also show plots that indicate which states of the first level 15-state HMM are included in each of the second-level hidden states, with their corresponding means and standard deviation.

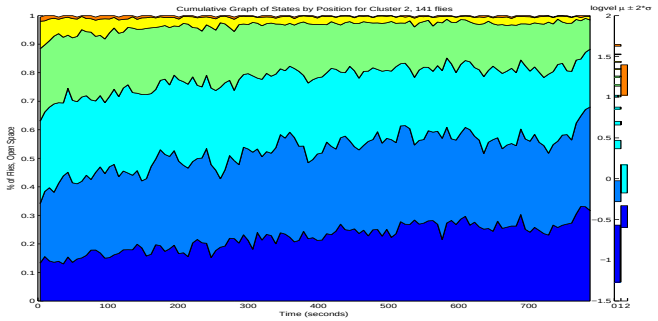
The four clusters indicate roughly the following stories: cluster 1 indicates flies that are transitioning between states and hence are changing velocities throughout the movie, although they tend to become more stationary towards the end; cluster 2 on the other hand indicates flies that are mostly stationary or slow walking but the frequency of states remains mostly constant throughout the movie, indicating mostly constant velocities of the flies; cluster 3 on the other hand indicates flies that are agitated throughout the movie, indicated by the high percentage of state 6 (orange), as well as constant changes in the frequencies of states indicating transition between states; finally cluster 4 is similar to cluster 2, except that the flies do not tend to stop by the end of the movie, but rather maintain a similar speed throughout the movie.

In this case, due to the lack of external stimuli, characterizing behavior in the form of stories is less meaningful as behavior is usually the result of a response to some external or internal stimuli. Nevertheless, the stories obtained become informative if we analyze the percentage of flies in each video that fall in each of the clusters. Figure 7.16 shows the number of flies in each of the clusters, for each of the videos analyzed. Notice that for some videos less than 20 flies are shown. This is the case as some preparations had less flies or because the tracker could not detect a reliable trajectory for some of the flies.

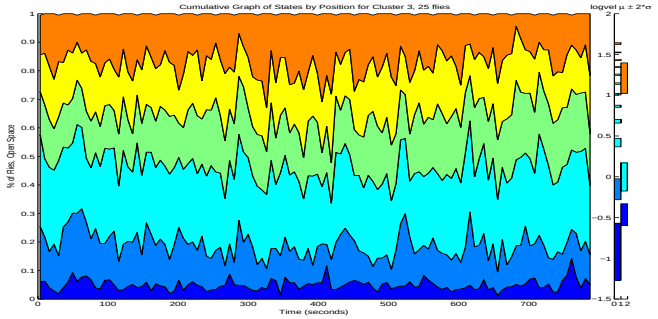
It may seem clear to the reader that the distribution among clusters is very different across the different videos in the dataset. In fact, when we discussed this difference with biologists, they immediately identified that the subset of videos used had different genetic lines of flies. Videos 1 and 2 correspond to one genetic line, videos 3 to 12 correspond to the control genetic line of wild type flies, videos 13 to 15 correspond to another genetic line, and the final six videos correspond to a different line. Clearly, the first two videos corresponding to one genetic line, account for most of the flies that fall into cluster 3, the story of highly agitated flies. The videos in the third group account for most of the flies falling in cluster 1. Finally the remaining videos have a similar distribution of flies in clusters 2 and 4, clusters which are very similar and indicate flies moving at a nearly constant and slow



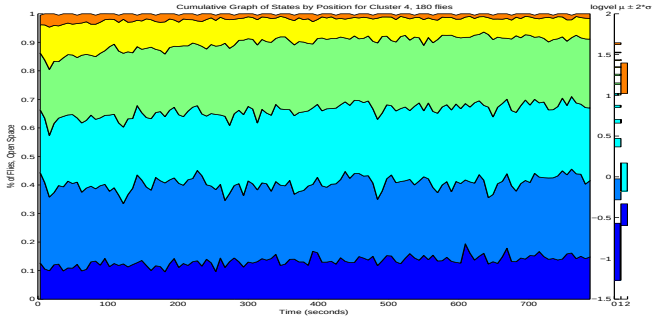
(a) Cluster 1: 49 flies



(b) Cluster 2: 141 flies



(c) Cluster 3: 25 flies



(d) Cluster 4: 180 flies

Figure 7.15: Histograms of states occupied by flies by cluster for a subset of the Fly Bowl dataset

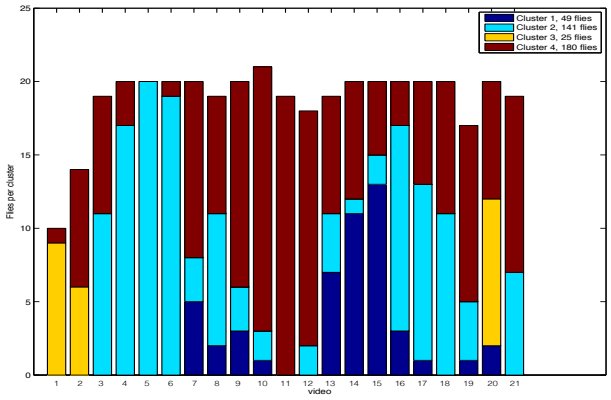


Figure 7.16: Flies per cluster per video in a subset of the Fly Bowl dataset

speed throughout the movie. That is, wild type flies under this setup tend to be "calm and steady" throughout the video, while the other genetic lines are prone to agitation and constant changes in speed. Hence, applying CUBA to this dataset in fact allows us to discriminate between genetic lines, just as the study that used predefined behaviors as descriptors in order to predict gender and genetic lines did.

In fact, CUBA can be modified to be a semi-supervised method, as once DTW distances are computed, we can use MD-scaling in this case not for visualization purposes but to embed the time-series in a high-dimensional Euclidean space. In this space, supervised machine learning methods can be used to identify various groups of flies, such as different genetic lines or genders, if this information is available. In our experiment, this was done using completely unsupervised methods as we were initially unaware the videos contained different genetic lines, and yet the method was able to discriminate these. Hence, applying CUBA to a different dataset illustrates a different use that can be given to the method developed.

The unsupervised method developed in this chapter is able to detect behavioral patterns at different time scales. We can detect movemes by means of a HMM and its hidden states, a technique that allows building classifiers for specific behaviors without the need for annotation. We can detect actions by further grouping movemes using a new HMM whose observed variables are the hidden states of the initial HMM. Furthermore, we can find distances between the observed time-series in this new hidden-state space, by using an appropriate measure as DTW, which allows time deformations between series when finding the comparison. This allows clustering in meaningful groups the time-series, yielding typical stories found in the data. The machinery then allows testing hypotheses, by comparing the models obtained, and allows understanding the data at a much more profound level, compared to the

case where simple statistics like the mean and median are used to summarize the data. Finally, the cluster membership distribution for groups of flies can be discriminative enough to detect different genetic lines, as exemplified in the experiments on the Fly Bowl data. These are just some of the advantages of using a system like CUBA. As biologists begin using this method to analyze their data, it will become clear how much more can be learned on how these computational methods can help advance the study of behavior.

Conclusion

In the first part of the thesis, we explored three fundamental questions that arise naturally when we conceive a machine learning scenario where the training and test distributions can differ. Contrary to conventional wisdom, we show in Chapter 2 that in fact mismatched training and test distribution can yield better out-of-sample performance. Thus, the natural step to follow is to find the optimal training distribution for a given test distribution, regardless of the target function that we want to learn. We called this distribution the dual distribution and showed how to obtain it in both discrete and continuous input spaces, as well as how to approximate it in a practical scenario in Chapter 3. Experiments on both synthetic and real data sets showed the benefits of using the dual distribution. Yet, in order to apply the dual distribution in the supervised learning scenario where the training data set is fixed, it was necessary to use weights to make the sample appear as if it came from the dual distribution. We then explored the negative effect that weighting a sample can have. The theoretical decomposition of the use of weights regarding its effect on the out-of-sample error is easy to understand but not actionable in practice, as the quantities involved cannot be computed. Hence, we proposed the Targeted Weighting algorithm in Chapter 4, that determines if, for a given set of weights, the out-of-sample performance will improve or not in a practical setting. This is necessary as the setting assumes there are no labeled points distributed according to the test distribution, only unlabeled samples. Experiments on real datasets showed the unanimous success of our proposed algorithm. Finally, we proposed a new class of matching algorithms in Chapter 5 that can be used to match the training set to a desired distribution, such as the dual distribution (or the test distribution). These algorithms can be applied to very large datasets, and we showed how they led to improved performance in a large real dataset such as the Netflix dataset. Their computational complexity is the main reason for their advantage over previous algorithms proposed in the covariate shift literature.

In the second part of the thesis, we apply Machine Learning to the problem of behavior recognition in biological experiments. We developed a wing extension classifier needed to analyze thousands of legacy videos efficiently. These videos were part of an ongoing study of fly aggression. The classifier aided in the investigation that culminated in discovering the neuron and substance that is responsible for fly male aggression. We then moved on to the more complex problem of analyzing behavior using

minimal supervision for humans. To do this, we developed CUBA, which allows detecting movemes, actions, and stories from time series describing the position of animals in videos. The method allows summarizing the data, as well as it provides biologists a mathematical tool to test new hypotheses. When applied to real data, the system also allowed finding classifiers for behaviors such as hopping and freezing in flies without the need for annotation, and it also allowed discriminating groups of flies according to their genetic line.

Appendix A

An analytic learning setup

In this appendix, we introduce an analytic setup for the learning problem that is both tractable and versatile. The setup uses a squared loss function and a linear learning model with non-linear transformations. Highly sophisticated target functions and learning models, as well as noise, can be handled under this setup, and we are able to get analytic solutions for the out-of-sample error in this framework. Various chapters of the thesis use the results of this section.

We begin by defining the notation. Let $R = \{x_i, y_i\}_{i=1}^N$ be the training set, with $x_i \in \mathcal{X}$, and $y_i \in \mathcal{Y}$. Assume x_i are iid $\sim P_R$, where P_R is the training distribution. Let the target function be $f : \mathcal{X} \to \mathcal{Y}$, and let ϵ be the stochastic noise process, where ϵ_i is the realization corresponding for x_i, so that $y_i = f(x_i) + \epsilon_i$. Let $\mathcal{H}$ be the hypothesis set used by the learning algorithm, where each $h \in \mathcal{H}$ is $h : \mathcal{X} \to \mathcal{Y}$. Finally we assume the learning algorithm returns a final hypothesis $g \in \mathcal{H}$ that minimizes the squared loss function $\ell_2 : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, $\ell_2(h(x_i), y_i) = (h(x_i) - y_i)^2$. That is,

$$g = \arg\min_h \sum_{i=1}^{N} (y_i - h(x_i))^2. \tag{A.1}$$

We let $\mathcal{H}$ be the set of linear functions in some transformed space $\mathcal{Z}_m$ of the input space $\mathcal{X}$, so that

$$h(x; \theta) = \theta^T \phi_M(x). \tag{A.2}$$

where $\theta, \phi_M(x) \in \mathcal{Z}_M$. For simplicity we let

$$z_M = \phi_M(x) \tag{A.3}$$

In these terms, the goal of the learning algorithm is to find θ^*, where $g(x, R, f, \epsilon) = h(x, \theta^*)$.

We further characterize the target functions by expressing them in terms of non-linear transfor-

mations, so that $f(x) = \tilde{\theta}^T \phi(x)$, where

$$\phi(x) = [\phi_M(x) \quad \phi_C(x)]^T \tag{A.4}$$

and $\phi_C(x) \in \mathcal{Z}_C$ represents the features of the target function that cannot be captured by the model. By letting the dimension of $\mathcal{Z}_C$ grow as desired, and allowing arbitrary non-linear transformation, f can be as complex as we want. Following the same notation as before, we have $z = [z_M^T \quad z_C^T]^T$, and $\theta = [\theta_M^T \quad \theta_C^T]^T$. Figure 2.5 shows sample target functions that can be generated if we use Fourier harmonics as the features of the non-linear transformation, with $\phi(x) \in \mathbb{R}^{10}$, and $\mathcal{X} = [-1, 1]$. As it is clear from the figure, there is great variety that can be achieved with this model.

We are interested in finding an expression for the out-of-sample error E_{out} in this framework. We begin by finding E_{out} at a point $x \in \mathcal{X}$. This error is also a function of R, f, ϵ, and we denote it by $E_{\text{out}}(x, R, f, \epsilon)$. Since both the stochastic noise ϵ and the complexity of the target function (also known as the deterministic noise) vary from problem to problem, we take the expected value with respect to these quantities. To do this, we make the usual assumption about ϵ, which is that it has zero mean ($\mathbb{E}[\epsilon] = 0$), and diagonal covariance matrix, $\mathbb{E}[\epsilon\epsilon^T] = \sigma_N^2 I$, where I is the identity matrix and σ_N is the standard deviation of the stochastic noise. For the target functions, we make the simplifying assumption that the coefficients of the features outside the model, namely $\theta_C \in \mathcal{Z}_C$, have covariance matrix $\mathbb{E}_[\theta_C \theta_C^T] = \sigma_C^2 I$. Then, the expected out-of-sample error is given by

$$\mathbb{E}_{f,\epsilon}[E_{\text{out}}(x, R, f, \epsilon)] = \mathbb{E}_{f,\epsilon}[(f(x) - g(x, R, f, \epsilon))^2]. \tag{A.5}$$

The final hypothesis $g(x, R, f, \epsilon)$ is obtained by minimizing the squared loss function on the training set, and the solution is given by the pseudo-inverse of the data matrix. To be more precise, let

$$Z = \begin{bmatrix} -z_1^T- \\ -z_2 T- \\ \vdots \\ -z_N^T- \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{A.6}$$

where Z is the data matrix. Now

$$Z = [Z_M \quad Z_C]. \tag{A.7}$$

In matrix form, the learning problem reduces to the following quadratic program,

$$\min_{\theta} ||y - Z\theta||^2 \tag{A.8}$$

with analytic solution $\theta^\star$ found through basic calculus:

$$\theta^\star \;=\; (Z^TZ)^{-1}Z^Ty \;=\; (Z^TZ)^{-1}Z^T(Z\theta+\epsilon) \;=\; \theta + Z^\dagger\epsilon, \tag{A.9}$$

where $Z^\dagger = (Z^TZ)^{-1}Z^T$ is the so-called pseudo-inverse of matrix Z. However, as stated before, we assume the target function is more complex than the learning model. Hence, the matrix Z cannot be constructed as ϕ_C is unknown. Instead, we use the matrix Z_M. In this case the learning algorithm will output the parameter vector $\hat{\theta} \in \mathcal{Z}_M$ given by

$$\hat{\theta} \;=\; Z_M^\dagger y \;=\; Z_M^\dagger Z^T\theta \;=\; Z_M^\dagger(Z_M\theta_M + Z_C\theta_C + \epsilon) \tag{A.10}$$

Hence,

$$g(x,R,f,\epsilon) = z^TZ_M^\dagger(Z_M\theta_M + Z_C\theta_C + \epsilon) \tag{A.11}$$

Substituting, the expected out-of-sample error is given by

$$\begin{aligned}
&\mathbb{E}_{f,\epsilon}\left[E_{\text{out}}(x,R,f\epsilon)\right] \\
&= \mathbb{E}_{f,\epsilon}\left[\|z^T\theta + \epsilon_0 - z_M^T(Z_M^\dagger(Z_M\theta_M + Z_C\theta_C+\epsilon)\|^2\right] \\
&= \mathbb{E}_{\epsilon,f}\left[\|z_C^T\theta_C + \epsilon_0 - z_M^TZ_M^\dagger(Z_C\theta_C+\epsilon)\|^2\right] \\
&= \mathbb{E}_f\left[(z_C^T - z_M^TZ_M^\dagger Z_C)\theta_C\theta_C^T(z_C^T - z_M^TZ_M^\dagger Z_C)^T\right] + \mathbb{E}_\epsilon\left[\epsilon_0^2 + z_M^TZ_M^\dagger\epsilon\epsilon^T(Z_M^\dagger)^Tz_M\right] \\
&= \mathbb{E}_f\left[Tr\left((z_C^T - z_M^TZ_M^\dagger Z_C)\theta_C\theta_C^T(z_C^T - z_M^TZ_M^\dagger Z_C)^T\right)\right] + \mathbb{E}_\epsilon\left[Tr\left(z_M^TZ_M^\dagger\epsilon\epsilon^T(Z_M^\dagger)^Tz_M\right)\right] + \sigma_N^2 \\
&= \mathbb{E}_f\left[Tr\left(\theta_C\theta_C^T(z_C^T - z_M^TZ_M^\dagger Z_C)^T(z_C^T - z_M^TZ_M^\dagger Z_C)\right)\right] + \mathbb{E}_\epsilon\left[Tr(\epsilon\epsilon^T(Z_M^\dagger)^Tz_Mz_M^TZ_M^\dagger)\right] + \sigma_N^2 \\
&= Tr\left(\mathbb{E}_f\left[\theta_C^T\theta_C\right](z_C^T - z_M^TZ_M^\dagger Z_C)^T(z_C^T - z_M^TZ_M^\dagger Z_C)\right) + Tr\left(\mathbb{E}_\epsilon\left[\epsilon\epsilon^T\right](Z_M^\dagger)^Tz_Mz_M^TZ_M^\dagger\right)] + \sigma_N^2 \\
&= Tr\left(\sigma_C^2(z_C^T - z_M^TZ_M^\dagger Z_C)^T(z_C^T - z_M^TZ_M^\dagger Z_C)\right) + Tr\left(\sigma_N^T(Z_M^\dagger)^Tz_Mz_M^TZ_M^\dagger\right) + \sigma_N^2 \\
&= \sigma_C^2(z_C^T - z_M^TZ_M^\dagger Z_C)^T(z_C^T - z_M^TZ_M^\dagger Z_C) + \sigma_N^2Tr\left(z_M^TZ_M^\dagger(Z_M^\dagger)^Tz_M\right) \\
&= \sigma_C^2\|z_C^T - z_M^TZ_M^\dagger Z_C\|^2 + \sigma_N^2Tr\left(z_M^T(Z_M^TZ_M)^{-1}Z_M^TZ_M(Z_M^TZ_M)^{-1}z_M\right) \\
&= \sigma_C^2\|z_C^T - z_M^TZ_M^\dagger Z_C\|^2 + \sigma_N^2z_M^T(Z_M^TZ_M)^{-1}z_M + \sigma_N^2,
\end{aligned} \tag{A.12}$$

where ϵ_0 denotes the stochastic noise at the point x, and $Tr(A)$ denotes the trace of matrix A. The above derivation reorganizes the expression using the fact that the trace of a scalar is the scalar itself, and the fact that $Tr(AB) = Tr(BA)$. Finally, we use the assumptions on the stochastic and deterministic noise to find the expected values.

Bibliography

[1] J Aach and GM Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17(6):495–508, 2001.

[2] YS Abu-Mostafa, M Magdon-Ismail, and HT Lin. *Learning from data*. AMLBook, 2012.

[3] G Al-Naymat, S Chawla, and J Taheri. Sparsedtw: a novel approach to speed up dynamic time warping. In *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*, pages 117–127. Australian Computer Society, Inc., 2009.

[4] DJ Anderson and R Adolphs. A framework for studying emotions across species. *Cell*, 157(1):187–200, 2014.

[5] DJ Anderson and P Perona. Toward a science of computational ethology. *Neuron*, 84(1):18–31, 2014.

[6] K Asahina, K Watanabe, BJ Duistermars, E Hoopfer, CR González, EA Eyjólfsdóttir, P Perona, and DJ Anderson. Tachykinin-expressing neurons control male-specific aggressive arousal in drosophila. *Cell*, 156(1):221–235, 2014.

[7] K Bache and M Lichman. UCI machine learning repository, 2013.

[8] Z Bar-Joseph, G Gerber, DK Gifford, TS Jaakkola, and I Simon. A new approach to analyzing gene expression time series data. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 39–48. ACM, 2002.

[9] C Bargmann, W Newsome, A Anderson, E Brown, K Deisseroth, J Donoghue, P MacLeish, E Marder, R Normann, J Sanes, et al. Brain 2025: a scientific vision. *Brain Research through Advancing Innovative Neurotechnologies (BRAIN) Working Group Report to the Advisory Committee to the Director, NIH http://www. nih. gov/science/brain/2025/(US National Institutes of Health, 2014)*, 2014.

[10] S Ben-David, J Blitzer, K Crammer, A Kulesza, F Pereira, and JW Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[11] S Ben-David, J Blitzer, K Crammer, and F Pereira. Analysis of representations for domain adaptation. *Advances in Neural Information Processing Systems*, 19:137, 2007.

[12] DJ Berndt and J Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[13] S Bickel, M Brückner, and T Scheffer. Discriminative learning for differing training and test distributions. In *Proceedings of the 24th International Conference on Machine Learning*, pages 81–88. ACM, 2007.

[14] S Bickel, M Brückner, and T Scheffer. Discriminative learning under covariate shift. *The Journal of Machine Learning Research*, 10:2137–2155, 2009.

[15] J Blitzer, M Dredze, and F Pereira. Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 440, 2007.

[16] J Blitzer, R McDonald, and F Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128. Association for Computational Linguistics, 2006.

[17] A Blum. Learning form labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.

[18] A Blum and T Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 92–100. ACM, 1998.

[19] SP Boyd and L Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] K Branson, AA Robie, J Bender, P Perona, and MH Dickinson. High-throughput ethomics in large groups of drosophila. *Nature Methods*, 6(6):451–457, 2009.

[21] XP Burgos-Artizzu, P Dollár, D Lin, DJ Anderson, and P Perona. Social behavior recognition in continuous video. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1322–1329. IEEE, 2012.

[22] EG Caiani, A Porta, G Baselli, M Turiel, S Muzzupappa, F Pieruzzi, C Crema, A Malliani, and S Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *Computers in Cardiology 1998*, pages 73–76. IEEE, 1998.

[23] CC Chang and CJ Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[24] T Chen, Z Zheng, Q Lu, W Zhang, and Y Yu. Feature-based matrix factorization. *arXiv preprint arXiv:1109.2271*, 2011.

[25] C Cortes, Y Mansour, and M Mohri. Learning bounds for importance weighting. *Advances in Neural Information Processing Systems*, 23:442–450, 2010.

[26] C Cortes and M Mohri. Domain adaptation and sample bias correction theory and algorithm for regression. *Theoretical Computer Science*, 519:103–126, 2014.

[27] C Cortes, M Mohri, M Riley, and A Rostamizadeh. Sample selection bias correction theory. In *Algorithmic Learning Theory*, pages 38–53. Springer, 2008.

[28] R Courant and D Hilbert. *Methods of mathematical physics*, volume 1. CUP Archive, 1966.

[29] I Csisz et al. Information-type measures of difference of probability distributions and indirect observations. *Studia Sci. Math. Hungar.*, 2:299–318, 1967.

[30] B Dacorogna. *Introduction to the Calculus of Variations*, volume 4. World Scientific, 2004.

[31] H Dankert, L Wang, ED Hoopfer, DJ Anderson, and P Perona. Automated monitoring and analysis of social behavior in drosophila. *Nature Methods*, 6(4):297–303, 2009.

[32] V Digalakis, D Rtischev, and L Neumeyer. Speaker adaptation using constrained estimation of Gaussian mixtures. *Speech and Audio Processing, IEEE Transactions on*, 3(5):357–366, 1995.

[33] E Eyjolfsdottir, S Branson, XP Burgos-Artizzu, ED Hoopfer, J Schor, DJ Anderson, and P Perona. Detecting social actions of fruit flies. In *Computer Vision–ECCV 2014*, pages 772–787. Springer, 2014.

[34] J Gao, W Fan, J Jiang, and J Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 283–291. ACM, 2008.

[35] WT Gibson, CR. Gonzalez, CM Fernandez, L Ramasamy, T Tabachnik, RR Du, PD Felsen, MM Maire, P Perona, and DJ Anderson. Behavioral response to a repetitive visual threat stimulus express a persistent state of defensive arousal in Drosophila. *Current Biology*, May 2015.

[36] CR González and YS Abu-Mostafa. Mismatched training and test distributions can outperform matched ones. *Neural Computation*, 27(2):365–387, February 2015.

[37] T Groves and T Rothenberg. A note on the expected value of an inverse matrix. *Biometrika*, 56(3):690–691, 1969.

[38] J Huang, AJ Smola, A Gretton, K Borgwardt, and B Scholkopf. Correcting sample selection bias by unlabeled data. *Advances in Neural Information Processing Systems*, 19:601, 2007.

[39] H Jhuang, E Garrote, X Yu, V Khilnani, T Poggio, AD Steele, and T Serre. Automated home-cage behavioural phenotyping of mice. *Nature Communications*, 1:68, 2010.

[40] J Jiang and C Zhai. Instance weighting for domain adaptation in nlp. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 264, 2007.

[41] M Kabra, AA Robie, M Rivera-Alba, S Branson, and K Branson. Jaaba: interactive machine learning for automatic annotation of animal behavior. *Nature Methods*, 10(1):64–67, 2013.

[42] T Kanamori, S Hido, and M Sugiyama. A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research*, 10:1391–1445, 2009.

[43] T Kanamori and H Shimodaira. Active learning algorithm using the maximum weighted log-likelihood estimator. *Journal of Statistical Planning and Inference*, 116(1):149–162, 2003.

[44] Y Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434. ACM, 2008.

[45] JB Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[46] C Leggetter and P Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech and Language*, 9(2):171, 1995.

[47] Y Mansour, M Mohri, and A Rostamizadeh. Domain adaptation: Learning bounds and algorithms. *arXiv preprint arXiv:0902.3430*, 2009.

[48] A Margolis. A literature review of domain adaptation with unlabeled data. *University of Washington, http://ssli. ee. washington. edu/˜ amargoli/review_Mar23. pdf*, 2011.

[49] L Mason, J Baxter, PL Bartlett, and MR Frean. Boosting algorithms as gradient descent. In S.A. Solla, T.K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 512–518. MIT Press, 2000.

[50] K Murphy and M Dunham. Probabilistic model toolkit for MATLAB and Octave (pmtk3).

[51] A Ng, M Jordan, Y Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002.

[52] S Pan, J Kwok, and Q Yang. Transfer learning via dimensionality reduction. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 2, pages 677–682, 2008.

[53] E Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, pages 1065–1076, 1962.

[54] J Quiñonero-Candela, M Sugiyama, A Schwaighofer, and ND Lawrence. *Dataset shift in machine learning*. Cambridge, MA: MIT Press, 2009.

[55] J Ren, X Shi, W Fan, and P Yu. Type-independent correction of sample selection bias via structural discovery and re-balancing. In *Proceedings of the Eighth SIAM International Conference on Data Mining, SDM*, pages 565–576, 2008.

[56] M Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[57] S Rosset, J Zhu, H Zou, and T Hastie. A method for inferring label sampling mechanisms in semi-supervised learning. *Ann Arbor*, 1001:48109, 2005.

[58] S Sabato and R Munos. Active regression by stratification. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 469–477. Curran Associates, Inc., 2014.

[59] S Salvador and P Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. *KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, 2004.

[60] J Sherman and WJ Morrison. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix. *The Annals of Mathematical Statistics*, 20(4):620–624, 1949.

[61] H Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000.

[62] Y Sogawa, T Ueno, Y Kawahara, and T Washio. Active learning for noisy oracle via density power divergence. *Neural Networks*, 46:133–143, 2013.

[63] M Sugiyama. Active learning in approximately linear regression based on conditional expectation of generalization error. *The Journal of Machine Learning Research*, 7:141–166, 2006.

[64] M Sugiyama and M Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT Press, 2012.

[65] M Sugiyama, M Krauledat, and K Müller. Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research*, 8:985–1005, 2007.

[66] M Sugiyama and S Nakajima. Pool-based active learning in approximate linear regression. *Machine Learning*, 75(3):249–274, 2009.

[67] M Sugiyama, S Nakajima, H Kashima, P Von Buenau, and M Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. *Advances in Neural Information Processing Systems*, 20:1433–1440, 2008.

[68] L Torgo. LIACC regression datasets.

[69] JT Vogelstein, Y Park, T Ohyama, Rex A Kerr, JW Truman, CE Priebe, and M Zlatic. Discovery of brainwide neural-behavioral maps via multiscale unsupervised structure learning. *Science*, 344(6182):386–392, 2014.

[70] DP Wiens. Robust weights and designs for biased regression models: Least squares and generalized m-estimation. *Journal of Statistical Planning and Inference*, 83(2):395–412, 2000.

[71] B Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the 21st International Conference on Machine Learning*, page 114. ACM, 2004.

[72] C Zhang, L Zhang, and J Ye. Generalization bounds for domain adaptation. *Advances in Neural Information Processing Systems*, 2012.