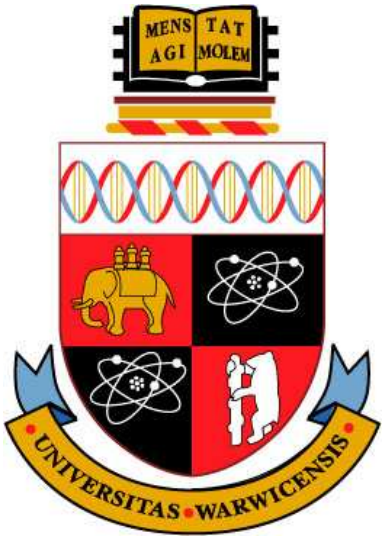


Pattern Classification via Unsupervised Learners

by

Nicholas James Palmer

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

The Department of Computer Science

March 2008

THE UNIVERSITY OF
WARWICK

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank Dr. Paul Goldberg for introducing me to the topic of machine learning and for his supervision, friendship and support throughout the duration of my PhD.

I would also like to thank Prof. Mike Paterson and Prof. Roland Wilson for their help and advice throughout my time as a postgraduate.

Finally I thank the EPSRC for grant GR/R86188/01 which helped fund this research.

Declarations

This thesis contains published work and work which has been co-authored. [38] and [39] were co-authored with Dr. Paul Goldberg of the University of Liverpool. [39] was published in the Proceedings of ALT 05 and a revised version has since been published in "Special Issue of Theoretical Computer Science on ALT 2005" [40]. [38] is Technical Report 411 of the Department of Computer Science at the University of Warwick, and has not been published but is available on arXiv. Other than the contents stated below, the rest of the thesis is the author's own work.

Material from [38] is included in Chapter 2. Goldberg made the suggestion of the technique to smooth distributions in Section 2.2.4 and constructed the proof of Lemma 22. Section 5.7 is also taken from this paper and was written by the author.

Material from [40] is included in Chapter 5. Goldberg contributed Section 5.5.1 based on joint discussions, the basis of the proof in Section 5.5.2 (which has since been revised) and the idea behind Section 5.5.3.

Abstract

We consider classification problems in a variant of the Probably Approximately Correct (PAC)-learning framework, in which an unsupervised learner creates a discriminant function over each class and observations are labeled by the learner returning the highest value associated with that observation. Consideration is given to whether this approach gains significant advantage over traditional discriminant techniques.

It is shown that PAC-learning distributions over class labels under L_1 distance or KL-divergence implies PAC classification in this framework. We give bounds on the regret associated with the resulting classifier, taking into account the possibility of variable misclassification penalties. We demonstrate the advantage of estimating the a posteriori probability distributions over class labels in the setting of Optical Character Recognition.

We show that unsupervised learners can be used to learn a class of probabilistic concepts (stochastic rules denoting the probability that an observation has a positive label in a 2-class setting). This demonstrates a situation where unsupervised learners can be used even when it is hard to learn distributions over class labels – in this case the discriminant functions do not estimate the class probability densities.

We use a standard state-merging technique to PAC-learn a class of probabilistic automata and show that by learning the distribution over outputs under the weaker L_1 distance rather than KL-divergence we are able to learn without knowledge of the expected length of an output. It is also shown that for a restricted class of these automata learning under L_1 distance is equivalent to learning under KL-divergence.

Abbreviations

The following general abbreviations and terminology are found throughout the thesis:

$\alpha(x, f(x))$ – The expected cost associated with classifier f for an observation of x.

δ – The confidence parameter commonly used in learning frameworks.

ϵ – The accuracy parameter commonly used in learning frameworks.

D_ℓ – Distribution D restricted to observations with label ℓ.

DFA – Deterministic finite-state automata.

f^* – The Bayes optimal classifier.

g_ℓ – The class prior of label ℓ (or a priori probability of ℓ).

HMM – Hidden Markov model.

$I(D||D')$ – Kullback-Leibler divergence.

KL-divergence – Kullback-Leibler divergence, $I(D||D')$.

L_1 distance – The variation distance (also rectilinear distance).

L_2 distance – The Euclidean distance.

OCR – Optical character recognition.

p-concept – Probabilistic concept, $c : X \to [0, 1]$.

PAC – Probably approximately correct.

PDFA – Probabilistic deterministic finite-state automata.

PFA – Probabilistic finite-state automata.

PNFA – Probabilistic nondeterministic finite-state automata.

POMDP – Partially observable Markov decision process.

$R(f)$ – The risk associated with classifier f.

Chapter 1

Introduction

The area of research classed as machine learning is a subset of the more general topic of artificial intelligence. Definitions of artificial intelligence vary between texts[1] but it is widely accepted that artificially intelligent systems exhibit one or more of a number of qualities such as the ability to learn, to respond to stimuli, to demonstrate cognition and to act in a rational fashion. This usually involves the design of intelligent agents, which have the ability to perceive their environment and act accordingly to stimuli. In relation to learning theory this behaviour manifests itself as the ability to respond to input observations of the state of the environment. In the context of this work, the environment is usually an arbitrary domain X which can be discrete or continuous depending on the problem setting. The response of the agent can generally be categorised as one of two things – a classification of the observed data, or an estimate of the source generating the observations. The ability to make these responses comes as a consequence of learning from previously-seen observations.

In the context of this thesis we will generally be concerned with solving classification problems. Classification problems involve selecting a label from a predefined set of class labels and associating one with an observation. The form of the observation depends on the setting of the problem, but in general the term observation can relate to any number of measurements or recorded values. For example, in the context of predicting a weather forecast for tomorrow, "an observation" may consist of a measurement of the temperature, wind direction, cloud cover and movement of local weather fronts (among many others). In order to make a classification, some mechanism must be in place for the agent to "learn" how observations should be classified. This can come in the form of feedback on its performance given by either a trainer or from the environment or — as is the case in this thesis — the agent is provided with a sample of data and tasked with identifying patterns in the data from which to draw comparisons

[1] See [43] for a summary of definitions.

with future observations. This form of classification problem is in contrast to the related topic of regression, where rather than learning to link observations with class labels, the aim is to find a correlation between observed values and a *dependent variable*. The resulting regression curve can be used to estimate the value of the dependent variable associated with new observations. Note that regression maps the data observations to a continuous real valued scale rather than the finite set of class labels used in classification problems.

In some settings it may be necessary to model the observed data rather than classifying observations. In this case the learner will examine a set of data and then output some sort of model in an attempt to approximate the way in which the data is being generated. In order to process complex data structures it is often useful to define such theoretical models to simulate the way in which data occurs. For example, natural language processing has sets of rules which define the way in which languages are generated, and these can be modeled using types of automata. In Chapter 5 we study a class of probabilistic automata and demonstrate how such a model can be learnt from positive examples by an unsupervised learner. In addition to automata, models such as neural networks, Markov models and decision trees are used to allow data to be modeled in an appropriate manner depending on the application.

In classification problems it is common to see data sets being represented by distributions over class labels. In a situation where there are k categories of data spread over some domain X, it is often the case that these k categories can be modeled by probability distributions over X (see [17]) – a form of generative learning. Generative learning can generally be described as generating a discriminant function over the data of each class label and then using these functions in combination to classify observations. This typically takes the form of estimating the distributions over each label and then using a Bayes classifier to select the most likely label for an element in the domain. An alternative approach is to establish the boundaries lying between the classes of data. In doing this we fail to retain the information about the spread of the data over each class, but instead we minimise the amount of data stored. Such a method is the use of support vector machines, which are a widely studied tool for classification and regression problems. This approach of finding decision boundaries between classes is known as discriminative learning and we shall look at the advantages and disadvantages of both the generative and discriminative methods in Section 1.2.4.

1.1 Learning Frameworks

To study a theoretical machine learning problem it is necessary to define the framework in which the algorithm is to function. The framework is basically a set of ground rules

suitable for a particular learning problem – such as the way in which the data is generated, the way data is sampled, and restrictions on the distribution over the data, error rate and confidence parameters. Below we define some of the main learning frameworks relevant to the area of research. Further definitions or additional restrictions are given in later chapters as required.

1.1.1 The PAC-Learning Framework

The Probably Approximately Correct (PAC) learning framework was proposed by Valiant [45] as a way to analyse the complexity of learning algorithms. The emphasis of PAC algorithm design is on the efficiency of the algorithms, which should run in time polynomial in the accuracy and confidence parameters, ϵ and δ, as described below.

A hypothesis h is a discriminative function over the problem domain, which is generated in an attempt to minimise the classification error in relation to the hidden function labelling the data. We refer to the error associated with h as err_h, and let err^* be the error incurred through the optimal choice of h.

Definition 1 *In the PAC-learning framework an algorithm receives labeled samples generated independently according to distribution D over X, where distribution D is unknown, and where labels are generated by an unknown function f from a known class of functions $\mathcal{F}$. In time polynomial in $1/\epsilon$ and $1/\delta$ the algorithm must output a hypothesis h from class $\mathcal{H}$ of hypotheses, such that with probability at least $1-\delta$, $err_h \leq \epsilon$, where ϵ and δ are parameters.*

Notice that in this setting, if $f \in \mathcal{H}$, then $err^* = 0$. Another important case occurs when $\mathcal{H} = \mathcal{F}$. In this case we say that $\mathcal{F}$ is *properly PAC-learnable* by the algorithm (see [26]).

The PAC-learning framework is considered to be rather restrictive for the majority of machine learning problems. The worst case scenario must always be considered in which an adversary is choosing the distributions over the data and the class labels. PAC algorithms must work to the ϵ and δ parameters and always run in polynomial time for the given classes of labelling functions and any distribution over the data. In practice these conditions are not generally necessary as some restrictions on the distributions and functions can be implemented without limiting the usefulness of the algorithms. Many of the negative results associated with the PAC framework are driven by the assumption of distribution independence ([35], for example) – where the distribution of the observations over the domain is independent of the distributions over the class labels.

A particular issue with the PAC framework is the requirement that the data is labeled by a function from a known class of functions, which is impractical in most

situations. This is due both to the fact that in many practical situations the class of functions is unknown and also the fact that the target may not be a function at all (labels may be generated stochastically). These are framework specific problems so slight relaxations of the framework allow for a wider range of problems to be examined.

1.1.2 PAC-Learning with Two Unsupervised Learners

In [22] Goldberg defines a restriction of the PAC framework in which an unknown function $f : X \to \{0, 1\}$ labels the data distributed by D over domain X. This data is divided into subsets $f^{-1}(0)$ and $f^{-1}(1)$, and each learner attempts to construct a discriminant function over one of these sets. When prompted by the algorithm, each learner returns the value its function associates with a given value of $x \in X$. To classify an instance each learner is prompted to return a value associated with the corresponding x, and the learner returning the higher value labels that instance (it is given the class label of the data from its learning set). The learners have no knowledge of the label associated with the data made available to them and no knowledge of the prior probabilities of each class label[2].

Note that the learners can create functions by approximating the distribution over data of their respective class labels and then returning the probability density associated with $x \in X$. In this case, if class priors are known, then the algorithm can use a Bayes classifier to return labels of observations. Note also that the unsupervised learners are not only denied access to the class labels, but they have no way of measuring the empirical error of any classifier based on their respective discriminant functions. This is in contrast to the majority of machine learning algorithms, where the ability to minimise empirical error may prove to be a useful tool.

Formally, we use the definition of the framework from [23] (Definition 1, p.286), where data has label $\ell \in \{0, 1\}$ and D_ℓ represents D restricted to $f^{-1}(\ell)$, which says:

Definition 2 *Suppose algorithm A has access to a distribution P over X, and the output of A is a function $f : X \to \mathbb{R}$. Execute A twice, using D_1 (respectively D_0) for P. Let f_1 and f_0 be the functions obtained respectively. For $x \in X$ let*

$$\begin{aligned} h(x) = 1 \quad &\textit{if} \quad f_1(x) > f_0(x) \\ h(x) = 0 \quad &\textit{if} \quad f_1(x) < f_0(x) \\ h(x) \ \textit{undefined} \quad &\textit{if} \quad f_1(x) = f_0(x) \end{aligned}$$

If A takes time polynomial in $1/\epsilon$ and $1/\delta$, and h is PAC with respect to ϵ and δ, then we will say that A PAC-learns via discriminant functions.

[2] It should be noted that this is equivalent to the case where the learner has access to "positive" and "negative" oracles with no knowledge of the class priors (as in [27]).

Note that "access" to a distribution means that in unit time a sample (an observation of X, without a label) can be drawn from the distribution.

1.1.3 Agnostic PAC-Learning

A common extension of the PAC framework is the Agnostic learning framework (see [5], [32] for example), whereby knowledge of the class of target concepts $\mathcal{F}$ is not assumed. Since the hypothesis class $\mathcal{H}$ may not contain a function which accurately matches the process labelling the data, an agnostic PAC algorithm must attempt to minimise misclassification error in relation to the optimal hypothesis in $\mathcal{H}$ – the aim is to achieve an error no greater than ϵ above the optimal error given class $\mathcal{H}$.

Definition 3 *In the agnostic PAC framework an algorithm receives labeled samples generated independently according to distribution D over X, where distribution D is unknown, and where labels are generated by some unknown process. In time polynomial in $1/\epsilon$ and $1/\delta$ the algorithm must output a hypothesis h from class $\mathcal{H}$ of hypotheses, such that with probability at least $1-\delta$, $err_h \leq err^* + \epsilon$, where ϵ and δ are parameters.*

Note that the framework still requires the adversarial restraints of complying with the worst case scenarios.

1.1.4 Learning Probabilistic Concepts

Probabilistic concepts (or p-concepts) are a tool for modeling problems where a stochastic rule, rather than a function, is labelling the data. We use the notation described in [31], such that $X = [0,1]$ is the domain, and p-concept c is a function $c : X \to [0,1]$. The value $c(x)$ is the probability that a point at $x \in X$ has label 1 (therefore the probability of the point having label 0 is equal to $1 - c(x)$). The framework for learning p-concepts is similar to the agnostic PAC framework – the difference being that in this case the data is being labeled by a process from a known class of probabilistic rules, whereas the agnostic setting assumes no knowledge of the rule labelling the data. The aim of an algorithm learning within the p-concept framework is to minimise the error of its associated classifier, and it should be noted that the optimal classifier commonly has a non-zero error associated with it due to the stochastic nature of the labelling rule.

1.2 Learning Problems

Learning theory differentiates between two main types of off-line[3] learning problems, although others do exist. In the context of a classification problem, *supervised learning* occurs when data consisting of observations and the corresponding labels is sampled. The algorithm is trained with this data and there is the potential for data with different class labels to be treated in different ways (for instance the problem of learning monomials described in [22], where unsupervised learning agents can solve the problem if they have knowledge of the label associated with the data set they are given[4]). Classification problems are learnt by supervised learners as the algorithm must have knowledge of the labels in the training data in order to be able to output a class label when classifying an observation.

Unsupervised learning is the setting of learning with a data set containing observations with no associated labels. Unsupervised learning algorithms typically attempt to recreate the process from which the data is sampled. An example of such an unsupervised learning problem is the problem in Chapter 5 of attempting to recreate the distribution over outputs of the target automaton – the data in this case consists of elements of the domain. Such distribution approximation is a common task of unsupervised learning.

A related topic is *semi-supervised learning*, which will not be covered in any detail here but is worth mentioning due to current research uses in active fields such as computer vision. Semi-supervised learning is the process of using both labeled and unlabeled data to solve classification problems [48]. This will be discussed in the context of generative and discriminative learning later in this chapter.

1.2.1 Distribution Approximation

In order to analyse how good an approximation of a distribution is, we need a way to measure the distance between two distributions. We define two such methods in Section 1.4, namely the variation or L_1 distance, and the Kullback-Leibler divergence or KL-divergence. Both are commonly used measurements. The variation distance is an intuitive measurement as it represents closeness in a way that can inspected manually and draws direct comparisons with the related quadratic distance. The KL-divergence is a widely used measurement as it represents the loss of information associated with using the estimated distribution instead of the true distribution. It is also the case that minimising the KL-divergence between a distribution and the empirical distribution of

[3]Data is sampled and learning takes place prior to the algorithm performing its output functions, as opposed to online learning where the algorithm receives data observations "on the fly".

[4]For instance, the learner given data with label 0 defines a discriminant function $f_0(x) = \frac{1}{2}$ and the learner with label 1 returns the value 1 if some criteria is met and 0 otherwise.

data leads to the maximisation of the likelihood of the data in the sample [1]. There have been a variety of settings in which it has been necessary to learn distributions in the PAC sense of achieving a high accuracy with high confidence, for example [14] shows how to learn mixtures of Gaussian functions in this way, [13] learns distributions over outputs of evolutionary trees (a type of Markov model concerning the evolution of strings), and [30] addresses a number of distribution-learning problems in the PAC setting.

The technique used to approximate the distributions over labels in Chapter 3 is known as a kernel algorithm. Kernel algorithms are widely used to solve density estimation problems (see [17] for example). The idea behind kernel estimation is to give some small probability density weighting to each observation in a data set, and then sum over all of these weightings to produce a distribution. Given a sample of N observations we generate N distributions, each one integrating to $1/N$ and centred at the point of an observation on the domain. We then sum these densities across the whole domain and the resulting distribution is likely to be representative of the distribution over the sample, given certain assumptions about the "smoothness" of the target distribution.

In many cases it can be shown that there is a correlation between L_1 distance and KL-divergence. In [1] it is shown that the learnability of probabilistic concepts (see Section 1.1.4) with respect to KL-divergence is equivalent to learning with respect to quadratic distance, and therefore to L_1 distance. In a similar sense, Chapter 2 shows that learning a distribution with respect to L_1 distance is equivalent to learning under KL-divergence for a restricted subset of distributions.

Distributions can also be defined by probabilistic models such as Markov models and automata. In Chapter 5 we consider the problem of learning probabilistic automata, where the success of the learning process is judged by the proximity of the probability distribution over all outputs of the hypothesis automaton to the distribution over outputs of the target automaton.

1.2.2 PAC-learning via Unsupervised Learners

In [22] a variant of the PAC framework is introduced to allow for PAC-learning classification problems to be solved via unsupervised learners, where sampled data is separated by class label and each subset is learnt by an unsupervised learner[5]. The framework is defined in Section 1.1.2, and we shall extend this to the more general case of learning k classes.

Although the algorithms are supervised learning algorithms as the labels of observations are present in the training data, the fact that the learning process used by

[5]This general approach of learning through distributions over classes used in conjunction with a Bayes Classifier is discussed in [17].

each agent is unsupervised leads to the name "classification via unsupervised learners". There are several reasons for breaking the problem down in this way and learning each class separately. First, it seems the natural way to approach many problems, such as the optical digit recognition in Chapter 3. Finding boundaries between the classes of data seems to be a less intuitive way of solving the problem. In image recognition, the process generating a digit will choose a digit and then generate the corresponding symbol rather than vice versa. In addition to this the process of learning from each class in isolation allows for data from classes to overlap and for this to be reflected by the model. This class overlap is something which cannot occur under the traditional PAC-learning framework, which renders the framework too strict for solving most practical learning problems. In order to compensate for this, it is shown in [22] how to extend the framework to allow for this type of overlap in a similar way to that of the framework for learning probabilistic concepts (see Section 1.1 for explanations of all of these frameworks). Also in the case of a practical problem such as optical character recognition, the fact that each class has been modeled in isolation means that any additions to or reductions from the set of class labels is easily implemented. The models would not have to be recalculated – data from the new class would simply be used to construct an additional class model.

It is also noted that despite the fact that dividing the problem into unsupervised learning tasks can often make it possible to model the class label distributions, this is not necessarily the case (as in Chapter 4). The aim of the learners is simply to produce a set of discriminant functions which work in conjunction with each other – not necessarily to model the distributions themselves. However, in most situations the approach of modeling the distributions is likely to be the desired method due to the benefits described in Section 1.2.4. Other methods of estimating the conditional probability distribution labels exist, such as the use of neural networks [7] or logistic regression.

One of the motivations for this topic is the uncertainty of how to learn a multi-class classification problem with a discriminative function (see [3]). There is no obvious way of extending many discriminative techniques such as support vector machines to separate more than two classes. The problem stems from the way that the method finds a plane of separation between pairs of classes – but where there are more than two classes to separate, there must be some ordering given to the way in which these planes are calculated. Whatever order is chosen it must be the case that the classes of data are being treated differently, whereas when using unsupervised learners to learn each class no differentiation is made between the classes.

1.2.3 PAC-learning Probabilistic Automata

As the other chapters all cover problems associated with learning classifiers which is a supervised learning problem, Chapter 5 deals with the task of modeling an automaton. Probabilistic deterministic finite-state automata, or PDFA, are a useful model for many machine learning problems. Speech recognition and natural language learning can both be modeled by PDFA, and learning PDFA in the PAC-framework has been shown to yield useful results in such practical settings ([41] demonstrates algorithms for building pronunciation models for spoken words and learning joined handwriting).

Expanding on results of [41] for learning acyclic probabilistic automata with a state-merging method (see [8]), [10] shows that PDFA can be PAC-learnt in terms of KL-divergence, although this requires that the expected length of an output is known as a parameter. A further requirement is that the states of the automaton are μ-distinguishable – that all pairs of states emit at least one suffix string with probabilities differing by at least μ. In [30] it is shown that PDFA are capable of encoding a noisy parity function (which it is accepted is not PAC-learnable), and [24] shows that the problem in [10] can be learnt using a more intuitive definition of distinguishability between states allowing for more reasonable similarity between states.

We show that by using a weaker measurement of distribution closeness – L_1 distance rather than KL-divergence – it is possible to dispense with the parameter of the expected length of an output. We also give details of a method of smoothing the distribution (based on observations made in Chapter 2) in order to estimate the target within the required KL-divergence, although the method for applying this smoothing is computationally inefficient. Smoothing of distributions and functions has been examined in [1] where algorithms for smoothing p-concepts are given, and a similar method was used in [13] over strings of restricted length.

1.2.4 Generative and Discriminative Learning Algorithms

By PAC-learning (see Section 1.2) with two unsupervised learners (see Section 1.2.2) we aim to construct discriminant functions over the domain for each class label and then classify data using the functions constructed in correspondence with one another. This is a generative method of learning. We shall now define this term and introduce new terms in order to make the distinction between two forms of generative learning, which we describe as "strong" generative learning and "weak" generative learning, as there is some variation in the literature as to the precise meaning of the term "generative".

Definition 4 Generative Learning *aims to solve multiclass classification problems by generating a discriminant function* $f_y(x) : X \to \mathbb{R}$*, mapping elements of domain X to*

real values, over each label $y \in Y$, such that label y maximising $f_y(x)$ is given to an observation x.

Strong generative learning is a specific case of generative learning (widely referred to as generative learning in the literature), defined as follows.

Definition 5 Strong Generative Learning *solves multiclass classification problems of predicting the class label $y \in Y$ from an observation $x \in X$ (in other words $\arg\max_y\{\Pr[y|x]\}$), by seeking to find the distribution of $\Pr[x|y]$ over each class y, which can then be used to estimate $\Pr[x|y].\Pr[y]$.*

In other words, strong generative learning estimates the joint probability distribution over X and Y. It is generally assumed that the class prior, or a priori probability $\Pr[y]$ (see Section 1.4.2), is known – or at least that it can be estimated relatively accurately from a random sample of data – as we are more interested in the process of estimating the distributions over each label.

Definition 6 Weak Generative Learning *is the method of generative learning with a discriminant function that is not an estimate of the probability density over that class.*

In contrast to generative learning, discriminative algorithms consider the data of all class labels in conjunction with each other, and attempt to find a method of separating the classes.

Definition 7 Discriminative Learning *calculates estimates of class boundaries in a multiclass classification problem, producing a function to classify data with respect to these decision boundaries with no reference to the underlying distributions over observations.*

Of course, although we have used the term "estimates of class boundaries", in practice it is often the case that no such well-defined boundaries exist and that some overlap occurs between classes. This is one of the weaknesses of discriminant learning, in that information about the nature of the class overlap in the empirical data is lost.

There is a general question concerning whether there are classes of problems which can be learnt discriminately but not by generative algorithms. Although discriminative algorithms seem to be theoretically capable of learning a larger class of problems [35], this is balanced against the fact that creating an approximation of the process generating the data is often advantageous in terms of the additional knowledge retained by the learner. We explore this further in Chapter 3, where we demonstrate a practical application of a generative method. We demonstrate the advantages of estimating the distributions over class labels in the context of optical digit recognition – a popular machine learning problem.

We choose the setting of optical digit recognition due to the availability of a good data set for which there is a wealth of known results. It is shown that by learning the distribution representing each of the digits we gain an advantage over standard methods when extending the problem to learning strings of images given some predefined contextual rule. For instance, we examine the problem of learning strings of three digits which must sum to a multiple of ten. The fact that the distributions have been estimated therefore allows for backtracking in cases where an error has been made, and ultimately allows a large proportion of mistakes to be corrected.

For the sake of comparison, we test two methods of optical digit recognition. The method outlined above, estimating the distributions over class labels is a generative technique. In contrast to this we demonstrate a discriminative algorithm that is commonly used in practice when solving classification problems. The technique used is a nonparametric technique known as the k-nearest neighbours algorithm, for which an observation is compared to the k closest observations in the data sample, and the label most prolific in those cases is used to label the observation. Despite the simplicity of this approach it is known to be surprisingly effective.

Strong generative learning is the same as "informative learning" as described in [42]. In this paper the authors compare the usefulness of the approaches of discriminative and strong generative learning

Semi-supervised learning

As previously mentioned, semi-supervised learning can be used to implement aspects of both discriminative and generative learning in situations where both labeled and unlabeled data is observed. In computer vision learning problems (such as object recognition) it is difficult to rely on supervised learning alone due to the lack of labeled data (the labelling must be performed by humans or highly specialised agents on the whole). It is shown in [37] that discriminative algorithms may perform less well on small amounts of data than generative algorithms (specifically the generative approach of the naive Bayes model and the discriminative method of using a linear classifier/logistic regression). A typical method of combining the two varieties of learning is to learn from the labeled data using a discriminative algorithm, and then apply the resulting classifier to the unlabeled data. The unlabeled data fitting well within the decision boundaries is then classified with the appropriate label and then the algorithm is trained again using this augmented data set. This is known as *self-training*. Another method, *co-training*, is to divide the feature set into two subsets, and learn from the labeled data using two discriminative algorithms – one using each subset of features. Again, once the classifiers have been learnt, they are applied to the unlabeled data, and the new data labeled by each algorithm is used to augment the data set of the algorithm using the other subset

of features before the training is repeated. Research into the optimal way of combining discriminative and generative classification is discussed in recent papers [33] and [16].

1.3 Questions to Consider

A question posed by Goldberg (in [22], [23]) is whether a class of learning problem exists which is solvable within the PAC-learning framework but not PAC-learnable using unsupervised learners. More generally we must examine the question of how much harder it is to learn if we must learn the distributions over classes. This problem is considered in part in Chapter 2. Here we show that if the distributions over labels have been PAC-learnt in polynomial time, then we are able to PAC-learn the associated classifier (of course we are not talking about PAC-learning in the strict sense – rather in the agnostic setting). However, this leaves open the question in relation to PAC-learning distributions and whether this is always possible. This problem of learning distributions has been discussed in 1.2.1 and Chapter 5 is concerned with learning the class of distributions representing PDFA

In [22] it is speculated that by restricting the distribution over observations to one belonging to a predefined subset (as was necessary to learn the class of monomials and rectangles in the plane using unsupervised learners in the same paper), it may be the case that PAC-learning using unsupervised learners in this restricted setting is equivalent to strict PAC-learning. In [23] a looser definition of the problems setting is also stated, where Definition 2 has the additional aspect that the distribution D over all observations is accessible by the algorithm. This leads to results such as the learnability of a restricted class of monomials as mentioned above. The equivalence of PAC-learning via discriminant functions (see Definition 2) to various related forms of learning framework is shown. It is shown that (under the noisy parity assumption) learning in this way is distinct from PAC-learning under uniform noise. It follows that this unsupervised learners framework is less restrictive.

The main questions we consider are the following:

- Are there problems learnable under the standard PAC conditions which are not learnable with unsupervised learners?
- What advantage is gained by learning with unsupervised learners over a discriminative algorithm?
- How much harder is it to learn with unsupervised learners?

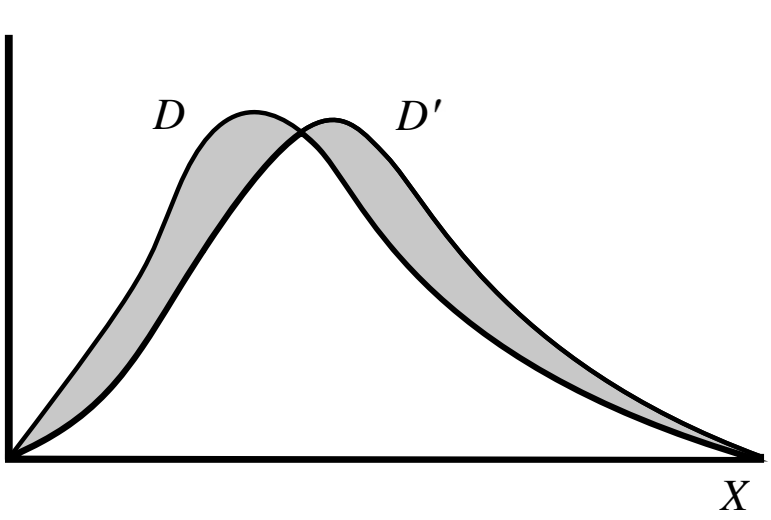


Figure 1.1: L_1 distance.

1.4 Terms and Definitions

We now define a variety of terminology that is used throughout the thesis. Any symbols or terms used in the later chapters are generally defined at the time of use, but as there are common themes running through the research it is useful to define some standard terms here.

1.4.1 Measurements Between Distributions

Suppose D and D' are probability distributions over the same domain X. The L_1 distance (also referred to as variation distance) between D and D' is defined as follows.

Definition 8 $L_1(D, D') = \int_X |D(x) - D'(x)| \, dx$.

We usually assume that X is a discrete domain, in which case

$$L_1(D, D') = \sum_{x \in X} |D(x) - D'(x)|.$$

The L_1 distance between distributions D and D' is illustrated in Figure 1.1. The shaded region represents the integral between the two curves, or the sum of the differences over a discrete scale.

The Kullback-Leibler divergence (KL-divergence) between distributions D and D' is also known as the *relative entropy*. It is a measurement commonly associated with information theoretic settings, where D represents the "true" distribution and D' represents an approximation of D.

Definition 9 $I(D||D') = \sum_{x \in X} D(x) \log \left(\frac{D(x)}{D'(x)} \right)$.

Note that the KL-divergence is not symmetric and that its value is always non-negative. (See Cover and Thomas [12] for further details.)

1.4.2 A Priori and A Posteriori Probabilities

In multiclass classification problems data is generated and labeled by some random process according to the particular learning problem being studied. The term "a priori probability" of a data sample having label ℓ is the probability that a randomly generated point will be given label ℓ by the process labelling the points, prior to the point being generated. The a priori probability of a label ℓ is also referred to as the class prior of ℓ, which is denoted g_ℓ.

Definition 10 $g_\ell = \sum_{x \in X} \Pr(\ell|x) . D(x)$

The probability of an instance being labeled ℓ given that it occurs at $x \in X$ is known as the "a posteriori probability" of label ℓ, and is denoted $\Pr(\ell|x)$.

It is assumed in Chapter 2 (and a similar assumption is made in Chapter 4) that the a priori probabilities of the k classes are known. This may or may not be the case depending on the setting, but it is a reasonable restriction to make on the problem. In reality these class priors can be estimated within additive error ϵ using standard Chernoff Bounds, from a sample size polynomial in ϵ, δ and k, with confidence at least $1 - \delta$.

1.4.3 Loss/Cost of a Classifier

The performance of a classifier (or discriminant function) is usually assessed by way of a loss function (or *cost function*)[6]. The most basic loss function is a linear loss function – the function incurs a unit loss for any misclassification of a data point and a loss of zero otherwise. In multiclass classification problems a cost matrix may be defined, whereby the cost of misclassifying data varies according to the label assigned.

Let $\mathcal{L}$ be the set of all class labels and let f be a discriminant function defined on domain X, such that $f : X \to \mathcal{L}$. A cost matrix $\mathcal{C}$ may be used (it is often unnecessary – for instance in the case of 2 classes) to specify the cost associated with any classification – where c_{ij} is the cost of classifying a data point which has label i as label j. In the case of a basic linear loss function the matrix would consist of a grid of 1s with 0s on the diagonal, with $c_{ij} = 0$ if $i = j$, and 1 elsewhere.

[6] The terms loss and cost are used interchangeably in this context.

We often use D_ℓ to signify the distribution over data with label ℓ in multiclass classification problems, where D is a mixture of these distributions weighted by their class priors g_ℓ, $D(x) = \sum_{\ell \in \mathcal{L}} g_\ell . D_\ell(x)$.

The expected cost, $\alpha(x, f(x))$, associated with classifier f at a given value x in the domain is the sum of the cost $c_{\ell\ f(x)}$ associated with each label $\ell \in \mathcal{L}$, weighted by the a posteriori probability of that label at x, which is $g_\ell . D_\ell(x)/D(x)$.

Definition 11 $\alpha(x, f(x)) = \sum_{\ell \in \mathcal{L}} g_\ell . D_\ell(x) . D(x)^{-1} . c_{\ell\ f(x)}$.

The *risk* associated with function f is the expectation of the loss incurred by f when classifying a randomly generated data point. The risk is obtained by averaging $\alpha(x, f(x))$ over X.

Definition 12 $R(f) = \int_{x \in X} D(x) . \alpha(x, f(x))\ dx = \int_{x \in X} \sum_{\ell \in \mathcal{L}} g_\ell . D_\ell(x) . c_{\ell\ f(x)}\ dx$.

Over a discrete domain, this is equivalent to

$$R(f) = \sum_{x \in X} \sum_{\ell \in \mathcal{L}} g_\ell . D_\ell(x) . c_{\ell\ f(x)}.$$

The general aim of a classification algorithm is to output a function which minimises its risk. The *Bayes classifier* associated with two or more probability distributions is the function that maps an element x of the domain to the label associated with the probability distribution whose value at x is largest. This is a well-known approach for classification, see [17]. Given knowledge of the true underlying probability distributions, the optimal classifier is known as the Bayes optimal classifier.

Definition 13 *The Bayes Optimal Classifier, denoted f^*, is the classifier in $\mathcal{H}$ minimising the risk such that:*

$$f^* = \arg\min_f \sum_{x \in X} \sum_{\ell \in \mathcal{L}} g_\ell . D_\ell(x) . c_{\ell\ f(x)}$$

over discrete domain X.

In cases where $R(f^*) > 0$, the goal is still to minimise the risk associated with the classifier – but since the risk cannot be reduced to 0, the aim is to achieve a risk as close to $R(f^*)$ as possible. For this purpose the term *regret* is introduced, where regret is equal to the risk associated with the classifier in question, minus the risk associated with the optimal classifier.

Definition 14 $Regret(f) = R(f) - R(f^*)$.

1.5 Synopsis

The contents of each chapter are as follows:

Chapter 2 – PAC Classification from PAC Estimates of Distributions

In this chapter we examine the problem of solving multiclass classification tasks in a variation of the PAC framework allowing for stochastic concepts (including p-concepts) to be learnt. For the method of learning each class label distribution using unsupervised learners, we show that if these distributions can be PAC learnt under L_1 distance or KL-divergence then this implies PAC learnability of the classifier by using the Bayes classifier in conjunction with these estimated distributions. A general smoothing technique showing the equivalence of learning under L_1 distance and KL-divergence for a restricted class of distributions is described.

Chapter 3 – Optical Digit Recognition

Here we study the practical task of optical character recognition, and use the method of estimating distributions over each class label (as described in Chapter 2) with unsupervised learners to classify images of handwritten digits. We compare the results obtained using this method with the results obtained by using a standard discriminative algorithm – the k nearest neighbour algorithm. Having seen how the algorithms compare for single digit recognition, we explore the benefits of the strong generative learning approach when classifying strings of digits obeying a variety of contextual rules.

Chapter 4 – Learning Probabilistic Concepts

We show that unsupervised learners can be used to solve the problem of learning the class of p-concepts consisting of functions with at most k turning points, as an extension to the problem solved in [31] of learning the class of non-decreasing functions.

It should be noted that the algorithm used is not a strong generative algorithm as the learners do not attempt to model the distributions over the classes. Rather this demonstrates that a weak generative algorithm can be used in situations where it is hard to estimate the distributions over labels, and an example is given of why this is the case.

Chapter 5 – Learning PDFA

Probabilistic automata are a widely used model for many sequential learning problems. As probabilistic automata define probability density functions over their outputs they are also useful in conjunction with the methods of Chapter 2. We learn a class of probabilistic automata with respect to L_1 distance, using a variation of an established

state-merging algorithm, and show that the use of this distance metric allows us to dispense with the need for the parameter of expected string length (as is necessary when learning with respect to KL-divergence as shown in [10]). We demonstrate that the method of smoothing from L_1 distance to KL-divergence in Chapter 2 can be used in relation to a restricted class of probabilistic automaton, which shows that for this class, learning under L_1 distance is equivalent to learning under KL-divergence (although this is far from efficient).

Chapter 6 – Conclusion

Finally we draw conclusions about the respective benefits and drawbacks of performing classification using unsupervised learners. We discuss the benefits of the generative learning approach and the implications of applying such techniques to practical problems.

Chapter 2

PAC Classification from PAC Estimates of Distributions

In this chapter we consider a general approach to pattern classification in which elements of each class are first used to train a probabilistic model via some unsupervised learning method. The resulting models for each class are then used to assign discriminant scores to an unlabeled instance, and a label is chosen to be the one associated with the model giving the highest score. This approach is used in Chapter 3 where learners give scores corresponding to the digit they have been trained on to images of digits, and [6] uses this approach to classify protein sequences by training a probabilistic suffix tree model (of Ron et al. [41]) on each sequence class. Even where an unsupervised technique is mainly being used to gain insight into the process that generated two or more data sets, it is still sometimes instructive to try out the associated classifier, since the misclassification rate provides a quantitative measure of the accuracy of the estimated distributions.

The work of [41] has led to further related algorithms for learning classes of probabilistic finite state automata (PDFAs) in which the objective of learning has been formalised as the estimation of a true underlying distribution over strings output by the target PDFA with a distribution represented by a hypothesis PDFA. The natural discriminant score to assign to a string is the probability that the hypothesis would generate that string at random. As one might expect, the better one's estimates of label class distributions (the class-conditional densities), the better the associated classifier should be. The aim of this chapter is to make precise that observation. Bounds are given on the risk of the associated Bayes classifier (see Section 1.4.3) in terms of the quality of the estimated distributions.

These results are partly motivated by an interest in the relative merits of estimating a class-conditional distribution using the variation distance, as opposed to the KL-divergence. In [10] it has been shown how to learn a class of PDFAs using KL-

divergence, in time polynomial in a set of parameters that includes the expected length of strings output by the automaton. In Chapter 5 we examine how this class can be learnt with respect to variation distance, with a polynomial sample-size bound that is independent of the length of output strings. Furthermore, it can be shown that it is necessary to switch to the weaker criterion of variation distance in order to achieve this. We show here that this leads to a different—but still useful—performance guarantee for the Bayes classifier.

Abe and Warmuth [2] study the problem of learning probability distributions using the KL-divergence via classes of probabilistic automata. Their criterion for learnability is that—for an unrestricted input distribution D—the hypothesis PDFA should be as close as possible to D (i.e. within ϵ). Abe et al. [1] study the negative log-likelihood loss function in the context of learning *stochastic rules*, i.e. rules that associate an element of the domain X to a probability distribution over the range Y of class labels. We show here that if two or more label class distributions are learnable in the sense of [2], then the resulting stochastic rule (the conditional distribution over Y given $x \in X$) is learnable in the sense of [1].

If the label class distributions are well estimated using the variation distance, then the associated classifier may not have a good negative log-likelihood risk, but will have a *misclassification rate* that is close to optimal. This result is for general k-class classification, where distributions may overlap (i.e. the optimum misclassification rate may be positive). We also incorporate variable misclassification penalties (sometimes one might wish a false negative to cost more than a false positive – consider, for example, the case of medical diagnosis from image analysis), and show that this more general loss function is still approximately minimised provided that discriminant likelihood scores are rescaled appropriately.

As a result we show that PAC-learnability and more formally, p-concept learnability (defined in Section 1.1 – see Chapter 4 for further explanation), follows from the ability to learn class distributions in the setting of Kearns et al. [30]. Papers such as [13, 20, 36] study the problem of learning various classes of probability distributions with respect to KL-divergence and variation distance, in this setting.

It is well-known (noted in [31]) that learnability with respect to KL-divergence is stronger than learnability with respect to variation distance. Furthermore, the KL-divergence is usually used (for example in [10, 29]) due to the property that when minimised with respect to a sample, the empirical likelihood of that sample is maximised.

It appears that Theorem 16 is essentially a generalisation of Exercise 2.10 of Devroye et al's textbook [15], from 2 class to multiple classes, and in addition we show here that variable misclassification costs can be incorporated. This is the closest thing that has been found to this Theorem which has already appeared but it is suspected

that other related results may have appeared. Theorem 17 is another result which may be known, but likewise no statement of it has been found.

2.1 The Learning Framework

We consider a k-class classification setting, where labeled instances are generated by distribution D over $X \times \{1, ..., k\}$. The aim is to predict the label ℓ associated with $x \in X$, where x is generated by the marginal distribution of D on X, $D|_X$. A non-negative cost is incurred for each classification, based either on a cost matrix (where the cost depends upon both the hypothesised label and the true label) or the negative log-likelihood of the true label being assigned. The aim is to optimise the expected cost, or risk, associated with the occurrence of a randomly generated example.

Let D_ℓ be D restricted to points (x, ℓ), $\ell = \{1, ..., k\}$. D is a mixture $\sum_{\ell=1}^{k} g_\ell D_\ell$, where $\sum_{i=1}^{k} g_i = 1$, and g_ℓ is the a priori probability of class ℓ.

The PAC-learning framework described previously is unsuitable for learning stochastic models such as the one described in this chapter. Note that PAC-learning requires the concept labelling data to belong to a known class of functions, and in this case a stochastic process is generating labels. Instead we use a variation on the framework used in [31] for learning p-concepts – as described in Section 1.1 – which adopts performance measures from the PAC model, extending this to learn stochastic rules with k classes. Rather than having a function $c : X \to [0, 1]$ mapping members of the domain to probabilities (such that $c(x)$ represents the a posteriori probability of an instance at x having label 1), we have k classes so the equivalent function would map elements of X to a k-tuplet of real values summing to 1, representing the a posteriori probabilities of the k labels for any $x \in X$.

Our notion of learning distributions is similar to that of Kearns et al. [30].

Definition 15 *Let $\mathcal{D}_n$ be a class of distributions over n labels across domain X. $\mathcal{D}_n$ is said to be efficiently learnable if an algorithm A exists such that given $\epsilon > 0$ and $\delta > 0$, and access to randomly drawn examples (see below) from any unknown target distribution $D \in \mathcal{D}_n$, A runs in time polynomial in $1/\epsilon$, $1/\delta$ and n and returns a probability distribution D' that with probability at least $1 - \delta$ is within ϵ L_1 distance (alternatively KL-divergence) of D.*

The following results show that if estimates of the distributions over each class label are known (to an accuracy in terms of ϵ, with confidence in terms of δ), then the discriminative function optimised on these estimated distributions is such that the function operates within ϵ accuracy of the optimal classifier, with confidence at least $1 - \delta$ from a sample size polynomial in these parameters.

2.2 Results

In Section 2.2.1 we give bounds on the risk associated with a hypothesis, with respect to the accuracy of the approximation of the underlying distribution generating the instances. In Section 2.2.2 we show that these bounds are close to optimal, and in Section 2.2.3 we give corollaries showing what these bounds mean for PAC learnability.

We define the accuracy of an approximate distribution in terms of L_1 distance and KL-divergence. It is assumed that the class priors of each class label are known.

2.2.1 Bounds on Regret

In terms of L_1 distance

First we examine the case where the accuracy of the hypothesis distribution is such that the distribution for each class label is within ϵ L_1 distance of the true distribution for that label, for some $0 \leq \epsilon \leq 1$. Cost matrix C specifies the cost associated with any classification, where $c_{ij} \geq 0$. It is usually the case that $c_{ij} = 0$ for $i = j$.

The risk associated with classifier f over discrete domain X, $f : X \to \{1, ..., k\}$, is given by $R(f) = \sum_{x \in X} \sum_{i=1}^{k} c_{if(x)}.g_i.D_i(x)$ (as defined in Definition 12).

Let f^* be the Bayes optimal classifier, and let $f'(x)$ be the function with optimal expected cost with respect to alternative distributions $D'_i, i \in \{1, ..., k\}$. For $x \in X$,

$$\begin{aligned} f^*(x) &= \arg\min_j \textstyle\sum_{i=1}^{k} c_{ij}.g_i.D_i(x), \text{ and} \\ f'(x) &= \arg\min_j \textstyle\sum_{i=1}^{k} c_{ij}.g_i.D'_i(x). \end{aligned}$$

Recall that "regret" is defined in Definition 14 such that $Regret(f') = R(f') - R(f^*)$.

Theorem 16 *Let f^* be the Bayes optimal classifier and let f' be the classifier associated with estimated distributions D'_i. Suppose that for each label $i \in \{1, ..., k\}$, $L_1(D_i, D'_i) \leq \epsilon/g_i$. Then $Regret(f') \leq \epsilon.k.\max_{ij}\{c_{ij}\}$.*

Proof: Let $R_f(x)$ be the contribution from $x \in X$ towards the total expected cost associated with classifier f. For f such that $f(x) = j$,

$$R_f(x) = \sum_{i=1}^{k} c_{ij}.g_i.D_i(x).$$

Let $\tau_{\ell' - \ell}(x)$ be the increase in risk for labelling x as ℓ' instead of ℓ, so that

$$\begin{aligned} \tau_{\ell' - \ell}(x) &= \textstyle\sum_{i=1}^{k} c_{i\ell'}.g_i.D_i(x) - \sum_{i=1}^{k} c_{i\ell}.g_i.D_i(x) \\ &= \textstyle\sum_{i=1}^{k} (c_{i\ell'} - c_{i\ell}).g_i.D_i(x). \end{aligned} \tag{2.1}$$

Note that due to the optimality of f^* on D_i, $\forall x \in X : \tau_{f'(x)-f^*(x)}(x) \geq 0$. In a similar way, the expected contribution to the total cost of f' from x must be less than or equal to that of f^* with respect to D'_i – given that f' is chosen to be optimal on the D'_i values. We have $\sum_{i=1}^{k} c_{if'(x)}.g_i.D'_i(x) \leq \sum_{i=1}^{k} c_{if^*(x)}.g_i.D'_i(x)$. Rearranging this, we get

$$\sum_{i=1}^{k} D'_i(x).g_i.\left(c_{if^*(x)} - c_{if'(x)}\right) \geq 0. \tag{2.2}$$

From Equations 2.1 and 2.2 it can be seen that

$$\begin{aligned} \tau_{f'(x)-f^*(x)}(x) &\leq \sum_{i=1}^{k} \left(D_i(x) - D'_i(x)\right).g_i.\left(c_{if'(x)} - c_{if^*(x)}\right) \\ &\leq \sum_{i=1}^{k} \left|\left(D_i(x) - D'_i(x)\right)\right|.g_i.\left|\left(c_{if'(x)} - c_{if^*(x)}\right)\right|. \end{aligned}$$

Let $d_i(x)$ be the difference between the probability densities of D_i and D'_i at $x \in X$, $d_i(x) = |D_i(x) - D'_i(x)|$. Therefore,

$$\begin{aligned} \tau_{f'(x)-f^*(x)}(x) &\leq \sum_{i=1}^{k} |c_{if'(x)} - c_{if^*(x)}|.g_i.d_i(x) \\ &\leq \sum_{i=1}^{k} \max_j\{c_{ij}\}.g_i.d_i(x). \end{aligned}$$

In order to bound the expected cost, it is necessary to sum over X.

$$\sum_{x\in X} \tau_{f'(x)-f^*(x)}(x) \leq \sum_{x\in X}\sum_{i=1}^{k} \max_j\{c_{ij}\}.g_i.d_i(x) = \sum_{i=1}^{k} \max_j\{c_{ij}\}.g_i.\sum_{x\in X} d_i(x). \tag{2.3}$$

Since $L_1(D_i, D'_i) \leq \epsilon/g_i$ for all i, ie. $\sum_{x\in X} d_i(x) \leq \epsilon/g_i$, it follows from Equation 2.3 that $\sum_{x\in X} \tau(x) \leq \sum_{i=1}^{k} \max_j\{c_{ij}\}.g_i.\left(\frac{\epsilon}{g_i}\right)$. This expression gives an upper bound on expected cost for labelling x as $f'(x)$ instead of $f^*(x)$. By definition, $\sum_{x\in X} \tau(x) = R(f') - R(f^*) = Regret(f')$. Therefore it has been shown that

$$R(f') \leq R(f^*) + \epsilon.\sum_{i=1}^{k} \max_j\{c_{ij}\} \leq R(f^*) + \epsilon.k.\max_{ij}\{c_{ij}\},$$

and consequently that $Regret(f') \leq \epsilon.k.\max_{ij}\{c_{ij}\}$. □

In terms of KL-divergence

We next prove a corresponding result in terms of KL-divergence, for which we use the negative log-likelihood of the correct label as the cost function. We define $\Pr_i(x)$ to be

the probability that a data point at x has label i (the a posteriori probability of i given x), such that $\Pr_i(x) = g_i.D_i(x)\left(\sum_{j=1}^{k} g_j.D_j(x)\right)^{-1}$. We define $f : X \to \mathbb{R}^k$, where $f(x)$ is an estimation of the a posteriori probabilities of each label $i \in \{1, ..., k\}$ given $x \in X$, and let $f_i(x)$ represent f's estimate of the a posteriori probability of the i'th label at x, such that $\sum_{i=1}^{k} f_i(x) = 1$. The risk associated with f can be expressed as

$$R(f) = \sum_{x \in X} D(x) \sum_{i=1}^{k} -\log(f_i(x)).\Pr_i(x). \tag{2.4}$$

Let $f^* : X \to \mathbb{R}^k$ output the true class label distribution for an element of X. From Equation 2.4 it can be seen that

$$R(f^*) = \sum_{x \in X} D(x) \sum_{i=1}^{k} -\log(\Pr_i(x)).\Pr_i(x). \tag{2.5}$$

Theorem 17 *For $f : X \to \mathbb{R}^k$ suppose that $R(f)$ is given by Equation 2.4. If for each label $i \in \{1, ..., k\}$, $I(D_i||D'_i) \leq \epsilon/g_i$, then $Regret(f') \leq k\epsilon$.*

Proof: Let $R_f(x)$ be the contribution at $x \in X$ to the risk associated with classifier f, $R_f(x) = \sum_{i=1}^{k} -\log(f_i(x)).\Pr_i(x)$. Therefore $R(f') = \sum_{x \in X} D(x).R_{f'}(x)$.

We define $\Pr'_i(x)$ to be the estimated probability that a data point at $x \in X$ has label $i \in \{1, ..., k\}$, from distributions D'_i, such that $\Pr'_i(x) = g_i.D'_i\left(\sum_{j=1}^{k} g_j.D'_j(x)\right)^{-1}$. It is the case that

$$R_{f'}(x) = D(x).\sum_{i=1}^{k} -\log\left(\Pr'_i(x)\right).\Pr_i(x).$$

Let $\xi(x)$ denote the contribution to additional risk incurred from using f' as opposed to f^* at $x \in X$.[1] We define D' such that $D'(x) = \sum_{i=1}^{k} g_i.D'_i(x)$ (and of

[1] The contribution towards $Regret(f')$.

course $D(x) = \sum_{i=1}^{k} g_i.D_i(x)$). From Equation 2.5 it can be seen that

$$\begin{aligned}
\xi(x) &= R_{f'}(x) - D(x).\sum_{i=1}^{k} -\log\left(\Pr_i(x)\right).\Pr_i(x) \\
&= D(x).\sum_{i=1}^{k} \Pr_i(x).\left(\log\left(\Pr_i(x)\right) - \log\left(\Pr'_i(x)\right)\right) \\
&= D(x).\sum_{i=1}^{k} \left(\frac{g_i.D_i(x)}{D(x)}\right)\left(\log\left(\frac{g_i.D_i(x)}{D(x)}\right) - \log\left(\frac{g_i.D'_i(x)}{D'(x)}\right)\right) \\
&= D(x).\sum_{i=1}^{k} \left(\left(\frac{g_i.D_i(x)}{D(x)}\right).\left(\log\left(\frac{g_i.D_i(x)}{g_i.D'_i(x)}\right) - \log\left(\frac{D(x)}{D'(x)}\right)\right)\right) \\
&= \sum_{i=1}^{k} \left(g_i.D_i(x)\log\left(\frac{D_i(x)}{D'_i(x)}\right)\right) - D(x)\log\left(\frac{D(x)}{D'(x)}\right).
\end{aligned}$$

We define $I(D||D')(x)$ to be the contribution at $x \in X$ to the KL-divergence, such that $I(D||D')(x) = D(x)\log\left(D(x)/D'(x)\right)$. It follows that

$$\sum_{x\in X} \xi(x) = \sum_{i=1}^{k} \left(g_i.I(D_i||D'_i)\right) - I(D||D'). \tag{2.6}$$

We know that the KL-divergence between D_i and D'_i is bounded by ϵ/g_i for each label $i \in \{1, ..., k\}$, so Equation 2.6 can be rewritten as

$$\sum_{x\in X} \xi(x) \leq \sum_{i=1}^{k} \left(g_i.\left(\frac{\epsilon}{g_i}\right)\right) - I(D||D') \leq k.\epsilon - I(D||D').$$

Due to the fact that the KL-divergence between two distributions is non-negative, an upper bound on the cost can be obtained by letting $I(D||D') = 0$, so $R(f') - R(f^*) \leq k\epsilon$. Therefore it has been proved that $Regret(f') \leq k\epsilon$. □

2.2.2 Lower Bounds

In this section we give lower bounds corresponding to the two upper bounds given in Section 2.2.

Example 18 *Consider a distribution D over domain $X = \{x_0, x_1\}$, from which data is generated with labels 0 and 1 and there is an equal probability of each label being generated ($g_0 = g_1 = \frac{1}{2}$). $D_i(x)$ denotes the probability that a point is generated at*

$x \in X$ given that it has label i. D_0 and D_1 are distributions over X, such that at $x \in X$, $D(x) = \frac{1}{2}(D_0(x) + D_1(x))$.

Suppose that D'_0 and D'_1 are approximations of D_0 and D_1, and that $L_1(D_0, D'_0) = \frac{\epsilon}{g_0} = 2\epsilon$ and $L_1(D_1, D'_1) = \frac{\epsilon}{g_1} = 2\epsilon$, where $\epsilon = \epsilon' + \gamma$ (and γ is an arbitrarily small constant).

Given the following distributions, assuming that a misclassification results in a cost of 1 and that a correct classification results in no cost, it can be seen that $R(f^*) = \frac{1}{2} - \epsilon'$:

$$D_0(x_0) = \frac{1}{2} + \epsilon', D_0(x_1) = \frac{1}{2} - \epsilon',$$

$$D_1(x_0) = \frac{1}{2} - \epsilon', D_1(x_1) = \frac{1}{2} + \epsilon'.$$

Now if we have approximations D'_0 and D'_1 as shown below, it can be seen that f' will misclassify for every value of $x \in X$:

$$D'_0(x_0) = \frac{1}{2} - \gamma, D'_0(x_1) = \frac{1}{2} + \gamma,$$

$$D'_1(x_0) = \frac{1}{2} + \gamma, D'_1(x_1) = \frac{1}{2} - \gamma.$$

This results in $R(f') = \frac{1}{2} + \epsilon'$. Therefore $R(f') = R(f^*) + 2\epsilon' = R(f^*) + 2(\epsilon - \gamma)$.

In this example the regret is only 2γ lower than $R(f^*) + \epsilon.k.\max_j\{c_{ij}\}$, since $k = 2$. A similar example can be used to give lower bounds corresponding to the upper bound given in Theorem 17.

Example 19 *Consider distributions D_0, D_1, D'_0 and D'_1 over domain $X = \{x_0, x_1\}$ as defined in Example 18. It can be seen that the KL-divergence between each label's distribution and its approximated distribution is*

$$I(D_0||D'_0) = I(D_1||D'_1) = \left(\frac{1}{2} + \epsilon'\right) \log\left(\frac{\frac{1}{2} + \epsilon'}{\frac{1}{2} - \gamma}\right) + \left(\frac{1}{2} - \epsilon'\right) \log\left(\frac{\frac{1}{2} - \epsilon'}{\frac{1}{2} + \gamma}\right).$$

The optimal risk, measured in terms of negative log-likelihood, can be expressed as $R(f^*) = -\left(\frac{1}{2} + \epsilon'\right) \log\left(\frac{1}{2} + \epsilon'\right) - \left(\frac{1}{2} - \epsilon'\right) \log\left(\frac{1}{2} - \epsilon'\right)$. The risk incurred by using f' as the discriminant function is $R(f') = -\left(\frac{1}{2} + \epsilon'\right) \log\left(\frac{1}{2} - \gamma\right) - \left(\frac{1}{2} - \epsilon'\right) \log\left(\frac{1}{2} + \gamma\right)$.

Hence as γ approaches zero,

$$R(f') = R(f^*) + \left(\frac{1}{2} + \epsilon'\right) \log\left(\frac{\frac{1}{2} + \epsilon'}{\frac{1}{2} - \gamma}\right) + \left(\frac{1}{2} - \epsilon'\right) \log\left(\frac{\frac{1}{2} - \epsilon'}{\frac{1}{2} + \gamma}\right) = R(f^*) + \epsilon.$$

2.2.3 Learning Near-Optimal Classifiers in the PAC Sense

We show that the results of Section 2.2.1 imply learnability within the framework defined in Section 2.1.

The following corollaries refer to algorithms A_{class} and $A_{class'}$. These algorithms generate classifier functions $f' : X \rightarrow \{1, 2, \ldots, k\}$, which label data in a k-label classification problem, using L_1 distance and KL-divergence respectively as measurements of accuracy.

Corollary 20 shows (using Theorem 16) that a near-optimal classifier can be constructed given that an algorithm exists which approximates a distribution over positive data in polynomial time. We are given cost matrix C, and assume knowledge of the class priors g_i.

Corollary 20 *If an algorithm A_{L_1} approximates distributions within L_1 distance ϵ' with probability at least $1 - \delta'$, in time polynomial in $1/\epsilon'$ and $1/\delta'$, then an algorithm A_{class} exists which (with probability $1 - \delta$) generates a discriminant function f' with an associated risk of at most $R(f^*) + \epsilon$, and A_{class} is polynomial in $1/\delta$ and $1/\epsilon$.*

Proof: A_{class} is a classification algorithm which uses unsupervised learners to fit a distribution to each label $i \in \{1, ..., k\}$, and then uses the Bayes classifier with respect to these estimated distributions, to label data.

A_{L_1} is a PAC algorithm which learns from a sample of positive data to estimate a distribution over that data. A_{class} generates a sample N of data, and divides N into sets $\{N_1, ..., N_k\}$, such that N_i contains all members of N with label i. Note that for all labels i, $|N_i| \approx g_i.|N|$.

With a probability of at least $1 - \frac{1}{2}(\delta/k)$, A_{L_1} generates an estimate D' of the distribution D_i over label i, such that $L_1(D_i, D') \leq \epsilon \left(g_i.k.\max_{ij}\{c_{ij}\}\right)^{-1}$. Therefore the size of the sample $|N_i|$ must be polynomial in $g_i.k.\max_{ij}\{c_{ij}\}/\epsilon$ and k/δ. For all $i \in \{1, ..., k\}$ $g_i \leq 1$, so $|N_i|$ is polynomial in $\max_{ij}\{c_{ij}\}$, k, $1/\epsilon$ and $1/\delta$.

When A_{class} combines the distributions returned by the k iterations of A_{L_1}, there is a probability of at least $1-\delta/2$ that all of the distributions are within $\epsilon \left(g_i.k.\max_{ij}\{c_{ij}\}\right)^{-1}$ L_1 distance of the true distributions (given that each iteration received a sufficiently large sample). We allow a probability of $\delta/2$ that the initial sample N did not contain a good representation of all labels ($\neg\forall i \in \{1, ...k\} : |N_i| \approx g_i.|N|$), and as such – one or more iteration of A_{L_1} may not have received a sufficiently large sample to learn the distribution accurately.

Therefore with probability at least $1-\delta$, all approximated distributions are within $\epsilon(g_i.k.\max_{ij}\{c_{ij}\})^{-1}$ L_1 distance of the true distributions. If we use the classifier which is optimal on these approximated distributions, f', then the increase in risk associated with using f' instead of the Bayes Optimal Classifier, f^*, is at most ϵ. It has been

shown that A_{L_1} requires a sample of size polynomial in $1/\epsilon$, $1/\delta$, k and $\max_{ij}\{c_{ij}\}$. It follows that

$$|N| = \sum_{i=1}^{k} |N_i| = \sum_{i=1}^{k} p\left(\frac{1}{\epsilon}, \frac{1}{\delta}, k, \max_{ij}\{c_{ij}\}\right) \in O\left(p\left(\frac{1}{\epsilon}, \frac{1}{\delta}, k, \max_{ij}\{c_{ij}\}\right)\right).$$

□

Corollary 21 shows (using Theorem 17) how a near-optimal classifier can be constructed given that an algorithm exists which approximates a distribution over positive data in polynomial time.

Corollary 21 *If an algorithm A_{KL} has a probability of at least $1-\delta$ of approximating distributions within ϵ KL-divergence, in time polynomial in $1/\epsilon$ and $1/\delta$, then an algorithm $A_{class'}$ exists which (with probability $1-\delta$) generates a function f' that maps $x \in X$ to a conditional distribution over class labels of x, with an associated log-likelihood risk of at most $R(f^*)+\epsilon$, and $A_{class'}$ is polynomial in $1/\delta$ and $1/\epsilon$.*

Proof: $A_{class'}$ is a classification algorithm using the same method as A_{class} in Corollary 20, whereby a sample N is divided into sets $\{N_1, ..., N_k\}$, and each set is passed to algorithm A_{KL} where a distribution is estimated over the data in the set.

With a probability of at least $1-\frac{1}{2}(\delta/k)$, A_{KL} generates an estimate D' of the distribution D_i over label i, such that $I(D_i||D') \leq \epsilon(g_i.k)^{-1}$. Therefore the size of the sample $|N_i|$ must be polynomial in $g_i.k/\epsilon$ and k/δ. Since $g_i \leq 1$, $|N_i|$ is polynomial in k/ϵ and k/δ.

When $A_{class'}$ combines the distributions returned by the k iterations of A_{KL}, there is a probability of at least $1-\delta/2$ that all of the distributions are within $\epsilon(g_i.k)^{-1}$ KL-divergence of the true distributions. We allow a probability of $\delta/2$ that the initial sample N did not contain a good representation of all labels ($\neg\forall i \in \{1, ...k\} : |N_i| \approx g_i.|N|$).

Therefore with probability at least $1-\delta$, all approximated distributions are within $\epsilon(g_i.k)^{-1}$ KL-divergence of the true distributions. If we use the classifier which is optimal on these approximated distributions, f', then the increase in risk associated with using f' instead of the Bayes Optimal Classifier f^*, is at most ϵ. It has been shown that A_{KL} requires a sample of size polynomial in $1/\epsilon$, $1/\delta$ and k. Let $p(1/\epsilon, 1/\delta)$ be an upper bound on the time and sample size used by A_{KL}. It follows that

$$|N| = \sum_{i=1}^{k} |N_i| = \sum_{i=1}^{k} p\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right) \in O\left(k.p\left(\frac{1}{\epsilon}, \frac{1}{\delta}\right)\right).$$

□

2.2.4 Smoothing from L_1 Distance to KL-Divergence

Given a distribution that has accuracy ϵ under the L_1 distance, is there a generic way to "smooth" it so that it has similar accuracy under the KL-divergence? From [13] this can be done for $X = \{0,1\}^n$, if we are interested in algorithms that are polynomial in n in addition to other parameters. Suppose however that the domain is bit strings of unlimited length. Here we give a related but weaker result in terms of bit strings that are used to represent distributions, as opposed to members of the domain. We define class $\mathcal{D}$ of distributions specified by bit strings, such that each member of $\mathcal{D}$ is a distribution on discrete domain X, represented by a discrete probability scale. Let L_D be the length of the bit string describing distribution D. Note that there are at most 2^{L_D} distributions in $\mathcal{D}$ represented by strings of length L_D.

Lemma 22 *Suppose $D \in \mathcal{D}$ is learnable under L_1 distance in time polynomial in δ, ϵ and L_D. Then $\mathcal{D}$ is learnable under KL-divergence, with polynomial sample size.*

Proof: Let D be a member of class $\mathcal{D}$, represented by a bit string of length L_D, and let algorithm A be an algorithm which takes an input set S (where $|S|$ is polynomial in ϵ, δ and L_D) of samples generated i.i.d. from distribution D, and with probability at least $1-\delta$ returns a distribution D_{L_1}, such that $L_1(D, D_{L_1}) \leq \epsilon$.

Let $\xi = \frac{1}{12}\left(\epsilon^2/L_D\right)$. We define algorithm A' such that with probability at least $1-\delta$, A' returns distribution D'_{L_1}, where $L_1(D, D'_{L_1}) \leq \xi$. Algorithm A' runs A with sample S', where $|S'|$ is polynomial in ξ, δ and L_D (and it should be noted that $|S'|$ is polynomial in ϵ, δ and L_D).

We define D_{L_D} to be the unweighted mixture of all distributions in $\mathcal{D}$ represented by length L_D bit strings, $D_{L_D}(x) = 2^{-L_D}\sum_{D\in\mathcal{D}} D(x)$. We now define distribution D'_{KL} such that $D'_{KL}(x) = (1-\xi)D'_{L_1}(x) + \xi.D_{L_D}(x)$.

By the definition of D'_{KL}, $L_1(D'_{L_1}, D'_{KL}) \leq 2\xi$. With probability at least $1-\delta$, $L_1(D, D'_{L1}) \leq \xi$, and therefore with probability at least $1-\delta$, $L_1(D, D'_{KL}) \leq 3\xi$.

We define $X_< = \{x \in X | D'_{KL}(x) < D(x)\}$. Members of $X_<$ contribute positively to $I(D||D'_{KL})$. Therefore

$$\begin{aligned} I(D||D'_{KL}) &\leq \textstyle\sum_{x\in X_<} D(x)\log\left(\frac{D(x)}{D'_{KL}(x)}\right) \\ &= \textstyle\sum_{x\in X_<}(D(x) - D'_{KL}(x))\log\left(\frac{D(x)}{D'_{KL}(x)}\right) \\ &\quad + \textstyle\sum_{x\in X_<} D'_{KL}(x)\log\left(\frac{D(x)}{D'_{KL}(x)}\right). \end{aligned} \tag{2.7}$$

We have shown that $L_1(D, D'_{KL}) \leq 3\xi$, so $\sum_{x\in X_<}(D(x) - D'_{KL}(x)) \leq 3\xi$. Analysing the first term in Equation 2.7,

$$\sum_{x\in X_<}(D(x) - D'_{KL}(x))\log\left(\frac{D(x)}{D'_{KL}(x)}\right) \leq 3\xi \max_{x\in X_<}\left\{\log\left(\frac{D(x)}{D'_{KL}(x)}\right)\right\}.$$

Note that for all $x \in X$, $D'_{KL}(x) \geq \xi.2^{-L_D}$. It follows that

$$\max_{x \in X_<} \left\{ \log \left(\frac{D(x)}{D'_{KL}(x)} \right) \right\} \leq \log(2^{L_D}/\xi) = L_D - \log(\xi).$$

Examining the second term in Equation 2.7,

$$\sum_{x \in X_<} D'_{KL}(x) \log \left(\frac{D(x)}{D'_{KL}(x)} \right) = \sum_{x \in X_<} D'_{KL}(x) \log \left(\frac{D'_{KL}(x) + h_x}{D'_{KL}(x)} \right),$$

where $h_x = D(x) - D'_{KL}(x)$, which is a positive quantity for all $x \in X_<$. Due to the concavity of the logarithm function, it follows that

$$\begin{aligned} \textstyle\sum_{x \in X_<} D'_{KL}(x) \log \left(\frac{D'_{KL}(x)+h_x}{D'_{KL}(x)} \right) &\leq \textstyle\sum_{x \in X_<} D'_{KL}(x) h_x \left[\frac{d}{dy}(\log(y)) \right]_{y = D'_{KL}(x)} \\ &= \textstyle\sum_{x \in X_<} h_x \leq 3\xi. \end{aligned}$$

Therefore, $I(D||D'_{KL}) \leq 3\xi(1 + L_D - \log(\xi))$. For values of $\xi \leq \frac{1}{12}\left(\epsilon^2/L_D\right)$, it can be seen that $I(D||D'_{KL}) \leq \epsilon$. □

We have shown a close relationship between the error of an estimated input distribution (as measured by L_1 distance or KL-divergence) and the error rate of the resulting classifier. In situations where we believe that input distributions may be accurately estimated, the resulting information about the data may be more useful than just a near-optimal classifier.

Chapter 3

Optical Digit Recognition

Much of the research in the field of pattern recognition focuses on bounding the confidence and accuracy parameters of algorithms within learning frameworks such as the variants of the PAC framework. These frameworks are typically abstracted from the reality of practical recognition problems and tend to rely on restrictive conditions being adhered to, such as the PAC restriction that the algorithm must work for a distribution over observations selected by an adversary. In practice, there is generally some indication of the kind of distribution that will be encountered, with distributions over observations from a class typically being smooth to some degree. In order to demonstrate that the method of solving classification problems using unsupervised learners is a viable method in practice, we apply this method to a practical problem – that of optical character recognition (OCR).

The premise behind OCR is to compute labels corresponding to images of characters. There are three main stages involved in this process. Firstly a device must convert the sheet of text into an image – this may be a digital camera or scanner, for instance. The image may then undergo preprocessing in order to convert the image to a state in which it can most easily be identified. Finally the image is processed by a recognition algorithm which outputs the corresponding label or sequence of labels.

This chapter looks at the problem of processing images of handwritten digits and outputting corresponding labels or sequences of labels, and does not look at the stage of creating the initial images or that of preprocessing. The initial stages are image processing issues and have been widely researched for use in OCR software. The techniques involved are relevant to applications such as medical imaging, and are not specific to character recognition. However, the algorithms used to identify the labels from the images are what we are concerned with here as this is where we can implement the technique of using unsupervised learners to learn the class distributions. The results of others (as covered in later results sections) show that by applying preprocessing to

the images, the error rates can be roughly halved.

The topic of digit recognition is appealing for a number of reasons, the most important of which is that image data is readily available. In addition to this the field has a wealth of results with which ours can be compared, without the problem having been "solved". The accurate recognition of most forms of printed script is now considered to be a solved problem, although the problems of recognising handwritten text and particularly the recognition of such text in real time is an actively researched topic.

Although we are examining digit recognition the algorithms used can be applied to a wide range of other problems – a straightforward extension would be to any arbitrary alphabet, although the ideas behind the algorithms could be used for many multi-class classification problems. In most pattern recognition problems, knowledge specific to the problem can be used to gain a great advantage in recognition success rates. For example, bar codes obey a checksum[1] property in order to test whether the data from the bar code has been correctly identified by the scanner. If the code has been incorrectly read, there is a high probability that this will be identified by the algorithms used and as such the classification of the code is declared null.

In the testing carried out, we sample not only single digits but also strings of digits which obey some contextual rule – a checksum rule for example. This allows us to see the benefit of retaining the additional information of the a posteriori probabilities by using unsupervised learners to solve classification problems as opposed to alternative techniques such as the k-nearest neighbour algorithm (as described below).

3.1 Digit Recognition Algorithms

3.1.1 Image Data

The tests performed on the algorithms utilised two sets of data (available for download from [11]) – a training set consisting of sixty thousand images, and a test set consisting of ten thousand images. Each image is represented by a 28 by 28 array of bytes, with each byte representing a colour in grey scale – 0 being white, and 255 being black. The images depict black digits on a white background, and the images have been enlarged and centred from their original 20 by 20 binary pixel format. Some degree of grey scale has been introduced by the normalising algorithm as it performed this enlargement. Figure 3.1 shows three of the digits from the set of training images.

The images are of handwritten digits collected from Census Bureau employees and high-school students. The sets of people writing the digits for the training data

[1] Checksum rules are a form of redundancy check which guard against the misreading of data. When a checksum rule is referred to in this chapter, it means that the sum of the string of digits must divide by some integer with zero remainder (where the integer is context specific).

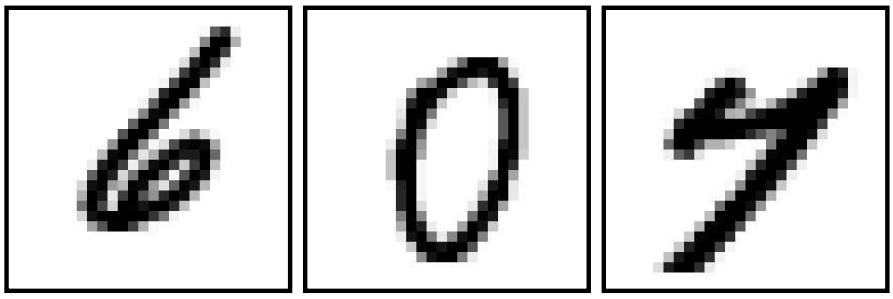


Figure 3.1: Images 1000-1002 in Training set, with respective labels 6, 0 and 7.

and test data sets are known to be disjoint, with half of the digits in each set coming from Census Bureau employees and half from the high-school students. The sets do not contain an equal proportion of images with each label – the precise numbers of each digit can be seen in Table A.1 of Appendix A. It should be noted that some underlying degree of error should be expected when classifying the images, as some images in the data set are almost impossible to label on inspection – an algorithm cannot be expected to label a 7 correctly when the image of the digit looks more like a 1 than a 7. This is a problem inherent in any pattern recognition system. Three examples of such confusing images (taken from the set of Test data – therefore not seen in the learning sample) are shown in Figure 3.2.

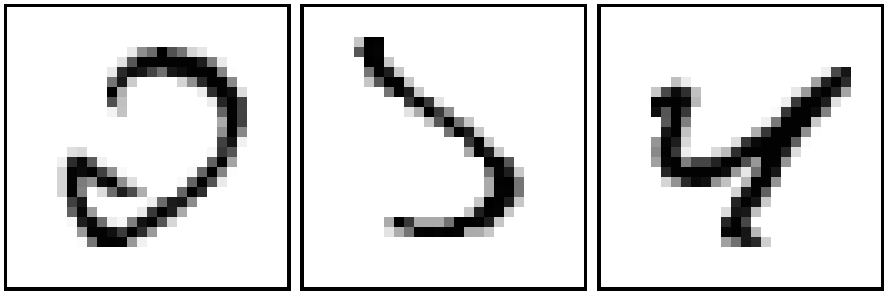


Figure 3.2: Images 2098, 1393 and 2074 in Test set, with respective labels 2, 5 and 4.

For further detail of the data sets and basic pre-processing performed on the images, see [11]. Here we are not concerned with pre-processing techniques, as these will be specific to the particular demands of OCR, and this is a general case study into the performance of algorithms based on the use of unsupervised learners to perform general classification tasks – not necessarily specific to OCR.

3.1.2 Measuring Image Proximity

Two measurements of the "closeness" of images are used – the L_2 distance and the complete Hausdorff distance.

L_2 distance

When we talk about the L_2 distance between two images, we treat the images as vectors of bytes where each pixel is represented in one dimension. Therefore we compute the distance between two 28^2 dimensional vectors. It is generally good practice in image processing to reduce the dimensionality of the data through feature selection or extraction before making comparisons between samples as this reduces the impact of anomalies in the data and may allow for patterns in the data to be better distinguished[2]. However in this case it was decided that dimension reduction is unnecessary since it is just another form of preprocessing and we are more interested in observing the performance benefits when introducing contextual information over the performance on single digit recognition, than in trying to optimise the performance on single digits.

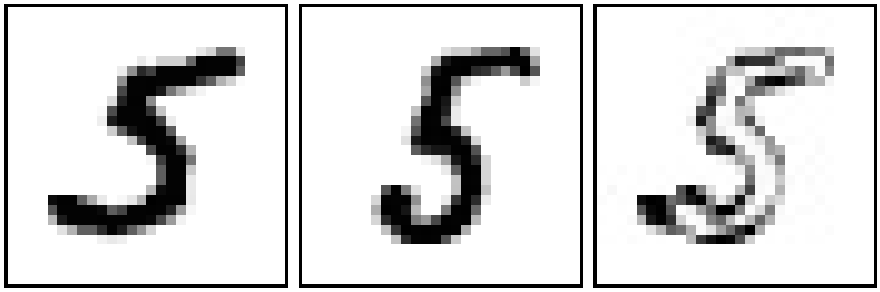


Figure 3.3: L_2 distance between two images with label 5.

The L_2 distance between two images can be illustrated by superimposing one image over the other, and taking the difference of the shades at each pixel. Figure 3.3 shows the difference between two images of the digit 5. The amount of shading on the image represents the distance between the two image vectors.

By contrast, Figure 3.4 shows the distance between images with labels 3 and 9. It is apparent that this image has far more shading than the difference in Figure 3.3 – and as such the distance between the images is far greater.

[2]See [21] for information on dimension reduction techniques.

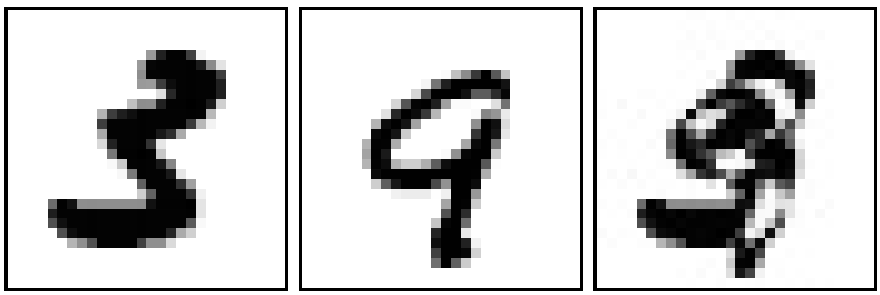


Figure 3.4: L_2 distance between images with labels 3 and 9.

Complete Hausdorff distance

In order to explain the concept of the Hausdorff distance between two images, we must view the images as sets of points, where the sets comprise all shaded elements of each array. In practice the algorithms based on Hausdorff measurements use a threshold value of 5 to convert the bytes into bits - with values greater than 5 being seen as shaded.

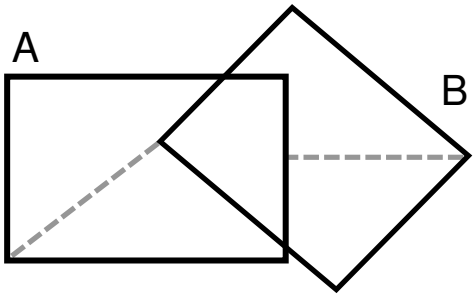


Figure 3.5: Hausdorff Distance.

Figure 3.5 illustrates the Hausdorff distance from A to B, and also from B to A. Formally, the Hausdorff distance from A to B is

$$h(A,B) = \max_{a \in A} \left\{ \min_{b \in B} \{L_2(a,b)\} \right\}.$$

This is shown by the left-most dotted line on the diagram – the point in A furthest from B, to the closest point in B. Similarly, the right-most dotted line shows $h(B,A)$. Notice that $h(A,B) \neq h(B,A)$ in this instance, and as such the Hausdorff distance is not a metric. However, we shall use the complete Hausdorff distance, $H(A,B) = \max\{h(A,B), h(B,A)\}$, which is a metric due to its symmetry. Note that the algorithms view the images as *sets of points* rather than geometric shapes, but for simplicity the illustration shows rectangles rather than sets of individual pixels.

3.1.3 k-Nearest Neighbours Algorithm

A standard algorithm for performing digit recognition – and one for which results are known (see [34], for example) – is the k-nearest neighbour algorithm. This simple algorithm performs reasonably well in many pattern classification settings and has the beneficial property that no training time is required as the algorithm compares the image to be classified with each image in the training set at run-time. Note that this is of no particular significance in this setting, as the algorithms have all been designed to work in this way[3].

Figure 3.6 shows the k-nearest neighbours algorithm. The algorithm finds the k images in the set of training data lying closest (in terms of L_2 distance) to image I – the image to be classified. The labels and proximities of the k images closest to I from the images observed to date are recorded in array N. Once all of the training data has been examined, the labels of these k images are then compared and the predominant label is chosen to label the unclassified image. In the case where the k-nearest neighbours include equal numbers of two or more different labels, one can be arbitrarily chosen (we choose the lowest label).

The algorithm takes k as an input, and an image from the set of test data. The algorithm has access to the set of sixty thousand training images, as well as the corresponding labels. The distance $L_2(a, b)$ between two images is simply the Euclidean distance between the two vectors in n dimensional space – where the dimension of the space is equal to the number of pixels in the images – such that

$$L_2(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \ .$$

3.1.4 Unsupervised Learners Algorithms

The method of using unsupervised learning agents to classify images requires the training data set to be partitioned into ten subsets – one for each label – such that the subset corresponding to label ℓ contains every image in the training data set with label ℓ. Each subset is passed to one of ten unsupervised learning agents. The agent uses a kernel based density estimation algorithm to estimate the probability density function over all images in its subset.

Kernel density estimation algorithms are a commonly used non-parametric technique for approximating distributions (see [17]). The kernel used by our algorithm is a normal distribution with a standard deviation specified by the user, centred at the

[3] To achieve a reduction in complexity when using the method involving unsupervised learners, it is necessary to perform a training pass over the data to estimate the a posteriori probabilities prior to performing recognition.

Algorithm 1

```
input: k, image I

let N[ ] be an array of k pairs (x_i, ℓ_i)
for i = 0 to k − 1
  N[i] = (∞, 0)

for each image J in the training set (with label ℓ_J)
  d = L_2(I, J)
  if d < x_{k−1}
    N[k − 1] = (d, ℓ_J)
    Sort N by increasing values of x

let ℓ be the label occurring with greatest frequency in N
(or the lowest such value)

return ℓ
```

Figure 3.6: k Nearest Neighbours technique using L_2 distance metric.

coordinate represented by the image in 28^2 dimensional vector space. The learner constructs a distribution over the data set by summing all of these normal distributions, such that the value at any point in the domain receives at least some small contribution from each of the distributions, but a far greater contribution from those centred nearby. This distribution is then scaled by the number of samples in the subset to achieve the probability density distribution.

Once we have ten such distributions we can classify an image by passing its vector to the learners, each of which returns the probability density for its associated label at that coordinate. The algorithm then selects the label with the highest corresponding probability density, and an estimate of the "likelihood" (or the a posteriori probability) attributed to this label can be calculated by normalising these probability densities. Figure 3.7 shows this algorithm.

Note that σ is the standard deviation. Note also that although the algorithm returns only the estimation of the label corresponding to the image being classified, the values of $Z[\,]$ represent estimates of the a posteriori probabilities of the labels 0 through

9. This algorithm uses a function, distance(a, b), to gauge the distance between two images. As previously mentioned, both the L_2 distance and the Complete Hausdorff distance were used in the experiments, and as such the "distance" function can refer to either measurement as required. Note that the relevant functions for these two measurements are provided in Section A.1 of Appendix A.

For the purpose of this work it is unnecessary to explicitly estimate the ten individual probability density functions – rather we can compare the image being classified with each image in the training data as we did in Figure 3.6. The distance between the pairs is then used to calculate the a posteriori probabilities using normal distributions with the standard deviation value declared as a parameter. This simplification will give identical results and is more convenient at this scale. If using this technique on larger amounts of data then for a finite domain (which invariably OCR is confined to) it is possible to make a pass through the training set and create the distributions representing each digit, and this would considerably speed up the classification process[4].

3.1.5 Results

During testing the algorithms were run on all images in the set of test data, and were trained on all images in the set of training data.

k-nearest neighbours

The tables in Section A.2.1 (Appendix A) show the true labels of the test digits in the rows, and the columns represent the output classifications per thousand tests. As one might expect, the diagonals corresponding to correct classifications (where an image with label i is classified with label i) contain the highest values.

Tests were conducted using values of k equal to $1, 3$ and 5, and the corresponding results can be found in Tables A.2, A.3 and A.4. A summary of the results is given in Table 3.1.

The results show that the nearest neighbour algorithm gives quite reasonably accurate labels considering its simplicity, attaining an error rate just below 4%.

Unsupervised learners

In order to make a direct comparison, the same tests were conducted (using L_2 distance) as for the k-nearest neighbours algorithm above, with different values being substituted

[4] An important advantage of using this generative approach, where the distributions are pre-computed, is that the set of classes being modeled can be augmented or diminished relatively easily. To remove a class label the algorithm can simply ignore the corresponding distribution for that label, and to add a symbol to the alphabet of labels, the algorithm can make a pass through the training data with that label and construct a corresponding distribution – leaving the other distributions unchanged.

Algorithm 2

```
input:  image I

let Z[ ] be an array of ten real numbers
for i = 0 to 9
  Z[i] = 0

for each image J in the training set (with label ℓ_J)
  d = distance(I, J)
  Z[ℓ_J] = Z[ℓ_J] + (σ√(2π))^{-1} exp(-d²/(2σ²))

for i = 0 to 9
  let n_i be the number of images in the training set with label i
  Z[i] = Z[i]/n_i

normalise elements of Z[ ] such that Σ_{i=0}^{9} Z[i] = 1

return arg max_i {Z[i]}
```

Figure 3.7: Algorithm to classify images of digits using a normal distribution as a Kernel.

for the standard deviation of the normal distribution. The results in Section A.2.2 (summarised in Table 3.2) show that for standard deviations of 1000, 2000 and 4000, the overall misclassification rates were 3.7%, 3.8% and 5.7% error rates. Referring back to the results for the k-nearest neighbours algorithms, we see that the error rates for the algorithms using standard deviations of 1000 and 2000 are virtually identical, which illustrates that the algorithm using unsupervised learners has the potential to work at least as well as the more commonly used k-nearest neighbours algorithm.

Tables A.8, A.9 and A.10 show the average likelihoods associated with each of the ten labels (the a posteriori distributions) when classifying images of each digit in the data set. In other words, the value u_{ij} is the average a posteriori probability (over the test data set) assigned to label j (represented by $Z[j]$ in the algorithm) having been given an image of digit i as input.

These tables also state the average negative log-likelihood associated with the algorithm for each value of σ. The negative log-likelihood for any given input is the

k	Average misclassification error per 1000 images
1	39
3	37
5	38

Table 3.1: Results of Nearest Neighbour algorithm.

Standard Deviation	Average misclassification error per 1000 images	Average Negative Log-Likelihood
1000	37	0.743
2000	38	0.203
4000	57	0.188

Table 3.2: Results of Unsupervised Learners algorithm (using by L_2 distance).

negative log of the a posteriori probability assigned to the correct label of the input image. For instance, if an algorithm classifies an image of the digit 7 correctly, let's say with an estimated a posteriori probability of 0.9, then the negative log of 0.9 is the negative log-likelihood of the correct label (0.046 – a small value). However, if the algorithm classified the image as a 1, then the estimated a posteriori probability of a 7 is likely to be much lower, say 0.4, in which case the negative log-likelihood of the correct label is the negative log of 0.4 (0.444 – a far larger value). It follows that the average negative log-likelihood associated with an algorithm is an assessment of how certain the classifications are. If the value is low, then it suggests that the algorithm consistently assigns a reasonably large probability to the correct label.

Table 3.2 gives the average negative log-likelihoods corresponding to each of the standard deviations tested. Even though the error rate is lower when using a smaller standard deviation, the negative log-likelihood is substantially greater. This suggests that the greater smoothing effect of using normal distributions with a large standard deviation as a kernel allows for an averaging to occur when an image is on the borderline between two possible labels. If an image of a 7 looks quite like an image of a 1, then a large standard deviation means that images with label 7 and 1 from the training set will have a more balanced influence on the relative probabilities than if the standard deviation is lower, in which case only the images in close proximity will have much effect – possibly all 1s in this case. As a result, although the use of a low standard deviation scores a lower error rate, the algorithm can be mislead by erroneous images

due to this overfitting. In contrast to this, the higher standard deviation classifies fewer images correctly, but in the case of a misclassification, a reasonably high probability is still assigned to the correct label.

Tests were performed on Algorithm 2 using the Hausdorff distance rather than L_2 distance. Four different values were used for the standard deviation of the kernels, and the average misclassification rate and average negative log-likelihood of the correct labels are recorded in Table 3.3.

Standard Deviation	Average misclassification error per 1000 images	Average Negative Log-Likelihood
0.4	39.5	0.198
0.45	40.4	0.171
0.5	40.7	0.154
1.0	78.2	0.303

Table 3.3: Results of Unsupervised Learners algorithm (using Hausdorff distance).

It can be seen that the performance of the algorithm using the complete Hausdorff metric, in terms of misclassification errors, is comparable to the algorithms observed so far. The algorithm can achieve error rates as low as 4.0% – which is only 0.3% higher than the best rates observed when using L_2 distance or the k-nearest neighbours algorithm as seen above.

The advantage of using the complete Hausdorff distance is seen in the figures for the average negative log-likelihood. When comparing the two measurements, it appears that the algorithm using the complete Hausdorff distance and normal distributions with standard deviations of 0.5 performs the best in terms of negative log-likelihood, and also have relatively low error rates, making them a good proposition for use in a context sensitive setting as described in Section 3.2.

Conclusions

In order to assess the significance of these results we look at the reasons why the algorithms misclassify images, and examine the results in the context of other known results.

As mentioned above, there is an underlying error rate inherent in the data set due to the handwriting seen in some of the images. Examples of the vagueness of images can be seen in Figure 3.2. These images may be so far removed from any of the images in the training data that it is difficult to see how any algorithm could classify them – other than perhaps more advanced algorithms using feature extraction to recognise the

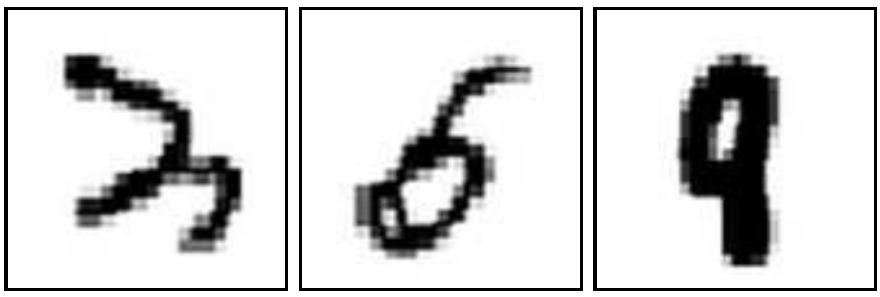


Figure 3.8: Images in Training set, with respective labels 3, 5 and 8.

construction of the handwritten digits.

Aside from these erroneous images, there are images which are ambiguous due to their similarity to other digits. Figure 3.8 shows three examples of misleading images. As these three images appear in the training data set, they can cause problems with the recognition of other digits. For example, the image of an 8 looks remarkably like a 9, and it is likely to have a very small L_2 and complete Hausdorff distance to many images with label 9. When studying the tables of results showing the classifications made by the algorithms, it is clear that there are pairs of digits which are commonly mistaken for each other. In the examples shown, the digits are written poorly and do not look how they are supposed to, but in fact there are inherent similarities between several pairs of digits – 4s and 9s for example can be written in a similar way, as can 1s and 7s, and in some cases 2s and 7s.

Comparing the results in this section with results published elsewhere, we see that our results fit in roughly as expected. Lecun et al. ([34]) give the error rates associated with a number of different learning algorithms including a k-nearest neighbour algorithm and various neural networks. The nearest neighbour algorithm in their paper achieves a 5% error rate, which is slightly higher than the rate achieved here. By preprocessing the images using a deslanting technique, the same algorithm achieves a rate of just 2.4% – less than half of the initial rate. According to [11], Kenneth Wilder at the University of Chicago achieved an error rate of 3.09% using a k-nearest neighbours algorithm based on L_2 distance, and this was further reduced to 1.8% after preprocessing the images (by de-skewing, blurring, and performing some form of noise removal).

The best result achieved in [34] was an error rate of 0.7% by a boosted neural network performing on a data set augmented with artificially distorted samples. Since then, a rate of 0.4% has been achieved on the data set by Simard et al. in [44], using a technique known as convolution nets, with much preprocessing performed on the data.

This shows that by performing preprocessing on the data before applying the algorithms, far lower error rates can be achieved. However, we are not concerned with

achieving low error rates using these simplistic algorithms – they are merely a benchmark to show the advantages of using the unsupervised learners in context sensitive settings as illustrated in the following sections.

3.2 Context Sensitivity

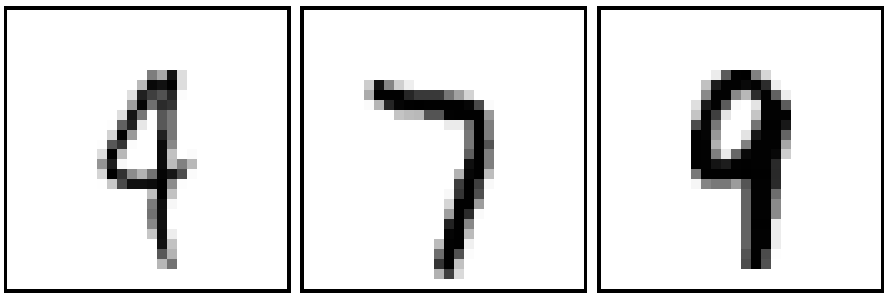
When talking about context sensitivity, we mean any setting making use of the additional information retained by a generative algorithm over a discriminative one – the a posteriori probabilities of each label for a given image. In this section we study an algorithm that classifies strings of digits rather than individual digits. In each case there is some predefined contextual rule defining the subset of strings from which the target string is drawn. An example of this is a checksum rule where the string of digits must sum to a multiple of ten.

If it is known that the string of digits belongs to a specific subset of possible strings, then once the initial recognition has taken place for each of the images it may be apparent that a mistake has been made if the string of corresponding labels does not belong to the subset. In this case it is beneficial for the algorithm to have some mechanism for backtracking and finding the likely root of the error. For this to happen the algorithm needs to have some measure of certainty linked to each classification. In the case of the unsupervised learners algorithm in the previous section, this measure of certainty is the a posteriori probability of the label chosen. Rather than discarding the a posteriori probabilities of each label, which have been calculated by the algorithm, they can be used to judge whether a mistake may have occurred in the recognition process.

As an example, imagine that the algorithm in Figure 3.7 is classifying the first of the three digits in Figure 3.8, given access to some arbitrary training set. Consider that for the labels from 0 to 10, the corresponding estimated a posteriori probabilities are 0.01, 0.03, 0.50, 0.37, 0.01, 0.02, 0.01, 0.01, 0.03, and 0.01. The algorithm has assigned the highest likelihood to label 2, and as such the image is given label 2. However – if we are now told that the label cannot be 2, we have a record of the likelihoods associated with each of the other labels, so we can simply find the label with the highest probability from the subset of labels which are allowed. In this case, the next highest likelihood is associated with the label 3.

Note that discriminative algorithms do not have estimates of the estimated a posteriori probabilities associated with each label, and would therefore need to be artificially altered in some way as to choose an alternative label.

Now let us examine a situation where rather than identifying a single image, we are now presented with a string of three digits. Figure 3.9 shows three images, as well as a possible set of likelihoods generated by an algorithm.



Three Most Likely Labels (with associated likelihoods)		
9 (0.53)	7 (0.84)	9 (0.76)
4 (0.42)	2 (0.08)	4 (0.19)
1 (0.03)	9 (0.02)	7 (0.03)

Figure 3.9: Images 1242, 4028 and 4009 in Test set, with respective labels 4, 7 and 9.

If we take the label with the highest likelihood for each of the images, we output the string 979. Now if we introduce a contextual rule to define the subset of strings to which these images must belong, we can analyse the way in which an algorithm might correct itself. Consider the rule where the digits of the string must sum to a multiple of ten. If this is the case, then the string 979 cannot be correct, as these digits sum to 25.

In order to find the most likely string belonging to the subset of strings which fit the rule, we can consider each string in descending value of likelihood until a string belonging to the subset is found. The likelihood associated with a string is the product of the likelihoods assigned to each individual digit in the string. In this case, the first few strings to consider in order of likelihood are 979 (0.338), 479 (0.268), 974 (0.085), and 474 (0.067). Of these strings, 479 and 974 both fit the rule (their digits sum to a multiple of ten), so we output 479 as this is the string with the highest associated likelihood which also fits the rule.

If a discriminative algorithm was faced with the same problem, then it is likely that a correct classification would not be made. One way of attempting to correct the problem would be to pick one of the three digits at random and alter the label of that digit in such a way as to make the three digits sum to a multiple of ten. In the example given, this would result in a classification of either 479, 929, or 974 – leading to a one in three chance of correcting the initial mistake accurately.

For this type of context sensitive classification task it can be beneficial to use an algorithm which sacrifices some degree of classification success rate in favour of having a lower negative log-likelihood. This results in smoother distributions being created by the unsupervised learners, which are less prone to overfitting[5].

[5]Overfitting is a problem commonly associated with machine learning, where the predictive model is

We study the application of the unsupervised learners method to three context sensitive learning problems. As suggested above, the complete Hausdorff distance is the better of the two metrics to use judging from the results in Section A.2.2, given that it demonstrates a relatively low negative log-likelihood.

The algorithm in Figure 3.10 takes n images of digits, and returns the string of digits most likely to be represented by those images – subject to the string fitting a specific contextual rule. The distance function can be whichever measurement is required, as in previous algorithms, but in this section the Hausdorff distance has been used as explained above. Three context rules were used and the details of each are in the following sections. To understand the working of the algorithm it is sufficient to know that rule(s) returns a boolean value determined by whether string s belongs to the subset of all possible strings as defined by the rule in question.

Testing procedure

It should be noted that the testing procedure used in this section was as follows. All tests were performed on randomly chosen images from the set of all images with appropriate labels in the test data set. For example – if a test was carried out on a string of three digits summing to a multiple of five, then the first two digits were selected randomly from the set of ten thousand test digits. After this, the third image was randomly selected from the remaining subset of available images. So if, for instance, a 5 and an 8 had been chosen (summing to 13), then the final digit was selected at random from the set of all digits with label 2 or label 7 in the test data set.

When carrying out tests with a specified set of parameters, each test was generated independently. If 300 tests were performed on strings of three digits, with a specified standard deviation, then each string of digits was generated at the time of the test. As such, it is possible (but highly unlikely) that all 300 tests could be performed on not only the same string of digits, but on the same images representing those digits. I have not calculated the probability of such an event occurring but needless to say it is not substantial enough to adversely affect the results.

As the testing was carried out, the parameters used were adjusted according to patterns in the results – the standard deviation values used were not predetermined. As a result, the intervals between test sets are not always regular, but this was necessary to get both a decent span of values in order to see overall patterns in the results, as well as trying to home in on the best classification rates and correction rates.

fitted too closely to the training data resulting in the model being tailored to this specific data set and any random features of this data.

Algorithm 3

input: images $I_0, \ldots, I_{n-1}$

let p_ℓ^x represent the likelihood of digit x having label ℓ
for each $\ell \in \{0, ..., 9\}, x \in \{0, ..., n-1\}$
  $p_\ell^x = 0$

for $x = 0$ to $n-1$
  for each image J in the training set (with label ℓ_J)
    $d =$ distance(I_x, J)
    $p_{\ell_J}^x = p_{\ell_J}^x + (\sigma\sqrt{2\pi})^{-1} \exp(-\frac{d^2}{2\sigma^2})$

  for $i = 0$ to 9
    let n_i be the number of images in the training set with label i
    $p_i^x = p_i^x / n_i$

  normalise elements p_ℓ^x such that $\sum_{i=0}^{9} p_i^x = 1$

let s^* represent the most likely string of digit labels and
l_{s^*} represent its associated likelihood
for $d_0 = 0$ to 9
  ...
    for $d_{n-1} = 0$ to 9
      let s be the string of digits $d_0 d_1 ... d_{n-1}$
      let $l_s = \Pi_{x=0}^{n-1} p_{d_x}^x$
      if$((l_s > l_{s^*}) \bigwedge$ rule$(s))$
        $s^* = s$ and $l_{s^*} = l_s$
return s^*

Figure 3.10: Algorithm to recognise n-digit strings obeying a contextual rule.

3.2.1 Three-Digit Strings Summing to a Multiple of Five

The first test involves strings of three digits, which must sum to a multiple of five. Formally, the digits d_0, d_1, d_2 must obey the following rule:

$$\exists k \in \mathbb{N} \cup \{0\};\ \sum_{i=0}^{2} d_i = 5k.$$

To analyse the performance of the algorithm, three hundred tests were performed using each of six values of standard deviation: $0.5, 0.6, 0.7, 0.8, 0.9$ and 1.0.

Standard Deviation	No. of Tests	Strings Recognised	%	Strings Corrected	%	Correct Classifications	%
0.5	300	266	88.7	19	55.9	285	95.0
0.6	300	258	86.0	26	61.9	284	94.7
0.7	300	247	82.3	34	64.2	281	93.7
0.8	300	255	86.3	31	68.9	286	95.3
0.9	300	250	83.3	32	64.0	282	94.0
1.0	300	229	76.3	47	66.2	276	92.0

Table 3.4: Results of classifying three-digit strings summing to a multiple of five.

The results can be viewed in Table 3.4, correct to 3 significant figures. A *string recognition* success is achieved when the algorithm associates the highest likelihood with the correct label for each of the three digits – therefore suggesting the correct labels and as a result producing a correct classification. A *string correction* is achieved when the algorithm has failed to recognise at least one of the digits, but then due to the resulting 3 digits not fitting the checksum rule the algorithm finds the error(s) and classifies the string correctly. Note that the percentage for string corrections is calculated as a proportion of those strings which were originally recognised incorrectly by the algorithm. Finally, the *correct classification* column records the total number of strings which were either recognised correctly at the outset, or were corrected accurately by the algorithm – thereby classifying the string correctly.

The table of results shows three records – the number and percentage of strings correctly *recognised* by the algorithm, the number and percentage of strings *corrected*, and the number and percentage of strings correctly *classified*. The algorithm is said to have *recognised* a string correctly if the correct labels of the three digits each received the highest estimated a posteriori probability – these are the strings that would have been classified correctly without the knowledge of the contextual information that the digits must sum to a multiple of 5. The string is *corrected* by the algorithm if the initial

recognition of the string is incorrect (the estimated a posteriori probabilities indicate an incorrect label for at least one of the three digits), but the algorithm returns the correct string having used the checksum rule to show that an error has been made and found the most likely string of digits to fit the rule. Finally, a string is correctly *classified* so long as the right digits are returned by the algorithm, regardless of how these digits were arrived at.

It is reasonable to expect that a lower standard deviation would result in a higher rate of recognition based on the findings of the tests on single digits, and indeed this appears to fit the overall trend shown in the results. It is also intuitive to assume that perhaps the average negative log-likelihood of the correct label has some bearing on the percentage of strings corrected by the algorithm. However, the results show that the percentage of strings corrected is higher when using a standard deviation of 1.0 than it is when using 0.5, despite Table 3.3 showing that the corresponding average negative log-likelihoods being 0.303 and 0.154. This may be an anomaly in the results – although there is an upward trend in the percentage of corrected strings as the standard deviation rises, there is a local maxima at $\sigma = 0.8$, and the result for $\sigma = 1.0$ may be misleadingly high. Looking at Table 3.3 it appears that the average negative log-likelihood is likely to be at its lowest somewhere between $\sigma = 0.5$ and $\sigma = 1.0$, so it could be expected that the percentage of strings corrected should peak between these values.

The overall classification rates show that the optimum classification rate occurs at $\sigma = 0.8$ which seems to be the point at which the increased correction rate begins to outweigh the drop in recognition rate as the standard deviation increases.

Interpretation of results

It would be misleading to draw any firm conclusions from this testing since the tests were performed on randomly selected images from the data set, and some images are harder to identify than others. Another feature of this particular test is the nature of the checksum rule – three digits must sum to a multiple of five. This contextual knowledge will not help the algorithm to correct mistakes when an image of a 2 is mistaken for 7 or vice versa. Similarly, 4s and 9s, and 8s and 3s can be mislabeled without the rule spotting that a mistake in recognition has occurred. This is demonstrated in Figure 3.11, where the third digit could be classified as a 4 or a 9 and the sum of the digits would be either 15 or 20 – both accepted values as they fit the rule. By examining the tables in Section A.2.2 it can be seen that these are all common mislabellings for the algorithm using L_2 distance, and it is likely to apply in the same way to the algorithm using the complete Hausdorff distance.

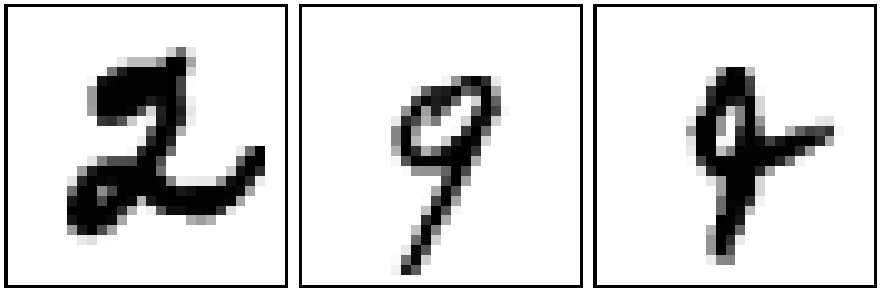


Figure 3.11: Images 5037, 4016 and 4017 in Test set, with respective labels 2, 9 and 4.

3.2.2 Six-Digit Strings Summing to a Multiple of Ten

This test involves strings of six digits, which must sum to a multiple of ten, such that

$$\exists k \in \mathbb{N} \cup \{0\};\ \sum_{i=0}^{5} d_i = 10k.$$

With a longer string there is likely to be a higher error rate in recognising the images – it is less likely that the algorithm will assign the highest a posteriori probabilities to the correct labels for six out of six images than in three out of three. This is borne out by the figures in Table 3.5, which show that the rate of recognition is about 10% lower on average than the results for tests using equivalent standard deviations in Table 3.4. However, the results also show a marked improvement in the correction ability of the algorithm when used on six digits as opposed to three. Correction rates as high as 80.7% were observed – on average roughly 15% higher than the results for three digit strings.

Standard Deviation	No. of Tests	Strings Recognised	%	Strings Corrected	%	Correct Classifications	%
0.4	250	198	79.2	35	67.3	233	93.2
0.5	250	186	74.4	47	69.1	233	93.2
0.55	250	193	77.2	46	80.7	239	95.6
0.6	250	178	71.2	55	76.4	233	93.2
0.7	250	181	72.4	54	78.3	235	94.0

Table 3.5: Results of classifying six-digit strings summing to a multiple of ten.

Interpretation of results

Again there seems to be a peak in standard deviation at which the correction process is performed optimally. For lower values it is likely that the peaks of the normal distri-

butions are narrow, leading to only the nearby images being influenced and this in turn leads to a lack of useful information when a choice must be made as to which digit to correct, and the additional decision on how it should be corrected. In the case of a higher standard deviation, the smoothing of the distributions may be too great, leading to the estimated a posteriori probabilities being closer together, and being overly affected by misleading images in the training data. This is fine when only one digit has been incorrectly recognised, since it is likely that the correction can be made, but if more than one digit needs correcting there may be several ways to change the digits whilst still obeying the checksum rule.

For instance, imagine the case where the string 477039 has been recognised as 127034. Three digits have been recognised incorrectly in this instance. It happens to be the case that if the initial 1 is corrected to a 4, the string 427034 obeys the rule that the digits must sum to a multiple of ten, so it is quite possible that the algorithm will give a higher probability to the string being 427034 than 477039, and as a result the string is misclassified.

3.2.3 Dictionary of Eight-Digit Strings

Having looked at strings obeying a checksum rule, we now examine a similar setting in which a random string is selected from a set of ten thousand strings of eight digits, and eight images with labels corresponding to these digits are passed to the algorithm for classification. The algorithm has access to the "dictionary" of ten thousand strings, and aims to return the appropriate string from the dictionary corresponding to the sequence of strings passed to it.

In each test a string was selected at random from the dictionary and for each digit in the string an image with the corresponding label was selected at random from the set of test data. The standard deviation used in the testing was 0.55, as this gave the best performance on the classification of 6-digit strings – in particular it produced the highest correct rate for strings recognised incorrectly. It should be noted that the strings in the dictionary are all different, and were generated at random from a uniform distribution over the set of all eight digit strings.

Standard Deviation	No. of Tests	Strings Recognised	Strings Corrected	%	Correct Classifications
0.55	10000	5801	4194	99.9	9995

Table 3.6: Results of classifying eight-digit strings belonging to a dictionary of ten thousand strings.

The results from the tests are displayed in Table 3.6. Of the ten thousand tests, 5801 of the strings were correctly recognised. This result is rather low, and equates to a recognition rate of 0.934 for a single digit, to 3 decimal places. However, the algorithm corrected 4194 of the 4199 strings incorrectly recognised (roughly 99.9%), giving an overall classification rate of 99.95%.

Interpretation of results

This result shows the power of the unsupervised learners algorithm in being able to correct misclassifications in a setting where the data is drawn from a relatively sparse set. The set itself contains ten thousand strings, out of a domain of 10^8 possible strings. In this setting it is reasonable to assume that a discriminative algorithm could be altered in some way so as to find the "closest" string in the dictionary to the string recognised, if this string is not itself in the dictionary. For instance, where only a single mistake has been made in the recognition of a string, it is likely that only one string in the dictionary is reasonably close to the recognised string – so it is not hard to correct the mistake manually.

However, the overall classification rate of 99.95% is equivalent to a recognition rate of at least 99.994% if correction is not taken into account, which shows that the method of using the a posteriori probabilities to predict string classifications is a powerful tool when compared to even the best single digit recognition algorithms.

If we analyse this result further, then the single digit recognition rate of 0.934 can be used to estimate roughly how many strings were recognised with 1 error, how many with 2 errors, etc. Figure 3.7 shows these estimated numbers. Note that the estimated number of strings does not sum to exactly ten thousand due to rounding.

No. of Recognition Errors	Estimated no. of strings
0	5801 *(known)*
1	3269
2	806
3	114
4	10
5	1
6+	0

Table 3.7: Estimated number of recognition errors over ten thousand tests.

As mentioned previously, the domain is quite sparse – containing 10^4 strings out of a possible 10^8 eight-digit strings. This means that if a string has a small number

of digits with recognition errors, there is likely to be only one string in the dictionary near to it in the state space. Therefore in these cases it would be relatively simple to manually correct the recognition errors. With this in mind, it is likely that a discriminative algorithm with a low error rate could achieve good results with a simple adjustment to correct strings with recognition errors.

However, in cases where 3 or more errors occur there may be more difficulty in spotting the mistakes. If 4 or more mistakes have been made in recognition then it is likely that the recognised string lies closer in the state space to some other string in the dictionary, and as a result a misclassification is likely to occur. Given the number of misleading images in the test data set (likely to number around seventy out of ten thousand on anecdotal evidence[6]) it is reasonable to presume that in at least 5% of strings tested, even the most accurate classifier would struggle to classify all eight digits correctly.

3.2.4 Conclusions

The nature of the testing for the context sensitive algorithms means that firm conclusions cannot be drawn from these results in terms of the error rates achieved, but the tests serve a good way of demonstrating the way in which string correction works. The results show that the algorithms using unsupervised learners can obtain good rates of correction, which is the main advantage of this technique over existing techniques. With further structured testing where images are chosen in a more predetermined manner, it would be possible to find out the precise effects of altering the standard deviation on the negative log-likelihood and on the error rates.

As discussed above, it would be possible to extend a discriminative algorithm so that it is able to estimate where errors have occurred in classification and in some way correct these errors. An algorithm such as the k nearest neighbours algorithm could be used with an additional rule to change an arbitrary digit to make the recognised string fit a checksum rule if it does not do so already – but this could only be expected to achieve a $1/n$ success rate when applied to strings of n digits under the condition of digits summing to a multiple of ten, and a one in $1/2n$ success rate when applied to the condition of digits summing to a multiple of five. There are ways of estimating a posteriori probability distributions from algorithms like the k nearest neighbours algorithm, but these can be computationally expensive to use, and seem somewhat artificial in comparison to estimating the true probability distributions of the labels over the domain. Once a discriminative algorithm has been modified in such a way, it is effectively performing the

[6]The best recorded performance of a single digit classifier recorded on [11] was an error rate of 0.7% which leads me to believe the number of misleading digits must be roughly this proportion of the data set.

same task as a generative algorithm so there can be little benefit to pursue this route.

While it is quite possible to imagine a corrective technique being applied to a basically discriminative algorithm when considering a sparse state space, it seems logical that with a more densely populated space, the unsupervised learners algorithm would perform relatively well in comparison. The more densely populated the state space, the less likely it is that an algorithm could correct an incorrect recognition of digits by "guesswork" alone – but having knowledge of the a posteriori probabilities would put the unsupervised learners algorithm at a great advantage.

Chapter 4

Learning Probabilistic Concepts

Probabilistic concept (or p-concept) learning is fundamentally a 2-class classification problem. Unlike traditional PAC-learning problems, the classes involved in the p-concept setting may exhibit some overlap over elements of the domain due to the stochastic nature of the model. In other words, over a domain X each $x \in X$ has some associated (a posteriori) probability distribution over the two class labels. The notion of a probabilistic concept was introduced by Kearns and Schapire in [31], and is defined in Section 1.1.4.

As this is a 2-class probabilistic model, it follows that rather than visualising the distributions over the two classes as separate curves with associated a priori probabilities, we can instead model the a posteriori probabilities of one of the classes from which the other can be inferred, and a probability density function representing the distribution of data over the domain. So, for class labels 0 and 1, the data is distributed over domain X according to D, with function $c(x)$ representing the probability of an observation at $x \in X$ belonging to the class with label 1.

4.1 An Overview of Probabilistic Concepts

In essence, a p-concept represents the probability of an event of "class 1" occurring given observation x. [31] uses examples of predicting whether rainfall will occur tomorrow given observations such as temperature and wind speed, and of whether a student will be admitted to a particular college based on their past record, to illustrate possible uses of the framework. The result of each of these events is basically a boolean function; either it does rain or it doesn't, there is no in between. However, the chance of either event can be modeled using observations, leading to a rule equivalent to $c(x)$ for the observed values x which is a probability of the event occurring.

It is noted that these examples exhibit three particular traits which fit with the p-concept setting. Firstly, there is an element of probabilistic behaviour involved

– whether it be caused by a truly random event, by a deterministic process where observations cannot be measured to a high enough accuracy to predict the outcome, or that insufficient is understood about the process to make an assured prediction either way.

Another important point is that although there may be some underlying probability of either event occurring, only the outcome of one event or the other is observed. Therefore the observer makes no measurement of the probability of that event from a single observation, and it may be the case that each observation x is so rare that no estimate of $c(x)$ can possibly be made no matter how large a sample has been taken.

Finally, it is pointed out that despite the probabilistic nature of the problem, there is an underlying structure to the events being predicted – the event is closely linked to the observations being made – the events are not independent. This is in contrast to models involving noise, where the concept being modeled is in fact deterministic and the problem involves filtering out the noise.

4.1.1 Comparison of Learning Frameworks

The framework for learning probabilistic concepts is similar in many respects to the traditional PAC-learning framework. However, following from the fact that the model is stochastic and therefore an overlap exists between the class distributions, the error rate in unlikely to converge to zero – there is some minimum error rate to which an algorithm can converge. Thus instead of aiming to upper bound the error rate of an algorithm within the standard parameter ϵ with high probability, as in the PAC framework, we must aim to upper bound the *regret* by ϵ instead, where the regret is the difference between the risk (or error rate in this case) associated with a classifier, and the optimal risk (see Section 1.4.3).

In some ways, learning p-concepts is similar to agnostic learning, which is less restrictive than PAC learning and allows stochastic rules to be learnt. However, in agnostic learning no assumptions are made about the class of mechanism generating the data. Within the p-concept framework the assumption is made that the concept being learnt belongs to a known class of concepts – in relation to this chapter we are learning the class of p-concepts which are functions with at most k turning points. Agnostic learning is further explained in Section 1.1.3.

In [31] an algorithm is given which efficiently learns p-concepts from the class of non-decreasing functions on real numbers. The p-concepts learnt by the algorithm described in this chapter are from the class of concepts with at most k turning points (maxima and minima) over real numbers. The distribution over the observations is unknown, and may be chosen by an adversary trying to minimise the success of the algorithm (as in the classic PAC setting). We separate the sampled data by class label

and pass each subset of the data to an unsupervised learner. An observation is then classified by asking each learner to return a value associated with the observation, and the label associated with the learner returning the higher value is given to the observation. Note that the unsupervised learners have no knowledge of the label associated with the data they receive.

Abe et al. (in [1]) extend the work of [31] to show that learning p-concepts in polynomial time with respect to quadratic distance (as in [31]) is equivalent to learning them in polynomial time in terms of KL-divergence[1].

4.1.2 The Problem with Estimating Distributions over Class Labels

Note that the approach taken in this chapter is different from the approach taken in Chapter 2, where each learner estimates the distribution over the samples with a particular class label. Here, we take a different approach – using a discriminant function to return a real value from each learner rather than an estimation of the probability density associated with an observation. In other words, no attempt is made to approximate the distributions over the two class labels.

The reason that this approach has been adopted is due to the "worst case scenario" of an adversary choosing the distribution over the observations. When using a generative algorithm and attempting to estimate the distributions over each class label, imagine a kernel technique being used to give some probability density "weighting" to each observation in a sample. No matter what measure of the width of these kernels is used, an adversary could choose a distribution consisting of spikes of probability density of a far smaller width, such that data with one label is situated next to data of the other label, with the distance separating the two classes being a tiny fraction of the width of the kernel. Whereas a discriminative algorithm could separate the two data sets no matter how small the gap, the generative algorithm would struggle to do so.

4.2 Learning Framework

The learning framework in which we study the learning problem is based on the PAC-learning framework, whereby we are learning an unknown concept from the class of all functions $c()$ with at most k local minima and maxima, and observations are drawn from an *oracle* according to some unknown arbitrary probability distribution.

An example oracle can be seen in Figure 4.1. Note that we can think of the oracle in terms of distributions D_0 and D_1, such that $D_0(x) = D(x)(1 - c(x))$ and $D_1(x) = D(x)c(x)$. This is illustrated in Figure 4.2.

[1] Learning p-concepts in this way focuses on estimating the function $c()$.

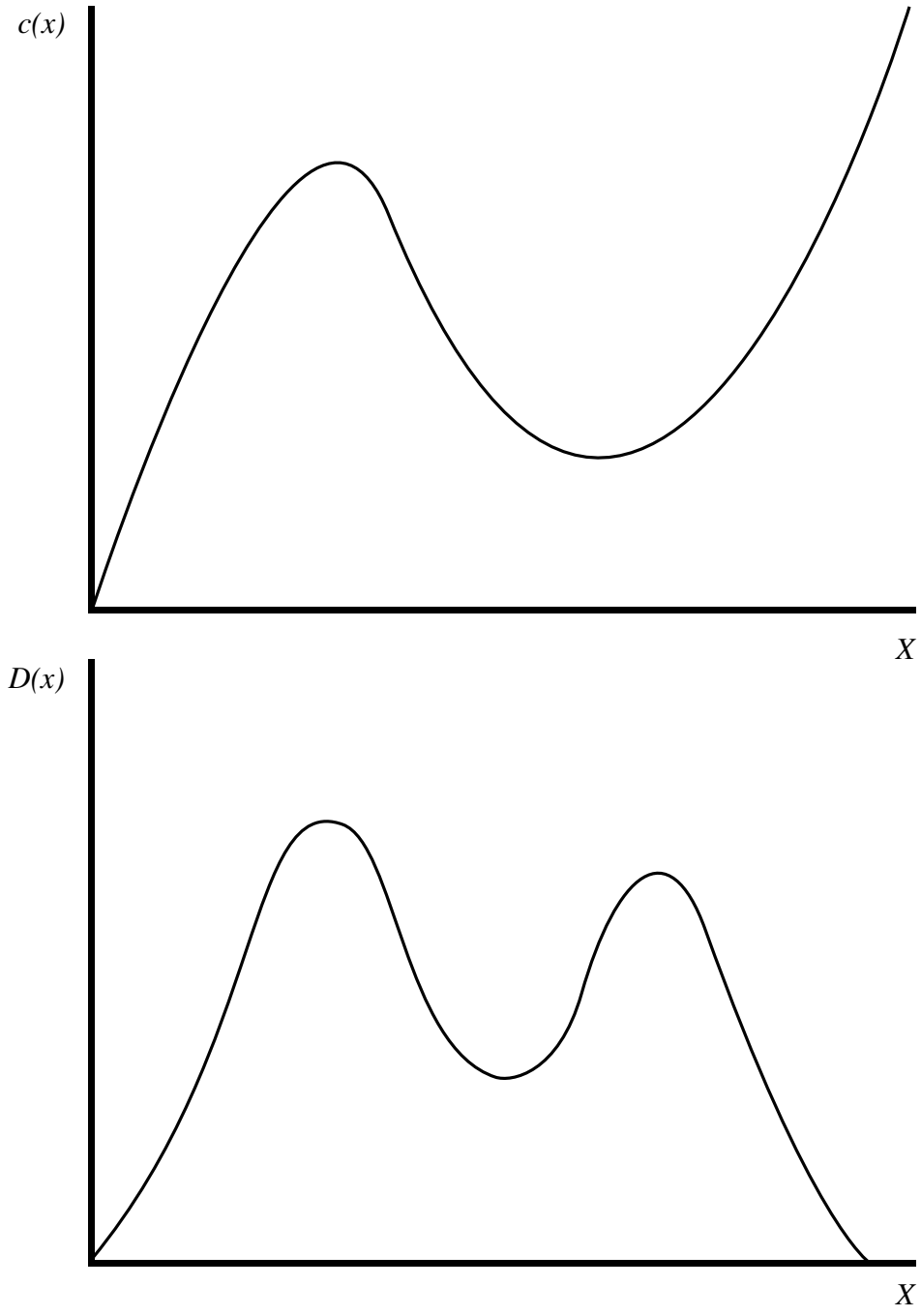


Figure 4.1: Example Oracle – $c(x)$ has 2 turning points.

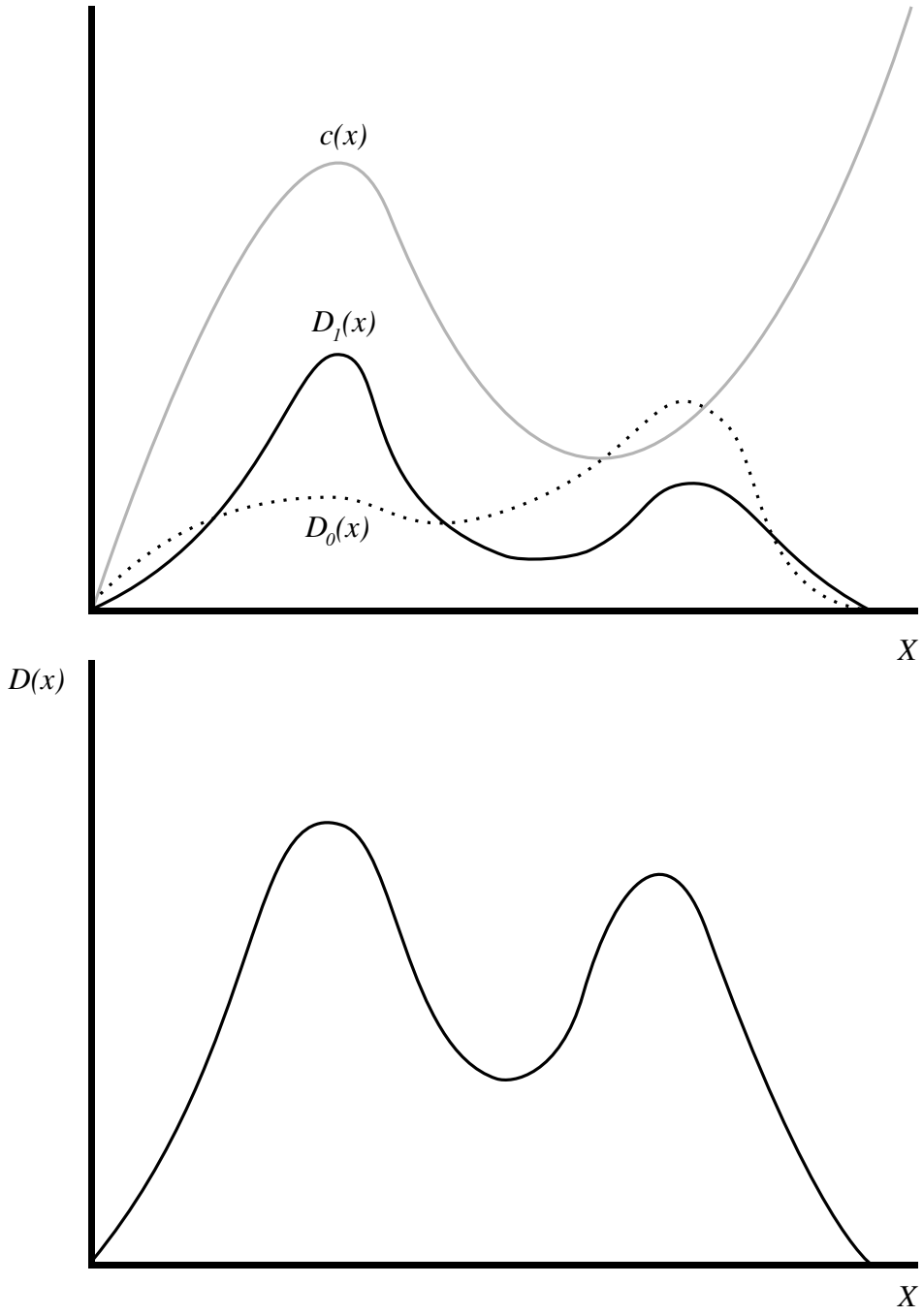


Figure 4.2: D_0 and D_1 – note that $D_0(x) = D(x)(1 - c(x))$ and $D_1(x) = D(x)c(x)$.

For the sake of simplicity we make the restriction that the a priori probabilities of generating an observation with label 0 or 1 are $\frac{1}{2}$ (as is commonly assumed when solving such problems). This allows us to concentrate on the fundamental properties of the problem and of the algorithm without being concerned with the additional complication of estimating a priori probabilities of the classes (an extension that can be made sampling a polynomial amount of data), and an overview of possible alterations to allow the algorithm to work without this condition is given in Section 6.1.

We use a linear loss function to evaluate the performance of a classifier, such that a misclassification of an observation leads to a unit cost and a correct classification leads to a loss of 0. It follows that the expected loss α of function f on an observed value $x \in X$, is given by $\alpha(x, f(x)) = |f(x) - c(x)|$.[2] Risk functional $R(f)$ is such that $R(f) = E_{x \sim D}[\alpha(x, f(x))] = \int_X D(x).\alpha(x, f(x))\ dx$.

In this case, the Bayes Optimal Classifier is defined as follows:

$$f^*(x) = \begin{cases} 1, & \text{if } c(x) \geq \frac{1}{2}, \\ 0, & \text{otherwise.} \end{cases}$$

See Figure 4.3 for an illustration of the optimal classifier. Note that the probability distribution of the observations over the domain is irrelevant to the performance of the optimal classifier and has been omitted from the diagram. Given knowledge of the concept c, the Bayes Optimal Classifier optimises the probability of assigning the correct label at every value in the domain. It should be noted that f^* gives the optimum risk, $R(f^*) = \int_X D(x).\min\{c(x), 1 - c(x)\}\ dx$.

4.3 Algorithm to Learn p-concepts with k Turning Points

The algorithm is given in Figure 4.4. The algorithm can be best explained if split into its three constituent parts as shown – Algorithm 4 generates the training data, and Algorithm 6 labels previously unseen data by calling the two unsupervised learners using Algorithm 5.

We draw a sample of observations from the oracle, from which the observations are generated independently at random, and identically distributed over the unknown probability distribution D on X. Observations are of the form (x, ℓ), where $\ell \in \{0, 1\}$. Note that p-concept c is a function over X with a total of at most k local maxima and minima, such that $\Pr(\ell = 1 \mid x) = c(x)$. Note that the observation comprises an element of the domain and a label of either 0 or 1, rather than a real value representing a probability.

[2] Note that this corresponds to Definition 11 given equal class priors and a linear loss function.

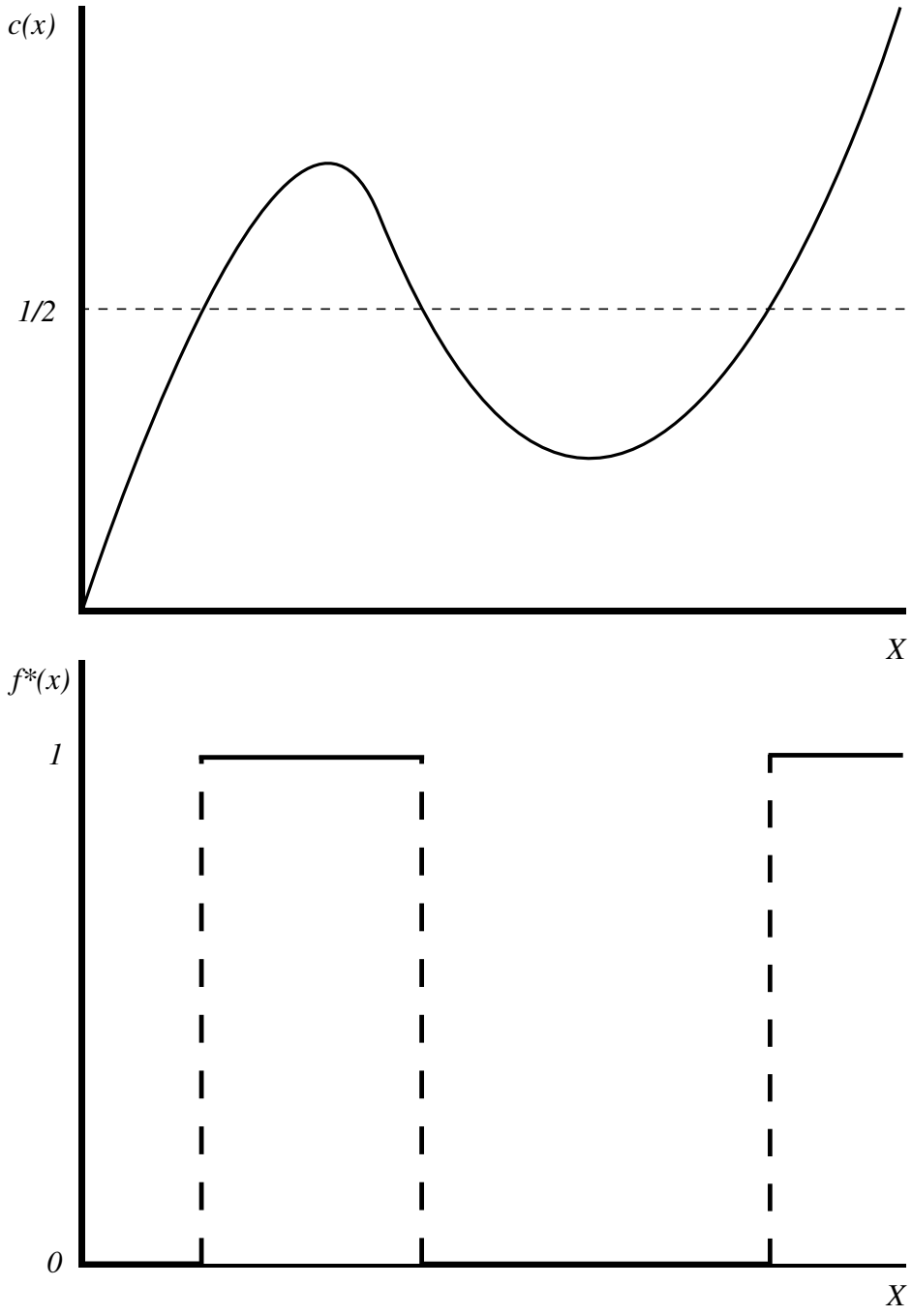


Figure 4.3: The Bayes Optimal Classifier.

Algorithm 4 *Training Algorithm.*

$S_0 = \emptyset,\ S_1 = \emptyset$
$m = \max\left\{\left(2^{11}\epsilon^{-3}(k+1)\right)^{\frac{3}{2}}, \left(\frac{8(k+1)}{\epsilon-(\epsilon^2/16)}\right)^3, 2^{39}\epsilon^{-9}, \delta^{-1}\epsilon^{-3}\right\}$
`repeat`
  `generate observation` (x, ℓ) `from oracle`
  `if` $(|S_\ell| < m)$ `then add` (x, ℓ) `to` S_ℓ
`until` $|S_0| = |S_1| = m$

Algorithm 5 *Learners return numerical values given* $x \in X$.

`getValue`$(\ell, x)\{$
  $w = \min_y\{$`window` W `of width` y `centred at` x, `such that`
    W `contains exactly` $m^{\frac{2}{3}}$ `observations in` $S_\ell\}$
  $v = 1/w$
  `return` v
$\}$

Algorithm 6 *Classification Algorithm.*

`input:` x'
$v_0 =$ `getValue`$(0, x')$
$v_1 =$ `getValue`$(1, x')$
`if` $(v_0 > v_1)$ `then` $\ell' = 0$, `else` $\ell' = 1$
`return` ℓ'

Figure 4.4: Algorithm to learn p-concepts with k turning points.

In Algorithm 4 we draw observations from the oracle until we have seen exactly m observations with each label, discarding any additional observations with a label for which m observations have already been obtained. The observations are stored in two sets, S_0 and S_1 - with S_0 containing m observations with label 0, and S_1 containing m observations with label 1. From these observations we construct classifier f such that the probability of f correctly labelling an element of X is optimised based on the sample data. Note that this classifier is constructed implicitly in the process of labelling newly observed data, and is never explicitly constructed in our algorithm – this will be discussed in Section 4.4.

4.3.1 Constructing the Learning Agents

We pass the sets S_0 and S_1 to a pair of learning agents, $\mathcal{L}_0$ and $\mathcal{L}_1$ respectively, such that learner $\mathcal{L}_0$ receives set S_0 and $\mathcal{L}_1$ receives set S_1. Notice at this stage that the data sample provided to each of the learning agents is made up of a set of m observations of values of X, and that the label of the observations is immaterial (as they are all identical). Therefore, the learning agents are acting within the unsupervised learning framework and can be referred to as unsupervised learners.

When a previously unseen observation is generated, the algorithm is given a value $x_q \in X$ and is asked to label it, as shown in Algorithm 6. The value x_q is passed to learners $\mathcal{L}_0$ and $\mathcal{L}_1$ (using the `getValue()` call to Algorithm 5), each of which returns a numerical value. The highest of these values is used to label the observation – with label 0 if the highest value came from $\mathcal{L}_0$, and with 1 if it came from $\mathcal{L}_1$.

Each learning agent has access to its sample of training data, and takes as input a single value x_q from the domain. Each learner then creates a window centred at x_q, defined as the interval with minimum width containing exactly $m^{\frac{2}{3}}$ observations from the training set. The inverse of this window width is then taken and returned to the algorithm by the learner.

It follows that whichever learner fitted the narrowest window to x_q returns the largest value, and the algorithm labels the observation with the label associated with that learner.

4.4 Analysis of the Algorithm

We show that with high probability, the distribution of observations in a sample generated by the oracle is close to the distribution D over which the sample has been randomly generated. In Lemma 23, we formalise this by imagining the domain to be divided into $\frac{1}{\zeta}$ intervals for some small fraction ζ such that each interval is expected to contain ζ proportion of the sample. It is then shown that the empirical sample is likely to contain,

to within a small multiplicative error, the expected proportion of observations occurring within each interval.

4.4.1 Bounds on the Distribution of Observations over an Interval

Lemma 23 *Let D be an arbitrary probability density function over continuous domain X, let ζ be a small positive constant, and let $\epsilon, \delta \in (0,1)$. We divide X into $\frac{1}{\zeta}$ intervals $[x_0, x_1], ..., \left[x_{\frac{1}{\zeta}-1}, x_{\frac{1}{\zeta}}\right]$, such that for all values of i in $1, ..., \frac{1}{\zeta}$, $\int_{x_{i-1}}^{x_i} D(x)\ dx = \zeta$. A sample S of observations of X generated i.i.d. over D, such that $|S| \geq \frac{3}{\epsilon^2 \zeta} \ln\left(\frac{4}{\zeta\delta}\right)$, contains between $(1-\epsilon)\delta|S|$ and $(1+\epsilon)\delta|S|$ observations in each interval with probability at least $1 - \frac{\delta}{2}$.*

Proof: Suppose that $y_{1,1}, ..., y_{|S|,\frac{1}{\zeta}}$ are $\{0,1\}$-valued random variables, such that $y_{i,j} = 1$ if the ith observation in sample S lies in interval $[x_j, x_{j+1}]$. From the definition of the intervals, and the fact that the observations are generated independently, it is clear that $\Pr(y_{i,j} = 1) = \zeta$.

From standard Chernoff bounds (see for instance [4, 9]) we see that, for $\epsilon \geq 0$ and $j \in \left\{1, ..., \frac{1}{\zeta}\right\}$:

$$\Pr\left(\frac{1}{|S|}\sum_{i=1}^{|S|} y_{i,j} \geq (1+\epsilon)\zeta\right) \leq e^{-\epsilon^2\zeta|S|/3},$$
$$\text{and} \quad \Pr\left(\frac{1}{|S|}\sum_{i=1}^{|S|} y_{i,j} \leq (1-\epsilon)\zeta\right) \leq e^{-\epsilon^2\zeta|S|/2}.$$

The above inequalities can be combined to show that

$$\Pr\left(\sum_{i=1}^{|S|} y_{i,j} \geq (1+\epsilon)\zeta|S| \bigvee \sum_{i=1}^{|S|} y_{i,j} \leq (1-\epsilon)\zeta|S|\right) < 2e^{-\epsilon^2\zeta|S|/3}. \tag{4.1}$$

This equation bounds the probability of a sample being generated in which the number of observations occurring within a specific interval is greater than a multiplicative factor of $(1+\epsilon)$ above or less than a multiplicative factor of $(1-\epsilon)$ below the expected number. We regard samples of this nature to be unrepresentative of the distribution from which it is generated. Equation 4.1 only looks at one specific interval on the domain, and there are $\frac{1}{\zeta}$ such intervals. By taking a union bound on the intervals, we get the following inequality to ensure that a sample is representative across all intervals:

$$\Pr\left(\forall j \in \left\{1, ..., \frac{1}{\zeta}\right\} : (1-\epsilon)\zeta|S| \leq \sum_{i=1}^{|S|} y_{i,j} \leq (1+\epsilon)\zeta|S|\right) \geq 1 - \left(\frac{2}{\zeta}\right) e^{-\epsilon^2\zeta|S|/3}. \tag{4.2}$$

We now show that a sample size of $|S| \geq \frac{3}{\epsilon^2\zeta} \ln\left(\frac{4}{\zeta\delta}\right)$ is sufficient to ensure that the probability in Equation 4.2 is at least $1 - \frac{\delta}{2}$. By substituting $|S| = \frac{3}{\epsilon^2\zeta} \ln\left(\frac{4}{\zeta\delta}\right)$ into the right hand side of Equation 4.2, we get

$$\begin{aligned}
1 - \left(\frac{2}{\zeta}\right) e^{-\epsilon^2\zeta|S|/3} &= 1 - \left(\frac{2}{\zeta}\right) e^{-\epsilon^2\zeta\left(\frac{3}{\epsilon^2\zeta}\ln\left(\frac{4}{\zeta\delta}\right)\right)/3} \\
&= 1 - \left(\frac{2}{\zeta}\right) e^{-\ln\left(\frac{4}{\zeta\delta}\right)} \\
&= 1 - \left(\frac{2}{\zeta}\right)\left(\frac{\zeta\delta}{4}\right) \\
&= 1 - \left(\frac{\delta}{2}\right).
\end{aligned}$$

Finally, by substituting this result back into Equation 4.2 we show that with probability at least $1 - \frac{\delta}{2}$, a sample S such that $|S| \geq \frac{3}{\epsilon^2\zeta} \ln\left(\frac{4}{\zeta\delta}\right)$, contains between $(1-\epsilon)\delta|S|$ and $(1+\epsilon)\delta|S|$ observations in each interval for $0 \leq \epsilon \leq 1$.

$$\Pr\left(\forall j \in \left\{1, ..., \frac{1}{\zeta}\right\} : (1-\epsilon)\zeta|S| \leq \sum_{i=1}^{|S|} y_{i,j} \leq (1+\epsilon)\zeta|S|\right) \geq 1 - \frac{\delta}{2}\ .$$

□

4.4.2 Bounds on the Regret Associated with the Classifier Resulting from the Algorithm

We now show that the regret associated with the classifier defined by the algorithm in Figure 4.4 is bounded within ϵ with probability at least $1 - \delta$, given a training sample size polynomial in ϵ and δ.

Theorem 24 *The algorithm in Figure 4.4 defines classifier $\hat{f}$ such that with probability at least $1 - \delta$, $Regret\left(\hat{f}\right) \leq \epsilon$.*

Proof: The analysis of the algorithm can be broken down into three distinct cases. Each of these cases represent a subset of the domain formed by the composition of noncontiguous intervals. The three subsets are mutually exclusive and span all elements of the domain. We examine each of these cases, bounding the addition to the total regret associated with the classifier in each case.

Notice that the aim is not to bound the risk associated with the classifier $\hat{f}$ defined by the algorithm, rather it is to bound the additional risk incurred by using $\hat{f}$ as opposed to f^*. There is a minimum risk $R(f^*)$ associated with any classifier, and

it is clear that for values of $x \in X$ where $c(x)$ is close to $\frac{1}{2}$, there is little advantage (in terms of risk) of a classifier choosing one label over the other – even the optimum classifier can do no better than a 50/50 success rate when classifying such observations. See Figure 4.5 for an illustration of such regions.

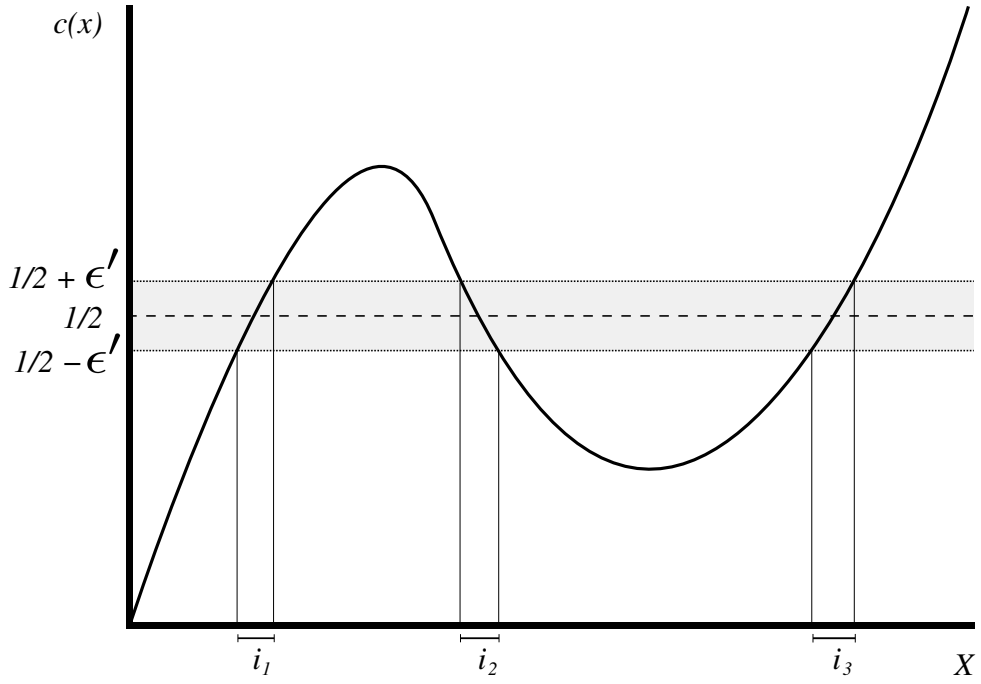


Figure 4.5: Case 1 – covering values of x where the value of $\hat{f}(x)$ has little effect on regret. $i_1 \cup i_2 \cup i_3 = I_1$.

From Figure 4.5 it can be seen that for all values of x in the intervals labeled i_1, i_2 and i_3, $\frac{1}{2} - \epsilon' \leq c(x) \leq \frac{1}{2} + \epsilon'$, where ϵ' is a small fraction of ϵ.[3] Informally, this is what we shall refer to as "Case 1". Formally, we define I_1 to be the subset of the domain representing Case 1, such that $I_1 = \left\{x \in X \mid \frac{1}{2} - \epsilon' \leq c(x) \leq \frac{1}{2} + \epsilon'\right\}$.

"Case 1"

To bound the regret associated with I_1, we consider the worst-case scenario (from the point of view of maximising the regret incurred over I_1). Consider the case where

[3] In fact we shall later define ϵ' such that $\epsilon' = \frac{\epsilon}{8}$, but for now we shall stick to using ϵ' to simplify the analysis.

$\forall x \in X : c(x) = \frac{1}{2} - \epsilon'$ (I_1 covers the whole domain). The optimal classifier f^* for this function is one which predicts label 0 for every value of $x \in X$, since the probability of any observation having label 0 is $\frac{1}{2} + \epsilon'$. It follows that $R(f^*) = \frac{1}{2} - \epsilon'$, regardless of the probability distribution over X.

If we now consider the worst possible classifier for this p-concept – which we shall call f^1 – such that f^1 gives the label 1 to all observations, then it can be seen that $R\left(f^1\right) = \frac{1}{2} + \epsilon'$. This is illustrated in Figure 4.6. It is therefore the case that $Regret\left(f^1\right) = R(f^1) - R(f^*) = 2\epsilon'$.

We have seen that in the worst scenario (where I_1 covers the whole domain, and where the regret associated with the classifier has been maximised) the total regret incurred over values of the domain in I_1 is $2\epsilon'$. There is no need to examine the detail of our algorithm for dealing with values of the domain in I_1 – it is clear that the worst possible classifier does little worse than the optimal classifier – so we use $2\epsilon'$ as the upper bound on the contribution to the regret associated with our classifier over this region.

In contrast to Case 1 where we do not make any claims about the algorithm's performance in labelling data, Case 2 consists of regions where one label has a significantly higher probability of occurring, and we would like the algorithm to reflect this and label the points with the appropriate label. I_2 consists of regions lying outside of I_1, such that either $c(x) < \frac{1}{2} - \epsilon'$ or $c(x) > \frac{1}{2} + \epsilon'$. However, we need to guarantee that when we label the points, the windows generated in Algorithm 5 lie wholly outside I_1 as any overlap will invalidate the method (as will be explained in due course). It is therefore necessary to have a third case defining "buffer zones" between I_1 and I_2, to cover regions where this overlap is possible. These buffer zones will comprise I_3.

Finally we will examine Case 3, with I_3 representing the corresponding subset of the domain. Case 3 deals with the situation where the point being classified lies outside the regions comprising I_1, yet close enough to the region for an overlap to occur with the windows constructed by $\mathcal{L}_0$ and $\mathcal{L}_1$ in Algorithm 5 as described above.

"Case 2"

Figure 4.7 shows the regions comprising I_2 (see the lower of the two graphs), labeled as i_4, i_5, i_6 and i_7. Notice that gaps have been left between these regions and the regions comprising I_1 (see the top graph). These are illustrated in Figure 4.8, and will be explained in detail later on.

We wish to ensure that given a *representative* sample (as discussed in Lemma 23) of observations, the algorithm will choose the most likely label for each value of the domain lying within I_2, namely 0 if $c(x) < \frac{1}{2} - \epsilon'$ and 1 if $c(x) > \frac{1}{2} + \epsilon'$. If this occurs (the most likely label is chosen over all $x \in I_2$) it follows that $\forall x \in I_2 : \hat{f}(x) = f^*(x)$.

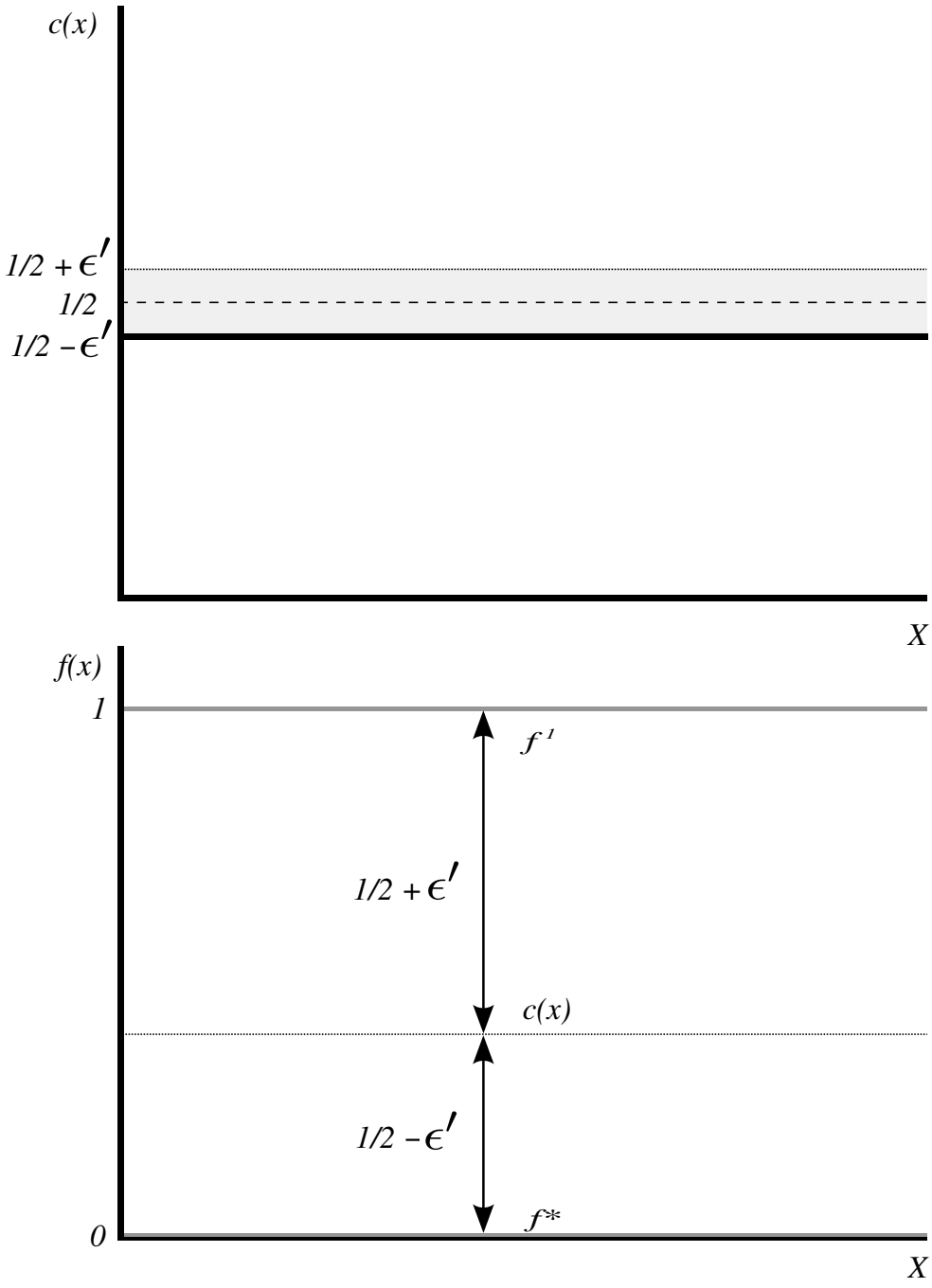


Figure 4.6: Case 1 – Worst Case Scenario.

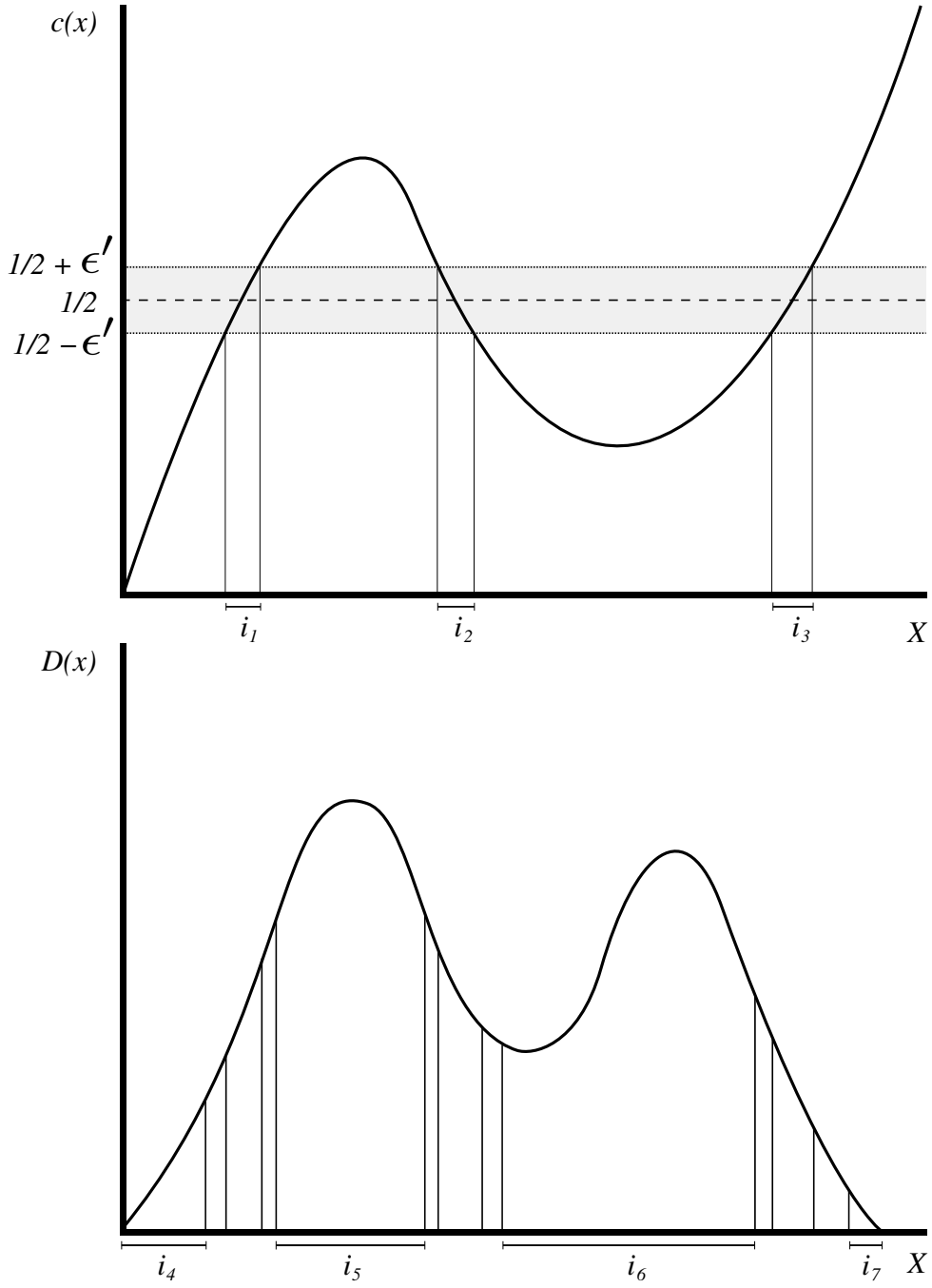


Figure 4.7: Case 2 – intervals where it is important that $\hat{f}(x)$ should predict the same label as $f^*(x)$. $I_1 = i_1 \cup i_2 \cup i_3$, $I_2 = i_4 \cup i_5 \cup i_6 \cup i_7$, and the remaining intervals are I_3.

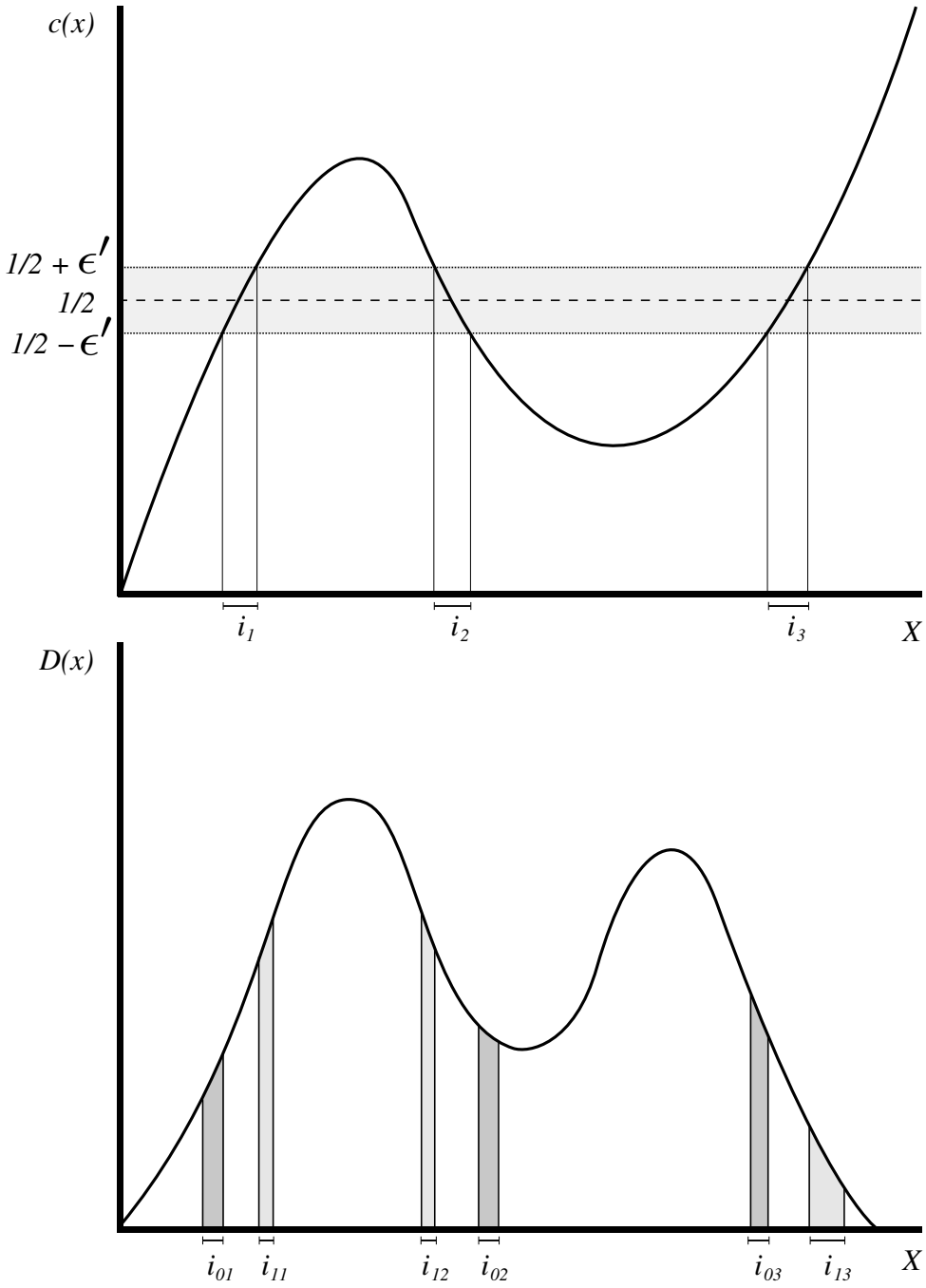


Figure 4.8: Case 3 – $I_3 = i_{01} \cup i_{11} \cup i_{02} \cup i_{12} \cup i_{03} \cup i_{13}$. The intervals with dark shading represent values of x for which $c(x) < \frac{1}{2} - \epsilon'$, and the lighter areas represent values of x for which $c(x) > \frac{1}{2} + \epsilon'$.

It is therefore the case that over this region no additional regret is accrued since the risk associated with $\hat{f}$ and f^* is identical.

To analyse the algorithm further we need to define what is meant by a *representative* sample of observations. We will then show that with probability at least $1-\delta$ the algorithm generates representative samples of observations with each label.

The algorithm generates a sample of m observations with each label from the oracle, where m is defined in Algorithm 4. For each label, we wish to divide up the domain in the way described in Lemma 23, into $\frac{1}{\zeta}$ intervals, each containing roughly the expected number of points (with that label). Note that the intervals defined for labels 0 and 1 will differ (unless $c(x)=\frac{1}{2}$ over the entire domain). We shall refer to these sets of intervals as $\mathbb{Z}_0$ and $\mathbb{Z}_1$ respectively. It can be seen from Lemma 23 that, for suitably small values of ζ and ϵ'' which will be defined later, if a sample of at least $\frac{3}{(\epsilon'')^2\zeta}\ln\left(\frac{4}{\zeta\delta}\right)$ observations with a specific label is generated then with probability at least $1-\frac{\delta}{2}$, each of the intervals will contain between $(1-\epsilon'')\,\zeta m$ and $(1+\epsilon'')\,\zeta m$ observations.

For the overall sample to be representative, we require that for each label's sample, the observations are divided more or less equally between the intervals, such that each interval contains between $(1-\epsilon'')\,\zeta m$ and $(1+\epsilon'')\,\zeta m$ observations as described above.

We now substitute expressions for ϵ'' and ζ – we choose $\epsilon''=\frac{\epsilon}{16}$ and $\zeta=\frac{2^8}{\epsilon^2 m^{\frac{2}{3}}}$. This means that with probability at least $1-\frac{\delta}{2}$ there are $2^{-8}\epsilon^2 m^{\frac{2}{3}}$ intervals, each containing between $\left(1-\frac{\epsilon}{16}\right)2^8\epsilon^{-2}m^{\frac{1}{3}}$ and $\left(1+\frac{\epsilon}{16}\right)2^8\epsilon^{-2}m^{\frac{1}{3}}$ observations, so long as

$$\begin{aligned} m &\geq 3m^{\frac{2}{3}}\ln\left(\frac{\epsilon^2 m^{\frac{2}{3}}}{2^6\delta}\right) \\ \frac{1}{3}m^{\frac{1}{3}} &\geq \ln\left(\frac{\epsilon^2 m^{\frac{2}{3}}}{2^6\delta}\right) \\ e^{-\frac{1}{3}m^{\frac{1}{3}}} &\leq \left(\frac{2^6\delta}{\epsilon^2 m^{\frac{2}{3}}}\right). \end{aligned}$$

It can be verified that this inequality is satisfied when $m\geq\delta^{-1}\epsilon^{-3}$ and $m\geq 2^{39}$. From Algorithm 4 it can be verified that this is the case.

Since the samples of observations with label 0 and label 1 are each representative with probability at least $1-\frac{\delta}{2}$, it follows that both generated samples will be representative with probability at least $1-\delta$.

Assuming that both samples used by the algorithm are representative, we show that all observations occurring in a region of I_2 for which $c(x)>\frac{1}{2}+\epsilon'$ are given label 1 by the algorithm – and by symmetry observations in regions of I_2 for which $c(x)<\frac{1}{2}-\epsilon'$ are labeled 0. Consider an observation at $x'\in X$, such that $x'\in I_2$ and $c(x)>\frac{1}{2}+\epsilon'$.

Learners $\mathcal{L}_0$ and $\mathcal{L}_1$ generate windows, centred at x', each containing $m^{\frac{2}{3}}$ observations from the training data with their respective label.

First we examine the window generated by $\mathcal{L}_1$. Given that $c(x) > \frac{1}{2} + \epsilon'$, the observation should be labeled 1 by the algorithm, and this happens when $\mathcal{L}_1$'s window is narrower than that of $\mathcal{L}_0$. Note that the whole region of the domain containing the window generated by $\mathcal{L}_1$ must lie outside I_1 for the following analysis to hold – however the same restriction is not put on the window of $\mathcal{L}_0$. If $\mathcal{L}_0$'s window encroaches on I_1 and $\mathcal{L}_1$'s does not, it follows that the window generated by $\mathcal{L}_1$ must be narrower and therefore the observation will be labeled correctly.

We define $range_1$ as the minimum set of intervals in $\mathbb{Z}_1$ such that $range_1$ contains the entire window generated by $\mathcal{L}_1$, and r_1 is the number of these intervals. By the definition of these intervals it follows that $r_1 \leq 2 + \frac{m^{\frac{2}{3}}}{(1-\epsilon'')\zeta m}$, as the window contains $m^{\frac{2}{3}}$ observations from the sample and each interval must contain at least $(1-\epsilon'')\zeta m$ observations for the sample to be representative, and in addition to this the window may overlap at most 2 incomplete intervals. Therefore,

$$r_1 \leq 2 + \frac{1}{(1-\epsilon'')\zeta m^{\frac{1}{3}}} \ . \tag{4.3}$$

Let $\bar{r}_1$ be the upper bound on r_1. $\bar{r}_1 = 2 + \frac{1}{(1-\epsilon'')\zeta m^{\frac{1}{3}}}$. We now define $range_0$ as the minimum set of intervals in $\mathbb{Z}_0$ such that $range_0$ contains the entire window generated by $\mathcal{L}_0$, and r_0 is the number of these intervals. Since $c(x) > \frac{1}{2} + \epsilon'$, the probability density of observations with label 0 occurring over this region must be at least a factor of $\frac{\frac{1}{2}-\epsilon'}{\frac{1}{2}+\epsilon'} \leq 1 - 2\epsilon'$ smaller than that of observations with label 1. Referring to Equation 4.3, it can be seen that

$$\begin{aligned} r_0 \leq \quad & 2 + \bar{r}_1(1-2\epsilon') \\ = \quad & 2 + \left(2 + \frac{1}{(1-\epsilon'')\zeta m^{\frac{1}{3}}}\right)(1-2\epsilon') \\ = \quad & 4(1-\epsilon') + \frac{1-2\epsilon'}{(1-\epsilon'')\zeta m^{\frac{1}{3}}} \ . \end{aligned} \tag{4.4}$$

Let m_0 be the maximum number of observations in $\mathcal{L}_0$'s sample occurring across r_0 intervals in $\mathbb{Z}_0$. We define the upper bound on r_0 as $\bar{r}_0 = 4(1-\epsilon') + \frac{1-2\epsilon'}{(1-\epsilon'')\zeta m^{\frac{1}{3}}}$.

From Equation 4.4, it can be seen that

$$\begin{aligned} m_0 = \quad & \bar{r}_0(1+\epsilon'')\zeta m \\ = \quad & \left(4(1-\epsilon') + \frac{1-2\epsilon'}{(1-\epsilon'')\zeta m^{\frac{1}{3}}}\right)(1+\epsilon'')\zeta m \\ \leq \quad & 4(1-\epsilon')(1+\epsilon'')\zeta m + \frac{(1+\epsilon'')(1-2\epsilon')m^{\frac{2}{3}}}{1-\epsilon''} \ . \end{aligned} \tag{4.5}$$

By substituting the values of $\zeta = \frac{2^8}{\epsilon^2 m^{\frac{2}{3}}}$, $\epsilon' = \frac{\epsilon}{8}$ and $\epsilon'' = \frac{\epsilon}{16}$ into Equation 4.5, we find that

$$\begin{aligned} m_0 \ &\leq \ 2^{10}\epsilon^{-2}\left(1-\frac{\epsilon}{8}\right)\left(1+\frac{\epsilon}{16}\right)m^{\frac{1}{3}} + \left(\frac{1+\frac{\epsilon}{16}}{1-\frac{\epsilon}{16}}\right)\left(1-\frac{\epsilon}{4}\right)m^{\frac{2}{3}} \\ &< \ 2^{10}\epsilon^{-2}\left(1-\frac{\epsilon}{8}\right)\left(1+\frac{\epsilon}{16}\right)m^{\frac{1}{3}} + \left(1+\frac{\epsilon}{8}\right)\left(1-\frac{\epsilon}{4}\right)m^{\frac{2}{3}}. \end{aligned}$$

For values of $m \geq 2^{39}\epsilon^{-9}$, we see that $m_0 < m^{\frac{2}{3}}$ from the following:

$$\begin{aligned} m_0 \ &< \ 2^{10}\epsilon^{-2}\left(1-\frac{\epsilon}{8}\right)\left(1+\frac{\epsilon}{16}\right)2^{13}\epsilon^{-3} + \left(1+\frac{\epsilon}{8}\right)\left(1-\frac{\epsilon}{4}\right)2^{26}\epsilon^{-6} \\ &= \ 2^{23}\epsilon^{-5}\left(1-\frac{\epsilon}{16}-\frac{\epsilon^2}{128}\right) + 2^{26}\epsilon^{-6}\left(1-\frac{\epsilon}{8}-\frac{\epsilon^2}{32}\right) \\ &= \ 2^{26}\epsilon^{-6} + 2^{23}\epsilon^{-5} - \left(2^{19}\epsilon^{-4} + 2^{16}\epsilon^{-3} + 2^{23}\epsilon^{-5} + 2^{21}\epsilon^{-4}\right) \\ &< \ 2^{26}\epsilon^{-6} \leq m^{\frac{2}{3}}. \end{aligned}$$

We know that the window generated by $\mathcal{L}_1$ contains $m^{\frac{2}{3}}$ observations, and we have shown that m_0 – the maximum number of observations with label 0 within $range_0$, is smaller than this. Therefore the window containing $m^{\frac{2}{3}}$ observations with label 0 must be wider than this. It follows that the value returned by $\mathcal{L}_1$ (the inverse of the window width) is higher, so the observation receives label 1.

From this result it can be seen that (provided that the data sample of each label is representative – as in Equation 4.2), all observations occurring within I_2 will be given the most probable label by the algorithm. This means that no additional regret is incurred in the region, since the algorithm predicts the same label as the optimal classifier in all cases.

"Case 3"

We now examine Case 3, which provides the buffer zones I_3 between I_1 and I_2 to ensure that for the analysis of Case 2, $c(x) > \frac{1}{2} + \epsilon'$ (or $c(x) < \frac{1}{2} - \epsilon'$) across the whole of $range_1$ (or the corresponding range for the analysis when $c(x) < \frac{1}{2} - \epsilon'$) as stated above.

In a similar way to Case 1, we make no claim about the performance of the algorithm on observations generated in I_3. However, in Case 1 the optimal algorithm does little better than to randomly guess the label, which means that the additional regret incurred is limited. In this case the optimal algorithm classifies at least a proportion $\frac{1}{2}+\epsilon'$ of the observations correctly, so if our algorithm chooses the less likely label as a classification, we incur additional expected loss.

For each of the turning points, of which there at most k, there can be at most two regions where the the function $c(x)$ crosses either $\frac{1}{2}-\epsilon'$ or $\frac{1}{2}+\epsilon'$ (note that there is an additional region at either end of the domain, where the function enters I_1 for the first time and when it leaves I_1 for the last time). In each of these regions we define an interval such that at least $m^{\frac{2}{3}}$ observations of the label with highest probability occur within that region – or until the threshold is crossed again into I_1. For example, when the function crosses $c(x)=\frac{1}{2}-\epsilon'$ at some value of $x \in X$, we define an interval so that $c(x) \leq \frac{1}{2}-\epsilon'$ over the entire interval, and such that the interval must contain at least $m^{\frac{2}{3}}$ points with label 0 in a representative sample, unless by doing so we cross back into I_1, at which point the interval ends. See Figure 4.8 for an illustration of these regions. Intervals i_{01}, i_{02} and i_{03} are all regions where $c(x) \leq \frac{1}{2}-\epsilon'$, and i_{11}, i_{12} and i_{13} are regions where $c(x) \geq \frac{1}{2}+\epsilon'$. As such, intervals i_{0*} are defined to contain a minimum of $m^{\frac{2}{3}}$ observations with label 0, and intervals i_{1*} are defined to contain a minimum of $m^{\frac{2}{3}}$ observations with label 1.

The addition to the regret incurred by the algorithm in I_3 is at most the probability of an observation with the higher $c(x)$ value being generated in the region. By symmetry the following applies both to intervals where $c(x) \leq \frac{1}{2}-\epsilon'$ and those where $c(x) \geq \frac{1}{2}+\epsilon'$, but we shall analyse the former. Let $range'$ be the minimum set of intervals in $\mathbb{Z}_0$ containing the entirety of the maximum-width window generated by $\mathcal{L}_0$ on a representative sample, such that the window contains the observation closest to I_1 (at the boundary under scrutiny) in its sample without encroaching on it. Let the number of these intervals in $range'$ be r', such that $r' \leq 2+\frac{m^{\frac{2}{3}}}{(1-\epsilon'')\zeta m}$.

This value of r' represents the largest number of intervals in $\mathbb{Z}_0$ which could be overlapped by the window described above. The probability of an observation with label 0 being generated in this region is at most $r'\zeta = \left(2+\frac{m^{\frac{2}{3}}}{(1-\epsilon'')\zeta m}\right)\zeta$. Therefore the maximum probability that an observation of the label with the higher likelihood being generated in any of at most $2(k+1)$ regions comprising I_3, is $\left(2+\frac{m^{\frac{2}{3}}}{(1-\epsilon'')\zeta m}\right)2(k+1)\zeta = 4(k+1)\zeta + \frac{2(k+1)}{(1-\epsilon'')m^{\frac{1}{3}}}$. This term is our upper bound on the regret incurred by the algorithm over I_3.

Examining the first part of this expression, we see that for $m \geq \left(\frac{2^{11}(k+1)}{\epsilon^3}\right)^{\frac{3}{2}}$,

$$4(k+1)\zeta = \frac{2^{10}(k+1)}{\epsilon^2 m^{\frac{2}{3}}} \leq \frac{2^{10}(k+1)}{\epsilon^2 \left(\frac{2^{11}(k+1)}{\epsilon^3}\right)} = \epsilon/2. \tag{4.6}$$

Now examining the second part of the expression, for $m \geq \left(\frac{8(k+1)}{\epsilon - \frac{\epsilon^2}{16}}\right)^3$,

$$\frac{2(k+1)}{(1-\epsilon'')m^{\frac{1}{3}}} = \frac{2(k+1)}{\left(1-\frac{\epsilon}{16}\right)m^{\frac{1}{3}}} \leq \frac{2(k+1)}{\left(1-\frac{\epsilon}{16}\right)\left(\frac{8(k+1)}{\epsilon - \frac{\epsilon^2}{16}}\right)} = \epsilon/4. \tag{4.7}$$

Summary

We have now shown that with probability at least $1-\delta$, both samples of data are representative. It has been shown that the regret associated with our algorithm's classifier is at most $2\epsilon' = \frac{\epsilon}{4}$ over I_1. It has also been shown that if both samples are representative and given sufficiently large sample sizes, no regret is incurred from I_2, and (from Equations 4.6 and 4.7), at most $\frac{\epsilon}{2} + \frac{\epsilon}{4} = \frac{3\epsilon}{4}$ is incurred from I_3. Therefore, it has been shown that with probability at least $1-\delta$, $Regret\left(\hat{f}\right) \leq \epsilon$. □

Chapter 5

Learning PDFA

A probabilistic deterministic finite-state automaton (PDFA) is an automaton with a finite number of states that has, for each state, a probability distribution over the transitions leaving that state. We study these automata in a setting in which a function maps the set of all transitions to symbols from a finite alphabet, such that a symbol is emitted as a transition is used. The automaton is deterministic in that at most one transition with a given symbol is possible from any state. Thus a PDFA defines a probability distribution over the set of all strings over its alphabet.

In the introduction to Chapter 2 we discuss the method of solving classification problems through using a Bayes classifier in conjunction with probability distributions over each class label. Automata and their associated distributions over outputs can be used in multiclass classification problems where sequences are being modeled. Such problems include genetic sequencing, natural language processing, and all manner of sound and image analysis such as speech recognition.

5.1 An overview of automata

5.1.1 Related Models

PDFAs are just one of a variety of structures used to model stochastic processes in fields such as AI and machine learning. Similar structures seen in related work include probabilistic nondeterministic finite automata (PNFA), hidden Markov models (HMM), and partially observable Markov decision processes (POMDP).

Probabilistic nondeterministic finite automata

A PNFA is similar to a PDFA, but whereas a PDFA may have at most one transition with a given symbol leaving a state, a PNFA may have more than one transition emitting the

same symbol. Thus even with knowledge of the starting state and the symbol generated by a transition from this state, the machine may be in one of several states. This model has more expressive power, and consequently it is harder to obtain positive results for learning.

Hidden Markov models

In a HMM, each state has a probability distribution over symbols, and a symbol is emitted when that state is visited. HMMs and PNFAs have essentially the same expressive power [18][1]. Abe and Warmuth [2] give a strong computational negative result for learning PNFAs and HMMs, namely that is it hard to maximise the likelihood of an individual string using these models (for a fixed number of states).

Partially observable Markov decision processes

POMDPs are associated with online learning problems where choices can be made by the learner as data is analysed. There is an underlying probabilistic finite automaton whose states are not directly observable. A POMDP takes actions as input from the learner, where an observation is output and a reward is awarded to the learner (at each step, the reward depends on the transition taken and the learner's action). The objective in these learning problems is to maximise some function of the rewards. A POMDP is an extension of the notion of a Markov decision process to situations where the state is not always known to the algorithm.

5.1.2 PDFA Results

Positive results for PAC-learning sub-classes of PDFAs were introduced by Ron et al. [41], where they show how to PAC-learn *acyclic* PDFAs, and apply the algorithm to speech and handwriting recognition. Clark and Thollard [10] presented an algorithm that PAC-learns general PDFAs, using the KL-divergence as the error measure (the distance between the true distribution defined by the target PDFA, and the hypothesis returned by the algorithm). The algorithm is polynomial in three parameters: the number of states, the "distinguishability" of states, and the expected length of strings generated from any state of the target PDFA. Distinguishability (defined in Section 5.3) is a measure of the extent to which any pair of states have an associated string that is significantly more likely to be generated from one state than the other. While unrestricted PDFAs can

[1] As it is shown in [2], a PNFA can only be encoded as a HMM if it is the case that from each state in the automaton, the ratio between the probabilities of the different symbols emitted by the outgoing transitions is independent of the states arrived at by the transitions. i.e. the probability of observing symbol σ given that a transition from state q_0 to state q_1 is followed, is equal to the probability of observing symbol σ given a transition to state q_2 from q_0, for any pair of states q_1 and q_2.

encode noisy parity functions [30] (believed to be hard to PAC-learn), these PDFAs have "exponentially low" distinguishability.

5.1.3 Significance of Results

We study the problem of PAC-learning general PDFAs (as in [10]), using variation distance instead of KL-divergence. This modification allows some strengthening and simplifications of the resulting algorithms. The main one is that—as conjectured in [10]—a polynomial bound on the sample-size requirement is obtained that does not depend on the length of strings generated by the automaton. We also have no need for a distinguished "final symbol" that must terminate all data strings, or a "ground state" in the automaton constructed by the algorithm[2]. We have also simplified the algorithm by not re-sampling at each iteration; instead we use the same sample in all iterations.

The L_1 distance and KL divergence are defined in Section 1.4. KL-divergence is in a strong sense a more "sensitive" measure than variation distance – this was pointed out in Kearns et al. [30], which introduced the general topic of PAC-learning probability distributions. In Cryan et al. [13] a smoothing technique is given for distributions over the boolean domain (where the length of strings is a parameter of the problem)—an algorithm that PAC-learns distributions using the variation distance can be converted to an algorithm that PAC-learns using the KL-divergence. (Abe et al. [1] give a similar result in the context of learning p-concepts.) Over the domain Σ^* (strings of unrestricted length over alphabet Σ) that technique does not apply, which is why we might expect stronger results as a result of switching to the L_1 distance.

Our approach follows [10], in that we divide the algorithm into two parts. The first (Algorithm 7 of Figure 5.1) finds a DFA that represents the structure of the hypothesis automaton, and the second (Algorithm 8 of Figure 5.2) finds estimates of the transition probabilities. Algorithm 7 constructs (with high probability) a DFA whose states and transitions are a subset of those of the target. Algorithm 8 learns the transition probabilities by following the paths of random strings through the DFA constructed by Algorithm 7, taking advantage of the fact that commonly-used transitions can be estimated more precisely.

[2]The presence of the ground state is the reason why it is necessary for the expected length of a string to be known in [10]. Due to the nature of KL-divergence, it is essential to avoid unbounded logarithmic errors occurring – therefore the constructed automaton must accept all strings over the alphabet. This is done by constructing a ground state, q^*, to which any undefined transitions are linked (where there is no corresponding transition for symbol σ leaving node q, a transition (q, σ) is created with some small associated probability such that $\tau(q, \sigma) = q^*$). At the ground node, there are transitions $\tau(q^*, \sigma) = q^*$ defined for all $\sigma \in \Sigma$, and the expected length of a string can be used to calculate $\gamma(q^*, \sigma)$ such that strings with the required expected length may be generated. Finally, a transition $\tau(q^*, \sigma_f) = q_f$ links the ground node to the finishing state, emitting the final symbol σ_f (which is not required in our definition of the problem).

In Section 5.7 we show that for the subclass of PDFA consisting of automata that can be represented as a finite length string, PAC-learning in terms of L_1 distance is equivalent to inefficient PAC-learning in terms of KL-divergence. The method used to convert from a distribution which is close to the target automaton in terms of L_1 distance to an equivalent distribution which is equivalently close in KL-divergence is similar to the ϵ-Bayesian averaging performed in [1].

5.2 Defining a PDFA

A PDFA can stochastically generate strings of symbols as follows. The automaton has a finite set of states – one of which is distinguished as the *initial state*. The automaton generates a string by making transitions between states (starting at the initial state), each transition occurring with a constant probability specifically associated with that transition. The symbol labelling that transition is then output. The automaton halts when the *final state* is reached. It is common for a definition of a PDFA to include the specification of a final symbol at the end of all words; we do not require that restriction here.

Definition 25 *A PDFA A is a sextuple $(Q, \Sigma, q_0, q_f, \tau, \gamma)$, where*

- *Q is a finite set of states,*
- *Σ is a finite set of symbols (the alphabet),*
- *$q_0 \in Q$ is the initial state,*
- *$q_f \notin Q$ is the final state,*
- *$\tau : Q \times \Sigma \to Q \cup \{q_f\}$ is the (partial) transition function,*
- *$\gamma : Q \times \Sigma \to [0, 1]$ is the function representing the probability of a given symbol (and the corresponding transition) occurring from a given state.*

Where appropriate, we extend the use of τ and γ to strings:

$$\begin{aligned} \tau(q, \sigma_1\sigma_2...\sigma_k) &= \tau(\tau(q, \sigma_1), \sigma_2...\sigma_k), \text{ and} \\ \gamma(q, \sigma_1\sigma_2...\sigma_k) &= \gamma(q, \sigma_1).\gamma(\tau(q, \sigma_1), \sigma_2...\sigma_k). \end{aligned}$$

It is required that $\sum_{\sigma\in\Sigma} \gamma(q, \sigma) = 1$ for all $q \in Q$, and when $\tau(q, \sigma)$ is undefined, $\gamma(q, \sigma) = 0$. In addition q_f is reachable from any state of the automaton, that is, for all $q \in Q$ there exists $s \in \Sigma^*$ such that $\tau(q, s) = q_f \wedge \gamma(q, s) > 0$.

If A denotes a PDFA, it follows that A defines a probability distribution over strings in Σ^*.[3] Let $D_A(s)$ denote the probability that A generates $s \in \Sigma^*$, so we have $D_A(s) = \gamma(q_0, s)$ for s such that $\tau(q_0, s) = q_f$.

We use the pair (q, σ) to denote the transition from state $q \in Q$ labeled with character $\sigma \in \Sigma$. Let $D_A(q)$ denote the probability that a random string generated by A uses state $q \in Q$. Thus $D_A(q)$ is the probability that $s \sim D_A$ (i.e. s sampled from distribution D_A) has a prefix p with $\tau(q_0, p) = q$. In a similar way, $D_A(q, \sigma)$ denotes the probability that a random string generated by A uses transition (q, σ)—the probability that a random string $s \sim D_A$ has a prefix $p\sigma$ with $\tau(q_0, p) = q$.

Suppose D and D' are probability distributions over Σ^*. A class $\mathcal{D}$ of probability distributions is PAC-learnable by algorithm $\mathcal{A}$ with respect to the variation distance if the following holds. Given parameters $\epsilon > 0$, $\delta > 0$, and access to samples from any $D \in \mathcal{D}$, using runtime and sample size polynomial in $1/\epsilon$ and $1/\delta$, $\mathcal{A}$ should, with probability at least $1 - \delta$, output a distribution D' with $L_1(D, D') < \epsilon$. If $D \in \mathcal{D}$ is described in terms of additional parameters that represent the complexity of D, then we require $\mathcal{A}$ to be polynomial in these parameters as well as $1/\epsilon$ and $1/\delta$.

5.3 Constructing the PDFA

In this section we describe the first part of the algorithm, which constructs the underlying DFA of a target PDFA A. That is, it constructs the states Q and transitions given by τ, but not the probabilities given by γ. The algorithm has access to a source of strings in Σ^* generated by D_A. We allow "very unlikely" states to be ignored, as described at the end of this section where we explain how our algorithm differs from previous related algorithms. Properties of the constructed DFA are proved in Section 5.4.

The algorithm is shown in Figure 5.1. We have the following parameters (in addition to the PAC parameters ϵ and δ):

- $|\Sigma|$: the alphabet size,
- n: an upper bound on the number of states of the target automaton,
- μ: a lower bound on distinguishability, defined below.

In the context of learning using the KL-divergence, a simple class of PDFAs (see Clark and Thollard [10]) can be constructed to show that the parameters above are insufficient for PAC learnability in terms of just those parameters. In [10], parameter L is also used, denoting the expected length of strings.

[3] The reachability of the final state ensures that A will halt with probability 1.

From the target automaton A we generate a hypothesis automaton H using a variation on the method described by [10] utilising *candidate nodes*, where the L_∞ norm between the suffix distributions of states is used to distinguish between them (as studied also in [28, 41]). We define a candidate node in the same way as [10]. Suppose G is a graph whose vertices correspond to a subset of the states of A, and whose edges correspond to transitions. Initially G will have a single vertex corresponding to the initial state; G is then constructed in a greedy incremental fashion.

5.3.1 Structure of the Hypothesis Graph

$G = \langle V, E \rangle$ denotes the directed graph constructed by the algorithm. V is the set of vertices and E the set of edges. Each edge is labeled with a letter $\sigma \in \Sigma$, so an edge is a member of $V \times \Sigma \times V$. Note that due to the deterministic nature of the automaton, there can be at most one vertex v_q such that $(v_p, \sigma, v_q) \in E$ for any $v_p \in V$ and $\sigma \in \Sigma$.

Candidate nodes

Definition 26 *A candidate node in hypothesis graph G is a pair (u, σ) (also denoted $\hat{q}_{u,\sigma}$), where u is a node in the graph and $\sigma \in \Sigma$ where $\tau_G(u, \sigma)$ is undefined.*

Let D_q denote the distribution over strings generated using state q as the initial state, so that

$$D_q(s) = \gamma(q, s) \text{ for } s \text{ such that } \tau(q, s) = q_f.$$

Given a sample S of strings generated from D_A, we define a multiset associated with each node or candidate node in a hypothesis graph. The multiset for node q is an i.i.d. sample from D_q, derived from S, obtained by taking members of S that use q and deleting their prefixes that reach q for the first time in a string. For a candidate node, we use the following definition.

Definition 27 *Given a sample S, candidate node $\hat{q}_{u,\sigma}$ has multiset $S_{u,\sigma}$ associated with it, where for each $s \in S$, we add s'' to $S_{u,\sigma}$ whenever $s = s'\sigma s''$ and $\tau_G(q_0, s') = u$.*

The notion of distinguishability

The L_∞-norm is a measure of distance between a pair of distributions, defined as follows.

Definition 28 $L_\infty(D, D') = \max_{s \in \Sigma^*} |D(s) - D'(s)|$.

Definition 29 *The parameter of distinguishability, μ, is a lower bound on the L_∞-norm between D_{q_1} and D_{q_2} for any pair of nodes (q_1, q_2), where q_1 and q_2 are regarded as having sufficiently different suffix distributions in order to be considered separate states.*

We define as follows the $\hat{L}_\infty$-norm (an empirical version of the L_∞-norm) with respect to multisets of strings S_{q_1} and S_{q_2}, where S_{q_1} and S_{q_2} have been respectively sampled from D_{q_1} and D_{q_2}.

Definition 30 *For nodes q_1 and q_2, with associated multisets S_{q_1} and S_{q_2},*

$$\hat{L}_\infty\left(D_{q_1}, D_{q_2}\right) = \max_{s \in \Sigma^*} \left(\left| \frac{|s \in S_{q_1}|}{|S_{q_1}|} - \frac{|s \in S_{q_2}|}{|S_{q_2}|} \right| \right)$$

where D_q is the empirical distribution over the strings in the multiset S_q associated with q, and where $|s \in S_q|$ is the number of occurrences of string s in multiset S.

As in [41, 10], we say that a pair of nodes (q_1, q_2) are *μ-distinguishable* if $L_\infty(D_{q_1}, D_{q_2}) = \max_{s \in \Sigma^*} |D_{q_1}(s) - D_{q_2}(s)| \geq \mu$.

Although we claim only to learn PDFA with a bounded μ-distinguishability between all pairs of states, in fact – as observed by [41] – it is enough that all pairs of states with non-negligible weights be distinguishable.

5.3.2 Mechanics of the Algorithm

The algorithm uses two quantities, m_0 and N. m_0 is the number of suffixes required in the multiset of a candidate node for the node to be added as a state (or as a transition) to the hypothesis. It will be shown that m_0 is a sufficiently large number to allow us to establish that the distribution over suffixes in the multiset that begin at state q is likely to approximate the true distribution D_q over suffixes at that state. N is the number of (i.i.d.) strings in the sample generated by the algorithm. Polynomial expressions for m_0 and N are given in Algorithm 7.

We show that the probability of Algorithm 7 failing to adequately learn the structure of the automaton is upper bounded by δ'. In Section 5.5 we show that the transition probabilities are learnt (with sufficient accuracy for our purposes) by Algorithm 8 with a failure probability of at most δ''. Overall, the probability of the algorithms failing to learn the target PDFA within a variation distance of ϵ is at most δ, for $\delta = \delta' + \delta''$.

Algorithm 7 differs from [10] as follows. We do not introduce a "ground node" – a node to catch any undefined transitions in the hypothesis graph so as to give a probability greater than zero to the generation of any string. Instead, any state q for which $D_A(q) < \frac{\epsilon}{4n|\Sigma|}$ can be discarded – no corresponding node is formed in our hypothesis graph. There is only a small probability that our hypothesis automaton rejects a random string generated by D_A (when there is no corresponding path through the graph), which means that the contribution to the overall variation distance is very small. This is in contrast to the KL distance, which would become infinite.

Algorithm 7 *Construct Automaton.*

Hypothesis Graph $G = \langle V, E \rangle = \langle \{q_0\}, \emptyset \rangle$
$m_0 = (16/\mu)^2(\log(32/\delta'\mu) + \log(n|\Sigma|) + n|\Sigma|)$
$N = \max\left(\frac{8n^2|\Sigma|^2}{\epsilon^2} \ln\left(\frac{2^{n|\Sigma|}n|\Sigma|}{\delta'}\right), \frac{8m_0 n|\Sigma|}{\epsilon}\right)$
generate a sample S of N strings iid from D_A
repeat
  for each $v \in V$, $\sigma \in \Sigma$, where $\tau_G(v, \sigma)$ is undefined
    create a candidate node $\bar{q}_{v,\sigma}$ with associated multiset $S_{v,\sigma} = \emptyset$
  for each string $s \in S$, where $s = r\sigma' t$ and $\bar{q}_{\tau_G(q_0,r),\sigma'}$ is a candidate node
    $S_{\tau(q_0,r),\sigma'} \leftarrow S_{\tau(q_0,r),\sigma'} \cup \{t\}$
  identify candidate node $\bar{q}_{u,\sigma''}$ with the largest multiset, $S_{u,\sigma''}$
  if $\left(|S_{u,\sigma''}| \geq m_0\right)$ % *candidate node has large enough multiset*
    if $\left(\exists v \in V : \hat{L}_\infty\left(D_{\bar{q}_{u,\sigma''}}, D_v\right) \leq \frac{\mu}{2}\right)$ % *candidate "looks like" existing node*
      add edge (u, σ'', v) to E
    else
      add node $\bar{q}_{u,\sigma''}$ to V, with multiset $S_{u,\sigma''}$
      add edge $(u, \sigma'', \bar{q}_{u,\sigma''})$ to E
until$(|S_{u,\sigma''}| < m_0)$ % *no candidate node has large enough multiset*
return G

Figure 5.1: Constructing the underlying graph

Note that in contrast to the previous version of this algorithm in [39], and the algorithm of [10], we make a single sample at the beginning of the algorithm and we use the whole sample at each iteration. The trade-off is that by re-using the same sample at each iteration, we need a much lower failure probability (or higher reliability). It turns out that the total sample-size is about the same, but the algorithm is simpler and corresponds with the natural way one would treat real-world data.

5.4 Analysis of PDFA Construction Algorithm

The initial state $\bar{q}_0$ of H corresponds to the initial state q_0 of A. Each time a new state $\bar{q}_{v,\sigma}$ is added to H, its corresponding state in A is (with high probability) $\tau(q_v, \sigma)$. (Note that q_v in H already has a corresponding state in A.) We show that there is a one to one correspondence and that we reproduce a subgraph of A. We claim that at

every iteration of the algorithm, with high probability a bijection Φ exists between the states of H and candidate states, and a subset of the states of A, such that $\tau_A(u, \sigma) = v \Leftrightarrow \tau_H(\Phi(u), \sigma) = \Phi(v)$.

5.4.1 Recognition of Known States

We start by showing that with high probability, candidate states are correctly identified as being either unseen so far, or the same as a pre-existing state in the hypothesis. This part exploits the fact that the target automaton is known to have a minimum degree of distinguishability between all pairs of states.

Proposition 31 *Let D be a distribution over a countable domain. Let δ and μ be positive probabilities. Suppose we draw a sample S of $(16/\mu)^2 \log(16/\delta\mu)$ observations of D. Let $\hat{D}$ be the empirical distribution, i.e. the uniform distribution over multiset S. Then with probability $1 - \delta$, $L_\infty(D, \hat{D}) < \frac{1}{4}\mu$.*

Proof: Let $X = \{x_1, x_2, \ldots\}$ be the domain. Associate x_i with the interval $I_i = [\sum_{j<i} \Pr(x_j), \sum_{j \leq i} \Pr(x_j)]$. Let U_1 denote the uniform distribution over the unit interval; a point drawn from U_1 selects x_i with probability $\Pr(x_i)$.

Suppose $k \in \mathbf{N}$, $k \leq 16/\mu$. We identify a sufficiently large size for a sample S from U_1 such that with probability at least $1 - (\delta\mu/16)$, the proportion of points in S that lie in $[0, k(\mu/16)]$, is within $\mu/16$ of $k(\mu/16)$. By Hoeffding's inequality it is sufficient that $m = |S|$ satisfies

$$\frac{\delta\mu}{16} \geq 2\exp\Big(-2m\Big(\frac{\mu}{16}\Big)^2\Big).$$

That is satisfied by

$$m \geq \Big(\frac{16}{\mu}\Big)^2 \log\Big(\frac{16}{\delta\mu}\Big).$$

Furthermore, by a union bound we can deduce that with probability at least $1 - \delta$, for all $k \in \{0, 1, \ldots, 16/\mu\}$, the proportion of points in $[0, k(\mu/16)]$ is within $\mu/16$ of expected value. This implies that for all intervals, including the I_i intervals, the proportion of points in those intervals is within $\mu/4$ of expected value. □

The following result shows that given any partially-constructed DFA, a candidate state is correctly identified with very high probability, using a sample of size $m_0 = (16/\mu)^2(\log(32/\delta'\mu) + \log(n|\Sigma|) + n|\Sigma|)$.

Proposition 32 *Let G be a DFA with transition function τ_G whose vertices and edges are a subgraph of the underlying DFA for PDFA A. Suppose D_A is repeatedly sampled,*

and we add s_2 to $S_{q,\sigma}$ whenever we obtain a string of the form $s_1\sigma s_2$, where $\tau_G(s_1)$ is state q of G.

If $|S_{q,\sigma}| \geq m_0$, then with probability at least $1 - \delta'(n|\Sigma|2^{n|\Sigma|+1})^{-1}$, $\hat{L}_\infty(S_{q,\sigma}, D_{q,\sigma}) < \mu/4$.

Proof: Given any G, strings s_2 obtained in this way are all sampled independently from $D_{q,\sigma}$.

Proposition 31 shows that a sufficiently large sample size is given by

$$\Big(\frac{16}{\mu}\Big)^2 \log\Big(\frac{16}{\delta'\mu(n|\Sigma|2^{n|\Sigma|+1})^{-1}}\Big) = \Big(\frac{16}{\mu}\Big)^2\Big(\log\Big(\frac{32}{\delta'\mu}\Big) + \log(n|\Sigma|) + n|\Sigma|\Big).$$

□

The following result applies a union bound to verify that whatever stage we reach at an iteration, and whatever candidate state we examine, the algorithm is unlikely to make a mistake.

Proposition 33 *With probability δ', for all candidate nodes $\bar{q}_{u,\sigma''}$ found by the algorithm, $\bar{q}_{u,\sigma''}$ is added to G such that G continues to be a subgraph of the PDFA for A.*

Proof: Proposition 31 and the metric property of L_∞ show that if distributions D_v associated with states v are empirically estimated to within L_∞ distance $\mu/4$, then our threshold of $\mu/2$ that is used to distinguish a pair of states, ensures that no mistake is made.

There are at most $2^{n|\Sigma|}$ possible subgraphs G and at most $n|\Sigma|$ candidate nodes for any subgraph. If the probability of failure is at most $\delta'(n|\Sigma|2^{n|\Sigma|+1})^{-1}$ for any single combination of G and candidate node, then by a union bound and Proposition 32, the probability of failure is at most $\delta'/2$. □

We have ensured that m_0 is large enough that with high probability the algorithm does not

- identify two distinct nodes with each other, or
- fail to recognise a candidate node as having been seen already.

5.4.2 Ensuring that the DFA is Sufficiently Complete

Next we have to check that it does not "give up too soon", as a result of not seeing m_0 samples from a state that really should be included in G.

Proposition 34 *Let A' be a PDFA whose states and transitions are a subset of those of A. Assume A' contains the initial state q_0. Suppose q is a state of A' but (q,σ) is not a transition of A'. Let S be a sample from D_A, $|S| \geq (8n^2|\Sigma|^2/\epsilon^2)\ln(2^{n|\Sigma|}n|\Sigma|/\delta')$. Let $S_{q,\sigma}(A')$ be the number of elements of S of the form $s_1\sigma s_2$ where $\tau(q_0,s_1)=q$ and for all prefixes s_1' of s_1, $\tau(q_0,s_1')\in A'$. Then*

$$\Pr\left(\left|\left(\frac{S_{q,\sigma}(A')}{|S|}\right) - E\left[\frac{S_{q,\sigma}(A')}{|S|}\right]\right| \geq \frac{\epsilon}{8n|\Sigma|}\right) \leq \frac{\delta'}{2^{n|\Sigma|}n|\Sigma|}.$$

Proof: From Hoeffding's Inequality it can be seen that

$$\Pr\left(\left|\left(\frac{S_{q,\sigma}(A')}{|S|}\right) - E\left[\frac{S_{q,\sigma}(A')}{|S|}\right]\right| \geq \frac{\epsilon}{8n|\Sigma|}\right) \leq 2\exp\left(-2|S|\left(\frac{\epsilon}{4n|\Sigma|}\right)^2\right). \quad (5.1)$$

We need $|S|$ to satisfy $\exp(-|S|\epsilon^2(8n^2|\Sigma|^2)^{-1}) \leq \delta'(2^{n|\Sigma|}n|\Sigma|)^{-1}$. Equivalently,

$$\frac{8n^2|\Sigma|^2}{\epsilon^2}\ln\left(\frac{2^{n|\Sigma|}n|\Sigma|}{\delta'}\right) \leq |S|.$$

So the sample size identified in the statement is indeed sufficiently large. □

The following result shows that the algorithm constructs a subset of the states and transitions that with high probability accepts a random string from D_A.

Theorem 35 *There exists T' a subset of the transitions of A, and Q' a subset of the states of A, such that $\sum_{(q,\sigma)\in T'} D_A(q,\sigma) + \sum_{q\in Q'} D_A(q) \leq \frac{\epsilon}{2}$, and with probability at least $1-\delta'$, every transition $(q,\sigma)\notin T'$ in target automaton A has a corresponding transition in hypothesis automaton H, and every state $q\notin Q'$ in target automaton A has a corresponding state in hypothesis automaton H.*

Proof: Proposition 33 shows that the probability of all candidate nodes having "good" multisets (if the multisets contain at least m_0 suffixes) is at least $1-\delta'/2$, from which we can deduce that all candidate nodes can be correctly distinguished from any nodes[4] in the hypothesis automaton.

Proposition 34 shows that with a probability of at least $1-\delta'(2^{n|\Sigma|}n|\Sigma|)^{-1}$, the proportion of strings in a sample S (generated i.i.d. over D_A, and for $|S| \geq (8n^2|\Sigma|^2/\epsilon^2)\ln(2^{n|\Sigma|}n|\Sigma|/\delta'))$ reaching candidate node $\bar{q}$ is within $\epsilon(8n|\Sigma|)^{-1}$ of the expected proportion $D_A(\bar{q})$. This holds for each of the candidate nodes (of which there are at most $n|\Sigma|$), and for each possible state of the hypothesis graph in terms of

[4]Note that due to the deterministic nature of the automaton, distinguishability of transitions is not an issue.

the combination of edges and nodes found (of which there are at most $2^{n|\Sigma|}$), with a probability of at least $1 - \delta'/2$.

If a candidate node (or a *potential candidate node*[5]) $\bar{q}$, for which $D_A(\bar{q}) \geq \epsilon(4n|\Sigma|)^{-1}$, is not included in H, then from the facts above it follows that at least $\epsilon N(8n|\Sigma|)^{-1}$ strings in the sample are not accepted by the hypothesis graph. For each string not accepted by H, a suffix is added to the multiset of a candidate node, and there are at most $n|\Sigma|$ such candidate nodes. From this it can be seen that some candidate node has a multiset containing at least $\frac{1}{8}\epsilon N$ suffixes. From the definition of N, $N \geq (8m_0 n|\Sigma|/\epsilon)$. Therefore, some multiset contains at least $m_0 n|\Sigma|$ suffixes, which must be at least as great as m_0. This means that as long as there exists some significant transition or state that has not been added to the hypothesis, some multiset must contain at least m_0 suffixes, so the associated candidate node will be added to H, and the algorithm will not halt.

Therefore it has been shown that all candidate nodes which are significant enough to be required in the hypothesis automaton (at least a fraction $\epsilon(4n|\Sigma|)^{-1}$ of the strings generated reach the node) are present with a probability of at least $1 - \frac{1}{2}\delta'$, and that since all multisets contain at least m_0 suffixes, the candidate nodes and hypothesis graph nodes are all correctly distinguished from each other (or combined as appropriate) with a probability of at least $1 - \frac{1}{2}\delta'$.

T' is those transitions that have probability less than $\epsilon/4n|\Sigma|$ of being used by a random string, and there can be at most $n|\Sigma|$ such transitions. Hence a random string uses an element of T' with probability at most $\frac{1}{4}\epsilon$. We conclude that with a probability of at least $1 - \delta'$, every transition $(q, \sigma) \notin T'$ in target automaton A for which $D_A(q, \sigma) \geq \epsilon(4n|\Sigma|)^{-1}$ and every state $q \notin Q'$ in target automaton A for which $D_A(q) \geq \epsilon(4n|\Sigma|)^{-1}$, has a corresponding transition or state in hypothesis automaton H. □

5.5 Finding Transition Probabilities

The algorithm is shown in Figure 5.2. We can assume that we have at this stage found DFA H, whose graph is a subgraph of the graph of target PDFA A. Algorithm 8 finds estimates of the probabilities $\gamma(q, \sigma)$ for each state q in H, $\sigma \in \Sigma$.

If we generate a sample S from D_A, we can trace each $s \in S$ through H, and each visit to a state $q_H \in H$ provides an observation of the distribution over the transitions that leave the corresponding state q_A in A. For string $s = \sigma_1\sigma_2 \ldots \sigma_\ell$, let q_i be the state reached by the prefix $\sigma_1 \ldots \sigma_{i-1}$. The probability of s is $D_A(s) = \prod_{i=0}^{\ell-1} \gamma(q_i, \sigma_{i+1})$.

[5] A potential candidate node is any state or transition in the target automaton which has not yet been added to H, and is not currently represented by a candidate node.

Letting $n_{q,\sigma}(s)$ denote the number of times that string s uses transition (q,σ), then

$$D_A(s) = \prod_{q,\sigma} \gamma(q,\sigma)^{n_{q,\sigma}(s)}. \tag{5.2}$$

Let $\hat{\gamma}(q,\sigma)$ denote the estimated probability that is given to transition (q,σ) in H. Provided H accepts s, the estimated probability of string s is given by

$$D_H(s) = \prod_{q,\sigma} \hat{\gamma}(q,\sigma)^{n_{q,\sigma}(s)}. \tag{5.3}$$

We aim to ensure that with high probability for $s \sim D_A$, if H accepts s (i.e. if s does not visit a node that has been omitted from the hypothesis) then the ratio $D_H(s)/D_A(s)$ is close to 1. This is motivated by the following observation.

Observation 36 *Suppose that with probability* $1-\frac{1}{4}\epsilon$ *for* $s \sim D_A$*, we have* $D_H(s)/D_A(s) \in [1-\frac{1}{4}\epsilon, 1+\frac{1}{4}\epsilon]$*. Then* $L_1(D_A, D_H) \leq \epsilon$*.*

Proof:

$$L_1(D_A, D_H) = \sum_{s\in\Sigma^*} |D_A(s) - D_H(s)|$$

Let $X = \{s \in \Sigma^* \ : \ D_H(s)/D_A(s) \in [1-\frac{1}{4}\epsilon, 1+\frac{1}{4}\epsilon]\}$. Then

$$L_1(D_A, D_H) = \sum_{s\in X} |D_A(s) - D_H(s)| + \sum_{s\in\Sigma^*\setminus X} |D_A(s) - D_H(s)| \tag{5.4}$$

The first term of the right-hand side of Equation 5.4 is

$$\sum_{s\in X} D_A(s)\Big|(1 - D_H(s)/D_A(s))\Big| \leq \sum_{s\in X} D_A(s).\Big(\frac{\epsilon}{4}\Big) \leq \frac{\epsilon}{4}.$$

$D_A(X) \geq 1-\frac{1}{4}\epsilon$ and $D_H(X) \geq D_A(X) - \frac{1}{4}\epsilon$, equivalently $D_A(\Sigma^* \setminus X) \leq \frac{1}{4}\epsilon$ and $D_H(\Sigma^* \setminus X) \leq D_A(\Sigma^* \setminus X) + \frac{1}{4}\epsilon \leq \frac{1}{2}\epsilon$, hence the second term in the right-hand side of Equation 5.4 is at most $\frac{3}{4}\epsilon$. □

We have so far allowed the possibility that H may fail to accept up to a fraction $\frac{1}{4}\epsilon$ of strings generated by D_A. Of the strings s that are accepted by H, we want to ensure that with high probability $D_H(s)/D_A(s)$ is close to 1, to allow Observation 36 to be used.

5.5.1 Correlation Between a Transition's Usage and the Accuracy of its Estimated Probability

Suppose that $n_{q,\sigma}(s)$ is large, so that s uses transition (q,σ) a large number of times. In that case, errors in the estimate of transition probability $\gamma(q,\sigma)$ can have a disproportionately large influence on the ratio $D_H(s)/D_A(s)$. What we show is that with high probability for random $s \sim D_A$, regardless of how many times transition (q,σ) typically gets used, the training sample contains a large enough subset of strings that use that transition more times than s does, so that $\gamma(q,\sigma)$ is nevertheless known to a sufficiently high precision.

We say that $s' \in \Sigma^*$ is (q,σ)-good for some transition (q,σ), if s' satisfies

$$\Pr_{s\sim D_A}(n_{q,\sigma}(s) > n_{q,\sigma}(s')) \leq \frac{\epsilon}{4n|\Sigma|}.$$

Informally, a (q,σ)-good string is one that is more useful than most in providing an estimate of $\gamma(q,\sigma)$.

Proposition 37 *Let $m \geq 1$. Let S be a sample from D_A, $|S| \geq m(32n|\Sigma|/\epsilon)\ln(2n|\Sigma|/\delta'')$. With probability $1-\delta''(2n|\Sigma|)^{-1}$, for transition (q,σ) there exist at least $\epsilon(8n|\Sigma|)^{-1}|S|$ (q,σ)-good strings in S.*

Proof: From the definition of (q,σ)-good, the probability that a string generated at random over D_A is (q,σ)-good for transition (q,σ), is at least $\epsilon(4n|\Sigma|)^{-1}$.[6]

Applying a standard Chernoff bound (see [4], p.360), for any transition (q,σ), with high probability over samples S, the number of (q,σ)-good strings in S is at least half the expected number as follows.

$$\Pr\left(|\{s\in S \ : \ s \text{ is } (q,\sigma)\text{−good}\}| < \frac{1}{2}\left(\frac{\epsilon}{4n|\Sigma|}|S|\right)\right) \leq \exp\left(-\frac{1}{8}\left(\frac{\epsilon}{4n|\Sigma|}\right)|S|\right). \tag{5.5}$$

We wish to bound this probability to be at most $\delta''(2n|\Sigma|)^{-1}$, so from Equation 5.5,

$$\begin{aligned}\exp\left(-\frac{1}{8}\left(\frac{\epsilon}{4n|\Sigma|}\right)|S|\right) &\leq \frac{\delta''}{2n|\Sigma|} \\ |S| &\geq \left(\frac{32n|\Sigma|}{\epsilon}\right)\ln\left(\frac{2n|\Sigma|}{\delta''}\right)\end{aligned}$$

[6] The probability that a random string $s \sim D_A$ uses the transition (q,σ) *more* times than a (q,σ)-good string is at most $\epsilon(4n|\Sigma|)^{-1}$, so the probability of generating a (q,σ)-good string must be at least that much.

which is indeed satisfied by the assumption in the statement. □

Notation. Suppose S is as defined in Algorithm 8. Let $M_{q,\sigma}(S)$ be the largest number with the property that at least a fraction $\epsilon(8n|\Sigma|)^{-1}$ of strings in S use (q,σ) at least $M_{q,\sigma}(S)$ times.

Informally, $M_{q,\sigma}(S)$ represents a "big usage" of transition (q,σ) by a random string — the fraction of elements of S that use (q,σ) more than $M_{q,\sigma}(S)$ times is less than $\epsilon(8n|\Sigma|)^{-1}$. The next observation states that $M_{q,\sigma}$ is likely to be an over-estimate of the number of uses of (q,σ) required for (q,σ)-goodness.

Observation 38 *For any (q,σ), with probability $1-\delta''(2n|\Sigma|)^{-1}$ (over random samples S with $|S|$ as given in the algorithm),*

$$\Pr_{s\sim D_A}(n_{q,\sigma}(s) > M_{q,\sigma}(S)) \leq \frac{\epsilon}{4n|\Sigma|}. \tag{5.6}$$

Proof: This follows from Proposition 37 (plugging in $m = (2n|\Sigma|/\delta'')(64n|\Sigma|/\epsilon\delta'')^2$). □

Algorithm 8 *Finding Transition Probabilities.*

Input: DFA H, a subgraph of A.

generate sample S from D_A; $|S| = \left(\frac{2n|\Sigma|}{\delta''}\right)\left(\frac{64n|\Sigma|}{\epsilon\delta''}\right)^2\left(\frac{32n|\Sigma|}{\epsilon}\right)\ln\left(\frac{2n|\Sigma|}{\delta''}\right)$;
for each state $q \in H$, $\sigma \in \Sigma$:
  repeat
    for strings $s \in S$, trace paths through H;
    let $N_{q,-\sigma}$ be random variable: number of observations of state q
    up to and including the next observation of transition (q,σ)
    (include observations of q and (q,σ) in rejected strings).
  until(all strings in S have been traced)
  let $\hat{\mu}(N_{q,-\sigma})$ be the mean of the observations of $N_{q,-\sigma}$;
  let $\hat{\gamma}(q,\sigma) = 1/\hat{\mu}(N_{q,-\sigma})$.
for each $q \in H$, rescale $\hat{\gamma}(q,\sigma)$ such that $\sum_{\sigma\in\Sigma}\hat{\gamma}(q,\sigma) = 1$.

Figure 5.2: Finding Transition Probabilities

5.5.2 Proving the Accuracy of the Distribution over Outputs

Theorem 39 *Suppose that H is a DFA that differs from A by the removal of a set of transitions that have probability at most $\frac{1}{2}\epsilon$ of being used by $s \sim D_A$. Then Algorithm 8 assigns probabilities $\hat{\gamma}(q,\sigma)$ to the transitions of H such the resulting distribution D_H satisfies $L_1(D_A, D_H) < \epsilon$, with probability $1 - \delta''$.*

Proof: Recall Observation 38, that with probability $1 - \delta''(2n|\Sigma|)^{-1}$,

$$\Pr_{s\sim D_A}(n_{q,\sigma}(s) > M_{q,\sigma}(S)) \leq \frac{\epsilon}{4n|\Sigma|}.$$

By definition of $M_{q,\sigma}(S)$, at least $|S|\epsilon(8n|\Sigma|)^{-1} > (2n|\Sigma|/\delta'')(64n|\Sigma|/\epsilon\delta'')^2$ members of $S_{q,\sigma}$ use (q,σ) at least $M_{q,\sigma}(S)$ times. Hence for any (q,σ), with probability $1 - \delta''(2n|\Sigma|)^{-1}$, there are $M_{q,\sigma}(S)(2n|\Sigma|/\delta'')(64n|\Sigma|/\epsilon\delta'')^2$ uses of transition (q,σ).

Consequently, (again with probability $1 - \delta''(2n|\Sigma|)^{-1}$ over random choice of S), for any (q,σ) the set S generates a sequence of independent observations of state q, which continues until at least $M_{q,\sigma}(S)(2n|\Sigma|/\delta'')(64n|\Sigma|/\epsilon\delta'')^2$ of them resulted in transition (q,σ).

Let $N_{q,-\sigma}$ denote the random variable which is the number of times q is observed before transition (q,σ) is taken. Each time state q is visited, the selection of the next transition is independent of previous history, so we obtain a sequence of independent observations of $N_{q,-\sigma}$. So, with probability $1-\delta''(2n|\Sigma|)^{-1}$, the number of observations of $N_{q,-\sigma}$ is at least $M_{q,\sigma}(S)(2n|\Sigma|/\delta'')(64n|\Sigma|/\epsilon)^2$.

Recall Chebyshev's inequality, that for random variable X with mean μ and variance σ^2, for positive k,

$$\Pr(|X - \mu| > k) \leq \frac{\sigma^2}{k^2}.$$

$N_{q,-\sigma}$ has a discrete exponential distribution with mean $\gamma(q,\sigma)^{-1}$ and variance $\leq \gamma(q,\sigma)^{-2}$. Hence the empirical mean $\hat{\mu}(N_{q,-\sigma})$ is a random variable with mean $\gamma(q,\sigma)^{-1}$ and variance at most $\gamma(q,\sigma)^{-2}(M_{q,\sigma})^{-1}(2n|\Sigma|/\delta'')^{-1}(64n|\Sigma|/\epsilon\delta'')^{-2}$. Applying Chebyshev's inequality with $\hat{\mu}(N_{q,-\sigma})$ for X, and
$k = \gamma(q,\sigma)^{-1}\epsilon\delta''(64n|\Sigma|\sqrt{M_{q,\sigma}})^{-1}$, we have

$$\Pr\left(|\hat{\mu}(N_{q,-\sigma}) - \gamma(q,\sigma)^{-1}| > \gamma(q,\sigma)^{-1}\left(\frac{\epsilon\delta''}{64n|\Sigma|\sqrt{M_{q,\sigma}}}\right)\right) \leq \frac{\delta''}{2n|\Sigma|}.$$

Note that for $x, y > 0$ and $\frac{1}{2} > \xi > 0$, if $|y - x| < x\xi$ then $|y^{-1} - x^{-1}| < 2x^{-1}\xi$, and applying this to the left-hand side of the above, we deduce

$$\Pr\left(|\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| > 2\gamma(q,\sigma)\left(\frac{\epsilon\delta''}{64n|\Sigma|\sqrt{M_{q,\sigma}}}\right)\right) \leq \frac{\delta''}{2n|\Sigma|}.$$

The rescaling at the end of Algorithm 8 (which may be needed as a result of infrequent transitions not being included in the hypothesis automaton) loses a factor of at most 2 from the upper bound on $|\gamma(q,\sigma) - \hat{\gamma}(q,\sigma)|$. Overall, with high probability $1 - \delta''(2n|\Sigma|)^{-1}$,

$$|\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \leq \left(\frac{\epsilon\delta''\gamma(q,\sigma)}{16n|\Sigma|\sqrt{M_{q,\sigma}}} \right). \tag{5.7}$$

For $s \in \Sigma^*$ let $n_q(s)$ denote the number of times the path of s passes through state q. By definition of $M_{q,\sigma}(S)$, for any transition (q,σ) with high probability $1 - \epsilon(4n|\Sigma|)^{-1}$,

$$E_{s\sim D_A}[n_q(s)] < M_{q,\sigma}(S)/\gamma(q,\sigma). \tag{5.8}$$

For $s \sim D_A$ we upper bound the expected log-likelihood ratio,

$$\log\left(\frac{D_H(s)}{D_A(s)}\right) = \sum_{i=1}^{|s|} \log\left(\frac{\hat{\gamma}(q_i,\sigma_i)}{\gamma(q_i,\sigma_i)}\right),$$

where σ_i is the i-th character of s and q_i is the state reached by the prefix of length $i-1$.

Suppose A generates a prefix of s and reaches state q. Let random variable X_q be the contribution to $\log(D_H(s)/D_A(s))$ when A generates the next character.

$$\begin{aligned} E[X_q] &= \sum_\sigma \gamma(q,\sigma) \log\left(\frac{\hat{\gamma}(q,\sigma)}{\gamma(q,\sigma)}\right) \\ &= \sum_\sigma \gamma(q,\sigma)[\log(\hat{\gamma}(q,\sigma)) - \log(\gamma(q,\sigma))]. \end{aligned}$$

For $|\xi| \ll x$ and some $|\alpha| < 2$, it is the case that $\log(x+\xi) - \log(x) = \xi x^{-1}(1 + \alpha\xi/x)$. Using this fact and plugging in $\gamma(q,\sigma)$ for x, then from Equation 5.7 we claim that (with high probability $1 - \delta''(2n|\Sigma|)^{-1}$):

$$\log(\hat{\gamma}(q,\sigma)) - \log(\gamma(q,\sigma)) = |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \left(\frac{1}{\gamma(q,\sigma)}\right) A_{q,\sigma} \tag{5.9}$$

for some $A_{q,\sigma} \in [1 - \epsilon\delta''(8n|\Sigma|\sqrt{M_{q,\sigma}})^{-1}, 1 + \epsilon\delta''(8n|\Sigma|\sqrt{M_{q,\sigma}})^{-1}]$.

Consequently,

$$\begin{aligned} E[X_q] &= \sum_\sigma \gamma(q,\sigma) \left(\frac{1}{\gamma(q,\sigma)}\right) A_{q,\sigma} |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \\ &= \sum_\sigma A_{q,\sigma} |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \\ &= \sum_\sigma |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| + \sum_\sigma B_{q,\sigma} |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \end{aligned}$$

for some $B_{q,\sigma} \in [-\epsilon\delta''(8n|\Sigma|\sqrt{M_{q,\sigma}})^{-1}, \epsilon\delta''(8n|\Sigma|\sqrt{M_{q,\sigma}})^{-1}]$. The first term vanishes, so we have

$$\begin{aligned} E[X_q] &= \sum_{\sigma} B_{q,\sigma} |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \\ &= \frac{\epsilon\delta''}{8n|\Sigma|} \sum_{\sigma} \left(\frac{1}{\sqrt{M_{q,\sigma}}} \right) |\hat{\gamma}(q,\sigma) - \gamma(q,\sigma)| \\ &\leq \frac{\epsilon\delta''}{8n|\Sigma|} \sum_{\sigma} \frac{\gamma(q,\sigma)}{M_{q,\sigma}} \end{aligned}$$

where the last inequality uses Equation 5.7. For $s \sim D_A$, the expected contribution to $\log(D_H(s)/D_A(s))$ from all $n_q(s)$ usages of state q is, using Equation 5.8, at most

$$E[n_q(s)] \left(\frac{\epsilon\delta''}{8n|\Sigma|} \right) \sum_{\sigma} \frac{1}{E[n_q(s)]} = \left(\frac{\epsilon\delta'' n_q(s)}{8n|\Sigma|} \right) |\Sigma| \left(\frac{1}{n_q(s)} \right) = \frac{\epsilon\delta''}{8n}.$$

The total expected contribution from all n states q, each being used $n_q(s)$ times is

$$\sum_{q \in Q} \frac{\epsilon\delta''}{8n} = \frac{\epsilon\delta''}{8}. \tag{5.10}$$

Using Markov's inequality, there is a probability at most δ'' that $\log(D_H(s)/D_A(s))$ is more than $\epsilon/8$.

Finally, in order to use Observation 36, note that $(D_H(s)/D_A(s)) \in \left[1 - \frac{1}{4}\epsilon, 1 + \frac{1}{4}\epsilon\right]$ follows from $\log(D_H(s)/D_A(s)) \in \left[-\frac{1}{8}\epsilon, \frac{1}{8}\epsilon\right]$. □

5.5.3 Running Algorithm 8 in $\log(1/\delta'')$ rather than $poly(1/\delta'')$

The sample size expression is polynomial in $1/\delta''$, as is necessary for a PAC algorithm. However, this expression can be converted into one that is logarithmic in $1/\delta''$ as follows. If we run the algorithm x times using $\delta'' = \frac{1}{10}$, we obtain x values for the likelihood of a string, rather than just one. It is not hard to show that for $x = O(\log(1/\delta''))$, the median will be accurate with probability $1 - \delta''$.

5.6 Main Result

We can now put the two algorithms together using any values of δ' and δ'' that add up to at most δ (δ being the overall uncertainty bound). By combining the results of Theorem 35 and Theorem 39, we get the following.

Theorem 40 *Given an automaton with alphabet Σ and at most n states, which is μ-distinguishable for some parameter μ, then Algorithm 1 and Algorithm 2 run in time polynomial in the above parameters (also ϵ and δ), producing a model which with probability at least $1-\delta$ differs (in L_1 distance) from the original automaton by at most ϵ.*

The algorithms are structurally similar to previous algorithms for learning PDFAs. One change worth noting that we have made, is that for each algorithm a single sample is taken at the beginning, and all elements of that sample are treated the same way. Previous related algorithms (including [39]) typically draw a sample at each iteration, so as to ensure independence between iterations. In practice it is natural and realistic to assume that every measurement is extracted from all the data.

We have shown that as a result of using the variation distance as a criterion for precise learning, we can obtain sample-size bounds that do not involve the length of strings generated by unknown PDFAs. In the appendix we show why the KL-divergence requires a limit on the expected length of strings that the target automaton generates (see Section B.1). Furthermore, this approach has addressed the issue of extracting more information from long strings than short strings, which is necessary in order to estimate heavily-used transitions with higher precision.

5.7 Smoothing from L_1 Distance to KL-Divergence

Consider the problem of learning PDFAs having n states, over finite alphabet Σ, and probabilities represented by bit strings of length ℓ. Using sample size (but not time) polynomial in n, $|\Sigma|$ and ℓ (and the PAC parameters ϵ and δ), a distribution over this class can be estimated within KL-divergence ϵ. The proof follows from Lemma 22 in Chapter 2, and from the observation that such a PDFA can be represented using a bit string whose length is polynomial in the parameters.

Consequently we can learn the same class of PDFAs under the KL-divergence that can be learned under the L_1 distance, i.e. PDFAs with distinguishable states but no restriction on the expected length of their outputs. However, note that the hypothesis is "inefficient" (using a mixture of exponentially many PDFAs).

See Section B.2 for further details of this.

Chapter 6

Conclusion

To conclude we give a summary of the results contained in this thesis. We shall examine the significance of these results and, where relevant, put them in the context of related work. We will then discuss the questions stated in the introduction – namely whether there is a benefit to learning with unsupervised learners, whether it is harder to learn with unsupervised learners, and whether we can draw conclusions about the equivalence of learning in this framework to classical PAC-learning.

6.1 Summary of Results

Chapter 2

In this chapter we gave results in the agnostic PAC learning framework bounding the accuracy and confidence of a PAC classifier to the accuracy of the distributions over class labels. We showed that if the distributions over k class labels are learnt within L_1 distance of ϵ/g_ℓ of their targets (where ℓ is the label of the respective distribution) then the associated Bayes classifier is accurate within $\epsilon.k.\max\{c\}$ of the optimal classifier ($\max\{c\}$ being the largest cost in the cost matrix). This use of the maximum cost is an upper bound and this was used to provide generality, but the term could be tightened with the result of adding complexity to the expression. It is also shown that if the class distributions are learnt with respect to KL-divergence, a regret of $k\epsilon$ upper bounds the additional risk. In addition to this matching lower bounds are given by way of example distributions.

It is then shown that for a class $\mathcal{D}$ of distributions specified on a finite discrete scale, distributions learnt under L_1 distance in polynomial time can be learnt under KL-divergence in polynomial time. We go on to give a demonstration of a setting in which this can be applied in Chapter 5.

Chapter 3

Following from the previous chapter we demonstrate the use of unsupervised learners to achieve optical digit recognition. We show that adequate results can be gained from an algorithm using estimates of distributions over class labels in the extension of the PAC-learning framework as described in Chapter 2. These results are comparable with the widely used method of k-nearest neighbours – a discriminative algorithm. Since no preprocessing was performed on the images it is reasonable to assume that the results obtained in our experiments could be considerably enhanced (possibly halving the achieved error rate) by applying such techniques as have been studied elsewhere [11].

When the technique is applied to strings of digits with an aspect of context sensitivity, the generative technique is shown to perform well, exhibiting the expected level of error correction. The results indicate that it is worthwhile to perform such generative techniques in order to be able to correct errors in the recognition process with some degree of success.

Chapter 4

We show that the class of probabilistic concepts consisting of functions with at most k turning points can be PAC-learnt in the sense of [31] using a weak generative algorithm. It is conjectured that the problem cannot be solved in the strong generative sense of estimating distributions over class labels, and as such we give an algorithm that provides an alternative discriminant function for each label.

For the analysis in Section 4.4 to hold, we imposed a restriction in the statement of the problem that the class priors of label 0 and label 1 must each be $\frac{1}{2}$. In learning problems of this nature there is often a restriction of this kind placed upon the learning framework, or at least an assumption that the priors are known beforehand. This restriction allows greater simplicity of analysis, but may be seen as a limitation to the usefulness of the algorithm. However, it appears that the algorithm will work equally well given pairs of labels with different prior probabilities but to see this requires more complex analysis.

Consider the following adaptation of the algorithm, whereby rather than generating two samples of m observations from which to learn, we generate samples containing m_0 and m_1 observations of labels 0 and 1 respectively. Given class priors g_0 and g_1 of the labels being generated by the oracle, where $g_1 = 1 - g_0$, we generate samples containing $\left(\frac{\max\{g_0,g_1\}}{\min\{g_0,g_1\}}\right) m$ observations of the label with the higher prior probability, and m observations of the label with the lower prior probability.

It appears that Theorem 24 still holds, although the method of showing this will differ slightly from the proof given. The additional complexity in the analysis comes

from the fact that when using Lemma 23 to show that a sample is representative of its distribution, one of the classes will have a higher expected number of observations per interval than the other. To get around this, it would be necessary to alter the lemma, such that the domain is divided into a larger number of intervals for the label with the higher prior, in order to keep the expected number of observations in the intervals the same for each class. Since the idea of the algorithm given here is to show that unsupervised learners can be used efficiently to solve the problem, it was deemed unnecessary to explore this further.

Chapter 5

It is shown that it is possible to PAC-learn PDFA in terms of the L_1 distribution between the estimated distribution over outputs and that representing the target automaton. This extends the work of Clark and Thollard [10] where the KL-divergence is used as a measure of proximity. We show how to dispense with the previous requirement of an upper bound on the expected length of a string by virtue of the fact that the weaker L_1 distance is used. We then show that for the class of PDFA capable of representation by bit strings (in other words the probabilities of transitions can be represented on a discrete finite scale), the output distribution can be smoothed such that it is good under KL-divergence but not in polynomial time.

In [24] the notion of μ-distinguishability is challenged, and a weaker notion of distinguishability termed μ_p-distinguishability is used to PAC-learn PDFA in terms of KL-divergence using a similar state-merging algorithm[1]. It is shown in [30] that PAC-learning of PDFA without distinguishability restrictions is hard – in that a subset of PDFA can encode noisy parity functions. It is conjectured in [25] that PAC-learning automata under L_1 distance suffers from the same problem, but learning μ_p-distinguishable PDFA does not.

As a further exploration of this problem, it would be interesting to investigate whether PDFAs are learnable without an a priori "distinguishability" of states. Given the research carried out on distinguishability of automata it is likely that any such results would involve restricting the algorithm to apply to specific subclasses of PDFA.

A further question concerns whether it is necessary to be given the number of states as a parameter of the problem – techniques exist for estimating such parameters for solving similar problems. To do so would involve a process of taking an initially small value of n and applying the algorithm, then testing the resulting hypothesis automata to see whether a sufficiently large proportion of strings are accepted by the resulting hypothesis graph. If not, the value of n is increased until the condition is met.

[1] A pair of states are said to be μ_p-distinguishable if the L_p distance between their suffix distributions is at least μ.

However, it appears that this problem is non-trivial since the lower the value of n is that is being tested, the higher the criteria for *significant* candidate nodes being included. So as n is halved, the proportion of random strings using a transition in order for it to be considered significant doubles. Therefore, there is a high probability that transitions are excluded from the resulting DFA which should be included. There is then a problem of how much data to generate in order to test the DFA to see what proportion of strings are accepted, which – in order to achieve any bounds on the confidence of success – must be polynomial in n. It seems that a variant of this method could well generate an automaton that in practice is close to the target automaton, but it is unlikely that such a method could claim any certainty of success.

6.2 Discussion

In general – is the method of classifying via unsupervised learners a worthwhile method? There are a number of reasons supporting the use of unsupervised learners for classification problems, particularly in the strong generative sense where the distributions over class labels are estimated. In terms of the accuracy with which classifications can be made, the results of Chapter 2 show that the classification error is linked strongly to the accuracy of the distribution estimation, and Chapter 3 shows that good practical results can be achieved this way.

The extension of the problem to circumstances where the a posteriori distribution over class labels is important shows that this additional information gives great benefits to the classification algorithm. Similar practical results [42] demonstrate that generative algorithms can outperform discriminative methods, particularly in cases where the data samples are relatively small. In situations such as those studied here where more than two classes are involved, benefits may be gained from the seemingly natural way in which classifications are made in contrast to finding class boundaries.

In addition to the results observed in terms of classification rates, a number of reasons have been mentioned as to why it may be beneficial to construct class distributions. The most fundamental of these include the fact that data of all classes is treated in the same way —there is no artificial ordering of data— and if a class is added to or removed from the problem then it is relatively simple to accommodate this change.

In [35] it is shown that a class of problems exists that can be solved by discriminative methods but not by generative methods. However, unlike the problems described in this thesis (and those in [22], [23]), the authors set the problem within the "Probably Approximately Bayes" framework (defined in [35]) in which the distributions over observations and labels need not be independent – indeed in the problem stated they are not. However, the result does indicate a certain weakness in the power of generative

learning for solving certain types of problems.

In terms of whether it is harder to learn with unsupervised learners than in the traditional PAC setting, it seems intuitive that the restrictions imposed on the problem ought to mean that it is. The fact that the empirical error is unknown to the learners, and that they have no knowledge of their class label ought to make it far tougher to solve standard problems. In fact there are currently no known problems that are PAC-learnable but not PAC-learnable via discriminant functions. The problem of learning unrestricted monomials with unsupervised learners is left as an open problem as there are no known positive or negative results to date.

Appendix A

Optical Digit Recognition

Results obtained from the practical testing of the algorithms in Chapter 3 are to be found within this appendix.

Table A.1 shows how many of each digit appear in the training data set and in the test data set.

Digit	Number in Training Data	Number in Test Data
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009
Total	60000	10000

Table A.1: Breakdown of image data sets into digit labels.

A.1 Distance Functions

A.1.1 L_2 Distance

The algorithm in Figure A.1 returns the L_2 distance between a pair of image vectors. The algorithm examines all pairs of corresponding pixels in the two images (p_i in image

Algorithm 9 *L_2 distance function.*

L_2(image I, image J)

let $d = 0$

for each pixel p_i in I
  let p_j be the corresponding pixel in J
  $d = d + |p_i - p_j|^2$

let $d_{L_2} = \sqrt{d}$

Return d_{L_2}

Figure A.1: Algorithm to compute the L_2 distance between 2 image vectors.

I corresponds to p_j in image J if the pixels have the same x and y coordinates) and computes the sum of the squares of the differences in values stored at the pixels (the difference in grey scale) and then returns the square root of this total.

A.1.2 Complete Hausdorff Distance

Figure A.2 shows the function for computing the complete Hausdorff distance between a pair of image vectors. Although the images are represented in grey scale, they are treated as binary black or white pixels in this case, with values over 5 being treated as shaded pixels, and those equal to or less than 5 being treated as blank. The algorithm then examines each shaded pixel in turn in image I, and finds the distance to the closest shaded pixel in image J. The maximum of all these distances is then taken, giving the Hausdorff distance from I to J. The same process is then repeated in the other direction, and the maximum of the two values is the value returned.

A.2 Tables of Results

A.2.1 k Nearest Neighbours Algorithm

Tables A.2, A.3 and A.4 show the results of tests run on the entire set of ten thousand images of handwritten digits, rounded to the nearest whole value. The true labels of the

Algorithm 10 *Hausdorff distance function.*

$Hausdorff($image $I,$ image $J)$

let $d_{max} = 0$

for each pixel p_i in I
  let $d_{min} = \infty$
  if p_i is shaded (*i.e.* $p_i > 5$)
    for each pixel p_j in J
      if (p_j is shaded AND $L_2(p_i, p_j) < d_{min}$)
        $d_{min} = L_2(p_i, p_j)$
    if $d_{min} > d_{max}$ then $d_{max} = d_{min}$

for each pixel p_j in J
  let $d_{min} = \infty$
  if p_j is shaded
    for each pixel p_i in I
      if (p_i is shaded AND $L_2(p_j, p_i) < d_{min}$)
        $d_{min} = L_2(p_j, p_i)$
    if $d_{min} > d_{max}$ then $d_{max} = d_{min}$

Return d_{max}

Figure A.2: Algorithm to compute the Hausdorff distance between 2 image vectors.

images are shown in each row, with the columns representing the label assigned to the image by the algorithm. The value $v_{i,j}$, where i is the row number and j is the column number, represents the number of images with label i, which were classified as having label j, out of every 1000 images with label i processed by the algorithm.

For example, out of every 1000 images of the digit 8 processed by the algorithm using 1 nearest neighbour (Table A.2), 18 were classified with label 5.

The final column, *Errors*, shows the number of misclassifications made by the algorithm in every 1000 tests, which is equivalent to $1000 - v_{ii}$ for any row i. At the foot of the table, the *Average misclassification error per 1000 images* is not equivalent to the average of the error values in the final column as the distribution of samples over the ten digits is not uniform (see Table A.1).

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	993	2	1	0	0	1	2	1	0	0	7
	1	0	995	3	0	1	1	1	0	0	0	5
	2	9	8	956	6	1	0	2	16	2	0	44
	3	0	2	4	955	1	21	0	9	4	4	45
	4	1	9	0	0	954	0	3	4	1	27	46
	5	2	1	0	19	2	951	10	1	6	8	49
	6	5	2	1	0	2	5	984	0	0	0	16
	7	0	19	4	2	4	0	0	962	0	9	38
	8	9	5	6	22	4	18	3	4	918	10	82
	9	1	5	1	7	13	5	1	9	1	957	43
Average misclassification error per 1000 images												39

Table A.2: 1 Nearest Neighbour algorithm – Classification results.

A.2.2 Unsupervised Learners Algorithms

Tables A.5, A.6 and A.7 show the results of using the kernel algorithm (with L_2 metric) to classify the set of ten thousand images of digits, rounded to the nearest whole value. The results in this section are in the same format as those in Section A.2.1.

Tables A.8, A.9 and A.10 show the average likelihoods, u_{ij}, of an image of digit i belonging to the class with label j, as estimated by the algorithm (to 3 decimal places). The true labels of the images are shown in the rows, and the classifications given to the image by the algorithm are displayed in the columns, with the values in the table representing the average probability of the images being assigned each of the ten possible labels by the algorithm. For example, over all of the images in the data set of ten thousand, the algorithm using Normal Distribution kernels with a standard deviation of 1000 (see Table A.8) gave an average likelihood of 0.013 that an image with label 9 should be assigned label 4, and a likelihood of 0.028 that an image with label 4 should be assigned label 9.

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	994	1	1	0	0	1	2	1	0	0	6
	1	0	999	1	0	0	0	0	0	0	0	1
	2	11	16	949	3	1	0	1	16	4	0	51
	3	1	3	4	964	1	14	0	9	2	2	36
	4	1	13	0	0	951	0	6	4	0	24	49
	5	6	2	0	11	3	963	6	1	1	7	37
	6	6	3	0	0	4	3	983	0	0	0	17
	7	0	24	4	0	2	0	0	961	0	9	39
	8	11	4	6	21	8	17	4	5	919	4	81
	9	3	7	2	9	10	3	1	18	2	945	55
Average misclassification error per 1000 images												37

Table A.3: 3 Nearest Neighbours algorithm – Classification results.

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	994	1	1	0	0	1	2	1	0	0	6
	1	0	998	2	0	0	0	0	0	0	0	2
	2	12	19	944	3	2	0	1	16	4	0	56
	3	0	3	2	963	1	15	1	10	2	3	37
	4	2	11	0	0	951	0	6	5	1	23	49
	5	7	1	0	12	2	966	4	1	1	4	34
	6	7	3	0	0	3	2	984	0	0	0	16
	7	0	29	3	0	4	0	0	954	0	10	46
	8	12	5	5	18	7	22	5	4	915	6	85
	9	5	7	4	9	9	4	1	16	1	944	56
Average misclassification error per 1000 images												38

Table A.4: 5 Nearest Neighbours algorithm – Classification results.

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	993	1	1	0	0	2	2	1	0	0	7
	1	0	995	3	0	1	1	1	0	0	0	5
	2	9	8	956	6	1	0	2	16	2	0	44
	3	0	2	4	955	1	21	0	9	4	4	45
	4	1	9	0	0	954	0	3	4	1	27	46
	5	2	1	0	19	2	951	10	1	6	8	49
	6	5	2	1	0	2	5	984	0	0	0	16
	7	0	19	4	2	4	0	0	962	0	9	38
	8	9	5	6	22	4	18	3	4	918	10	82
	9	1	5	1	7	13	5	1	9	1	957	43
Average misclassification error per 1000 images												37

Table A.5: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 1000) – Classification results.

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	993	1	1	0	0	2	2	1	0	0	7
	1	0	996	3	0	0	1	1	0	0	0	4
	2	9	9	954	6	1	0	2	16	3	0	46
	3	0	3	3	955	1	20	0	10	4	4	45
	4	1	12	0	0	953	0	3	4	1	25	47
	5	2	2	0	16	2	955	9	1	4	8	45
	6	5	3	0	0	3	5	983	0	0	0	17
	7	0	26	4	2	4	0	0	953	0	11	47
	8	9	5	5	23	4	20	3	4	918	9	82
	9	2	8	1	7	12	5	1	11	1	952	48
Average misclassification error per 1000 images												38

Table A.6: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 2000) – Classification results.

		Image Classifications per 1000										
		0	1	2	3	4	5	6	7	8	9	Errors
True Label	0	990	1	1	0	0	1	6	1	0	0	10
	1	0	998	2	0	0	0	0	0	0	0	2
	2	11	53	906	2	2	0	0	21	5	0	94
	3	0	23	1	949	1	10	0	10	3	4	51
	4	1	39	0	0	916	0	7	3	0	34	84
	5	2	27	0	11	1	933	10	3	2	10	67
	6	7	9	0	0	2	3	978	0	0	0	22
	7	0	56	2	0	2	0	0	920	0	19	80
	8	9	32	3	14	4	17	4	8	897	10	103
	9	5	19	1	7	7	3	1	17	0	941	59
Average misclassification error per 1000 images												57

Table A.7: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 4000) – Classification results.

	Average Likelihood of Classification									
	0	1	2	3	4	5	6	7	8	9
0	0.992	0.001	0.001	0	0	0.002	0.003	0.001	0	0
1	0	0.995	0.003	0	0.001	0.001	0.001	0	0	0
2	0.009	0.007	0.956	0.006	0.001	0	0.002	0.017	0.002	0
3	0	0.002	0.003	0.956	0.001	0.021	0	0.009	0.004	0.003
4	0.001	0.009	0	0	0.953	0	0.003	0.004	0.001	0.028
5	0.002	0.001	0	0.019	0.002	0.952	0.010	0.001	0.005	0.007
6	0.005	0.002	0.001	0	0.003	0.005	0.984	0	0	0
7	0	0.020	0.004	0.002	0.004	0	0	0.961	0	0.009
8	0.009	0.005	0.006	0.022	0.004	0.021	0.003	0.004	0.916	0.010
9	0.001	0.005	0.001	0.007	0.013	0.005	0.001	0.012	0.001	0.953
Average Negative Log-Likelihood										0.743

Table A.8: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 1000) – Likelihoods of labels.

	Average Likelihood of Classification									
	0	1	2	3	4	5	6	7	8	9
0	0.991	0.001	0.001	0	0	0.001	0.004	0.001	0	0
1	0	0.996	0.002	0	0	0.001	0.001	0	0	0
2	0.009	0.010	0.952	0.006	0.001	0	0.002	0.018	0.003	0
3	0	0.003	0.003	0.953	0.001	0.021	0	0.010	0.006	0.004
4	0.001	0.012	0	0	0.943	0	0.004	0.005	0.001	0.034
5	0.002	0.003	0	0.020	0.002	0.950	0.010	0.001	0.005	0.007
6	0.005	0.004	0	0	0.003	0.005	0.983	0	0	0
7	0	0.027	0.004	0.002	0.004	0	0	0.950	0	0.013
8	0.008	0.006	0.005	0.022	0.004	0.021	0.003	0.006	0.917	0.009
9	0.002	0.007	0.001	0.006	0.015	0.005	0.001	0.020	0.001	0.942
Average Negative Log-Likelihood										0.203

Table A.9: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 2000) – Likelihoods of labels.

	Average Likelihood of Classification									
	0	1	2	3	4	5	6	7	8	9
0	0.983	0.001	0.001	0.001	0	0.003	0.009	0.002	0.001	0
1	0	0.988	0.002	0.001	0.001	0	0.002	0.003	0.001	0.001
2	0.014	0.061	0.873	0.009	0.003	0.001	0.003	0.026	0.007	0.002
3	0.001	0.029	0.006	0.889	0.002	0.033	0.002	0.013	0.014	0.010
4	0.001	0.036	0.001	0.001	0.827	0.001	0.009	0.017	0.002	0.105
5	0.005	0.0027	0	0.039	0.005	0.877	0.016	0.006	0.010	0.015
6	0.010	0.014	0.001	0	0.005	0.008	0.960	0	0.001	0.001
7	0	0.061	0.004	0.002	0.011	0	0	0.868	0.001	0.052
8	0.009	0.041	0.009	0.039	0.009	0.035	0.008	0.013	0.813	0.024
9	0.004	0.020	0.002	0.008	0.062	0.006	0.002	0.072	0.004	0.820
Average Negative Log-Likelihood										0.188

Table A.10: Normal Distribution kernels (measured by L_2 distance, using standard deviation of 4000) – Likelihoods of labels.

Appendix B

Learning PDFA

Section B.1 demonstrates the necessity of a bound on the expected length of strings generated by the target automaton when learning a PDFA in terms of KL-divergence. Section B.2 gives details of the method of smoothing from an approximated distribution under L_1 distance to a close approximation under KL-divergence as discussed in Section 5.7.

B.1 Necessity of Upper Bound on Expected Length of a String When Learning Under KL-Divergence

We show that in order to learn a PDFA with respect to KL-Divergence, an upper bound on the expected length of string output by the target PDFA must be known.

Observation 41 *Consider the target automaton A, shown in Figure B.1.*

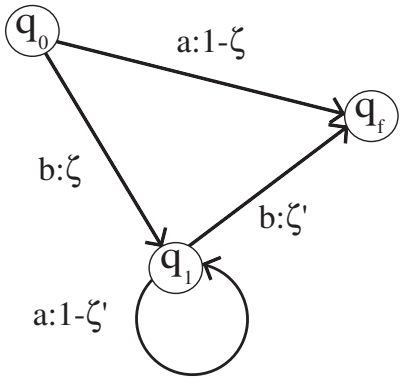


Figure B.1: Target PDFA A.

Suppose we wish to construct, with probability at least $1-\delta$, a distribution D_H such that $I(D_A||D_H) < \epsilon$, using a finite sample of strings generated by D_A. There is no algorithm that achieves this using a sample size that depends only on ϵ and δ.

Proof: A outputs the string a with probability $1-\zeta$, and outputs a string of the form $b(a)^*b$ with probability ζ.

Suppose an algorithm draws a sample S (from D_A) from which it is to construct D_H, with $|S| = f(\epsilon, \delta)$. Let $\zeta = \frac{1}{2|S|}$ be the probability that a random string is of the form $b(a)^n b$. Notice that S will be composed (entirely or almost entirely) of observations of string a. Therefore there is no way that the algorithm can accurately gauge the probability ζ' (see Figure B.1).

For $i \in \mathbf{N}$ let P_i be a probability distribution over the length ℓ of output strings, where $P_i(1) = (1-\zeta)$ and over all values of ℓ greater than 1 the distribution is a discrete exponential distribution defined as follows.

An infinite sequence $\{n_1, n_2, \ldots\}$ exists (see Observation 42), such that P_1 has a probability mass of $\frac{1}{4|S|}$ (half of the probability of generating a string of length greater than 1) over the interval $\{1, ..., n_1\}$, P_2 has probability mass of $\frac{1}{4|S|}$ over the interval $\{n_1+1, ..., n_2\}$, and in general P_i has probability of $\frac{1}{4|S|}$ over the interval $\{n_{i-1}+1, ..., n_i\}$. Let s_ℓ denote the string $ba^{(\ell-2)}b$ (with length ℓ). Given any distribution[1] D_H, for any $0 < \omega < 1$ there exists an interval $I_k = \{n_{k-1}+1, ..., n_k\}$ such that

$$\sum_{\ell \in I_k} D_H(s_\ell) \leq \omega.$$

To lower-bound the KL-divergence, we now redistribute the probability distribution of D_H in order to minimise the incurred KL-divergence (from the true distribution D_A), subject only to the condition that I_k still contains at most ω of the probability mass. In order to minimise the KL divergence, by a standard convexity argument, the algorithm must distribute the probability in direct proportion to D_A.

$$\begin{aligned} \forall \ell \in I_k \quad &: \quad \frac{D_H(\ell)}{D_A(\ell)} = 4|S|\omega, \text{ and} \\ \forall \ell \notin I_k \quad &: \quad \frac{D_H(\ell)}{D_A(\ell)} = \left(\frac{1-\omega}{1-\frac{1}{4|S|}}\right). \end{aligned}$$

[1] Note that this is a representation independent result - the distribution need not be generated by an automaton.

It follows that the KL-divergence can be written in terms of $|S|$ and ω in the following way:

$$\begin{aligned} I(D_A||D_H) &= \sum_{\ell \in \mathbb{N}} D_A(s_\ell).\log\left(\frac{D_A(s_\ell)}{D_H(s_\ell)}\right) \\ &= \sum_{\ell \in I_k} D_A(s_\ell).\log\left(\frac{1}{4|S|\omega}\right) + \sum_{\ell \notin I_k} D_A(s_\ell).\log\left(\frac{1-\frac{1}{4|S|}}{1-\omega}\right) \\ &= \left(\frac{1}{4|S|}\right)\log\left(\frac{1}{4|S|\omega}\right) + \left(1-\frac{1}{4|S|}\right)\log\left(\frac{1-\frac{1}{4|S|}}{1-\omega}\right) \\ &\geq \left(\frac{1}{4|S|}\right)(-2\log(|S|) - \log(\omega)) + \log\left(1-\frac{1}{4|S|}\right). \end{aligned}$$

Suppose that $\omega < 2^{-2\left(|S|\left(\epsilon - \log\left(1-\frac{1}{4|S|}\right)\right)+\log(|S|)\right)}$. It follows that:

$$\begin{aligned} &I(D_A||D_H) \\ &> \left(\frac{1}{4|S|}\right)\left(-2\log(|S|) + 2\left(2|S|\left(\epsilon - \log\left(1-\frac{1}{4|S|}\right)\right) + \log(|S|)\right)\right) + \\ &\quad \log\left(1-\frac{1}{4|S|}\right) \\ &= \left(\frac{-\log(|S|)}{2|S|}\right) + \left(\epsilon - \log\left(1-\frac{1}{4|S|}\right)\right) + \left(\frac{\log(|S|)}{2|S|}\right) + \log\left(1-\frac{1}{4|S|}\right) \\ &= \epsilon. \end{aligned}$$

It has been shown that for any specified ϵ, given any hypothesis distribution D_H, an exponential distribution D_A exists such that $I(D_A||D_H) > \epsilon$. □

Observation 42 *Given any positive integer $n_i \geq 1$ in the domain of string lengths, an exponential probability distribution exists such that at least $\frac{1}{4|S|}$ of the probability mass lies in the range $\{n_i + 1, ..., n_{i+1}\}$.*

Proof: If we look at strings of length greater than 1, then given some value n_i, there is some exponential distribution over these strings such that there exists an interval $\{n_i + 1, ..., n_{i+1}\}$ containing half of the probability mass of the distribution.

For an automaton A' (of a form similar to Figure B.1), the probability of an output string having a length greater than 1 is $\frac{1}{2|S|}$. Let $\ell_b = \ell - 1$ for those strings with $\ell > 1$ (where ℓ_b represents the number of characters following the initial b), and

let s_{ℓ_b} represent the string starting with b which has length ℓ. For any value of n_i, we can create a distribution:

$$D_{A'}(s_{\ell_b}) = \left(\frac{1}{2|S|}\right)\left(\frac{\ln\left(\frac{4}{3}\right)}{n_i+1}\right)\exp\left(-\left(\frac{\ln\left(\frac{4}{3}\right)}{n_i+1}\right)\ell\right).$$

A fraction $\frac{1}{8|S|}$ of the probability mass lies in the interval $\{2,...,n_i\}$. There exists a value $n_{i+1} = \left\lceil (n_i+1)\left(\ln(4)/\ln\left(\frac{4}{3}\right)\right)\right\rceil$ such that at least $\frac{1}{4|S|}$ of the probability mass lies in the interval $\{n_i+1,...,n_{i+1}\}$. □

B.2 Smoothing from L_1 Distance to KL-Divergence

We define DA_n to be the set of deterministic automata with a finite number n states. DA_n^c is the set of all members of DA_n which are complete graphs. Given alphabet Σ of symbols labelling transitions between states, it can be seen that $|DA_n^c| \leq n^{|\Sigma|+1}$.

Let $PDA_{n,\ell}$ be the set of all probabilistic deterministic automata with n states, where the probability associated with each transition has a binary representation with ℓ bits[2]. If each probability associated with a transition is represented by a bit string of length ℓ, and there are no more than $n|\Sigma|$ transitions in a deterministic automaton, then for any automata $\mathcal{A}_x \in DA_x^c$, there are no more than $\left(2^\ell\right)^{n|\Sigma|}$ automata with the same structure (states and transitions) in $PDA_{n,\ell}$. Therefore, $|PDA_{n,\ell}| \leq 2^{n|\Sigma|\ell}.n^{|\Sigma|+1}$.

We adapt the method of [13] to show that agnostic PAC-learnability of distributions generated from automata in $PDA_{n,\ell}$ in terms of L_1 distance implies learnability in terms of KL-divergence.

Lemma 43 *A probability distribution D over outputs of automata $\mathcal{A} \in PDA_{n,\ell}$ is learnable under KL-divergence in the agnostic PAC framework if it is agnostic PAC-learnable under L_1 distance.*

Proof: Let D be a distribution over outputs from automata $\mathcal{A} \in PDA_{n,\ell}$, and let A be an algorithm which takes an input set S (where $|S|$ is polynomial in $1/\epsilon$, $1/\delta$, n, $|\Sigma|$ and $1/\mu$) of samples generated i.i.d. from distribution D, and with probability at least $1-\delta$ returns a distribution $\hat{D}$ such that $L_1(D,\hat{D}) \leq \epsilon$.

Let $\xi = \epsilon^2(12n|\Sigma|\ell)^{-1}$. We define $\mathcal{A}'$ such that with probability at least $1-\delta$ $\mathcal{A}'$ returns distribution D', where $L_1(D,D') \leq \xi$. Algorithm A runs $\mathcal{A}$ with a sample S', where $|S'|$ is polynomial in $1/\xi$, $1/\delta$, n, $|\Sigma|$ and $1/\mu$ (and it should be noted that $\mathcal{A}'$ is still polynomial in $1/\epsilon$, $1/\delta$ and $1/n$).

[2]Note that an incomplete PDA $\mathcal{A}_i$ has an equivalent complete PDA $\mathcal{A}_c$, where each edge in $\mathcal{A}_c$ which is not in $\mathcal{A}_i$ has a probability of zero associated with it.

We define $D_{\mathcal{A}}$ to be the distribution over outputs of automaton $\mathcal{A}$ and $D_{n,\ell}$ to be the unweighted mixture of the distributions over outputs of automata in $PDA_{n,\ell}$, $D_{n,\ell}(s) = |PDA_{n,\ell}|^{-1} \sum_{\mathcal{A} \in PDA_{n,\ell}} D_{\mathcal{A}}(s)$.

Now let $D''(s) = (1-\xi)D'(s) + \xi D_{n,\ell}(s)$. It follows that $L_1(D', D'') \leq 2\xi$. With probability at least $1-\delta$, $L_1(D, D') \leq \xi$, and therefore with probability at least $1-\delta$, $L_1(D, D'') \leq \xi$.

Let $S_< = \{s \in \Sigma^* | D''(s) < D(s)\}$. Members of $S_<$ contribute positively to $I(D||D'')$. Therefore

$$\begin{aligned} I(D||D'') &\leq \sum_{s \in S_<} D(s) \log\left(\frac{D(s)}{D''(s)}\right) \\ &= \sum_{s \in S_<} (D(s) - D''(s)) \log\left(\frac{D(s)}{D''(s)}\right) + \sum_{s \in S_<} D''(s) \log\left(\frac{D(s)}{D''(s)}\right). \quad \text{(B.1)} \end{aligned}$$

We have shown that $L_1(D, D'') \leq 3\xi$, so $\sum_{s \in S_<} (D(s) - D''(s)) \leq 3\xi$. Analysing the first term in Equation B.1, it can be seen that

$$\sum_{s \in S_<} (D(s) - D''(s)) \log\left(\frac{D(s)}{D''(s)}\right) \leq 3\xi \max_{s \in S_<} \left\{ \log\left(\frac{D(s)}{D''(s)}\right) \right\}.$$

Note that for all $s \in \Sigma^*$, $D''(s) \geq \xi |PDA_{n,\ell}|^{-1}$. It follows that

$$\begin{aligned} \max_{s \in S_<} \left\{ \log\left(\frac{D(s)}{D''(s)}\right) \right\} &\leq \log\left(\left(\frac{1}{\xi}\right) 2^{n|\Sigma|\ell} . n^{|\Sigma|+1}\right) \\ &= \log(n^{|\Sigma|+1}) + n|\Sigma|\ell - \log(\xi). \end{aligned}$$

Examining the second term in Equation B.1,

$$\sum_{s \in S_<} D''(s) \log\left(\frac{D(s)}{D''(s)}\right) = \sum_{s \in S_<} D''(s) \log\left(\frac{D''(s) + h_s}{D''(s)}\right),$$

where $h_s = D(s) - D''(s)$, which is a positive quantity for all $s \in S_<$. Due to the concavity of the logarithm function, it follows that

$$\begin{aligned} \sum_{s \in S_<} D''(s) \log\left(\frac{D''(s) + h_s}{D''(s)}\right) &\leq \sum_{s \in S_<} D''(s) h_s \left[\frac{d}{dx}(\log(x))\right]_{x = D''(s)} \\ &= \sum_{s \in S_<} h_s \\ &\leq 3\xi. \end{aligned}$$

Therefore, $I(D||D'') \leq 3\xi(1 + \log(n^{|\Sigma|+1}) + n|\Sigma|\ell - \log(\xi))$. For values of $\xi \leq \epsilon^2 (12n|\Sigma|\ell)^{-1}$, and values of $\epsilon \leq \left(\log\left(n^{|\Sigma|+1}\right)\right)^{-1}$, it can be seen that $I(D||D'') \leq \epsilon$. □

Bibliography

[1] N. Abe, J. Takeuchi and M. Warmuth. Polynomial Learnability of Stochastic Rules with respect to the KL-divergence and Quadratic Distance. *IEICE Trans. Inf. and Syst.*, Vol E84-D(3) pp. 299-315 (2001).

[2] N. Abe and M.K. Warmuth. On the Computational Complexity of Approximating Distributions by Probabilistic Automata. *Machine Learning*, 9, pp. 205-260 (1992).

[3] E.L. Allwein, R.E. Schapire and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1, pp. 113-141 (2000).

[4] M. Anthony and P.L. Bartlett. Neural Network Learning: Theoretical Foundations. *Cambridge University Press* (1999).

[5] P. Auer, R.C. Holte and W. Maass. Theory and Applications of Agnostic PAC-Learning with Small Decision Trees. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 21-29 (1995).

[6] G. Bejerano and G. Yona. Variations on Probabilistic Suffix Trees: Statistical Modeling and Prediction of Protein Families. *Bioinformatics*, Vol. 17, No. 1, pp. 23-43 (2001).

[7] C.M. Bishop. Neural Networks for Pattern Recognition. *Oxford University Press* (1995).

[8] R.C. Carrasco and J. Oncina. Learning Stochastic Regular Grammars by means of a State Merging Method. In *The 2nd Intl. Collo. on Grammatical Inference and Applications*, pp. 139-152 (1994).

[9] H. Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *Annals of Mathematical Statistics*, 23 pp. 493-509 (1952).

[10] A. Clark and F. Thollard. PAC-learnability of Probabilistic Deterministic Finite State Automata. *Journal of Machine Learning Research*, 5 pp. 473-497 (2004).

[11] C. Cortes and Y. LeCun. The MNIST Database of Handwritten Digits. *yann.lecun.com*, viewed 13 February 2007, <http://yann.lecun.com/exdb/mnist>.

[12] T.M. Cover and J.A. Thomas. Elements of Information Theory. *Wiley Series in Telecommunications*, John Wiley & Sons (1991).

[13] M. Cryan and L. A. Goldberg and P. W. Goldberg. Evolutionary Trees can be Learned in Polynomial Time in the Two-State General Markov Model. *SIAM Journal on Computing*, 31(2) pp. 375-397 (2001).

[14] S. Dasgupta. Learning Mixtures of Gaussians. *40th IEEE Symposium on Foundations of Computer Science* (1999).

[15] L. Devroye, L. Györfi and G. Lugosi. A Probabilistic Theory of Pattern Recognition. *Springer* (1996).

[16] G. Druck, C. Pal, A. McCallum and X. Zhu. Semi-supervised classification with hybrid generative/discriminative methods. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 280-289 (1996).

[17] R.O. Duda and P.E. Hart. Pattern Classification and Scene Analysis. *John Wiley and Sons* (1973).

[18] P. Dupont, F. Denis and Y. Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38, pp. 1349-1371 (2005).

[19] P. Dupont and J-C. Amengual. Smoothing Probabilistic Automata: An Error-Correcting Approach. *Lecture Notes in Computer Science*, 1891/2000, pp. 51-64 (2004).

[20] J. Feldman and R. O'Donnell and R. Servedio. Learning Mixtures of Product Distributions over Discrete Domains. *46th Symposium on Foundations of Computer Science (FOCS)*, pp. 501-510 (2005).

[21] I.K. Fodor. A Survey of Dimension Reduction Techniques. *Lawrence Livermore National Laboratory*, viewed 13 February 2007, <http://www.llnl.gov/tid/lof/documents/pdf/240921.pdf> (2002).

[22] P.W. Goldberg. When Can Two Unsupervised Learners Achieve PAC Separation? In *Proceedings of the 14th Annual Conference on Computational Learning Theory, COLT 2001* and *5th European Conference on Computational Learning Theory, EuroCOLT 2001*, Vol. 2111, pp. 303-319 (2001).

[23] P.W. Goldberg. Some Discriminant-based PAC Algorithms. *Journal of Machine Learning Research*, Vol.7, pp. 283-306 (2006).

[24] P. Guttman, S.V.N. Vishwanathan and R.C. Williamson. Learnability of Probabilistic Automata via Oracles. In *Proceedings of ALT 05*, LNAI 3734, pp. 171-182 (2005).

[25] P. Guttman, S.V.N. Vishwanathan and R.C. Williamson. On distribution classes induced by probabilistic automata. *Unpublished*, viewed 05 September 2007, <http://users.rsise.anu.edu.au/ oguttman/home/Myhill_Nerode_PFA.pdf>.

[26] D. Haussler. Probably Approximately Correct Learning. In*Proceedings of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pp. 1101-1108 (1990).

[27] D. Haussler, M. Kearns, N. Littlestone and M.K. Warmuth. Equivalence of Models for Polynomial Learnability. *Information and Computation*, 95[2], pp. 129-161 (1991).

[28] C. de la Higuera and J. Oncina. Learning Stochastic Finite Automata. In *Proceedings of the 7th International Colloquium on Grammatical Inference (ICGI)*, LNAI 3264, pp. 175-186 (2004).

[29] K. Hoffgen. Learning and Robust Learning of Product Distributions. In *ACM COLT*, pp. 77-83 (1993).

[30] M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R.E. Schapire and L. Sellie. On the Learnability of Discrete Distributions. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pp. 273-282 (1994).

[31] M.J. Kearns and R.E Schapire. Efficient Distribution-free Learning of Probabilistic Concepts. *Journal of Computer and System Sciences*, 48(3), pp. 464-497 (1994).

[32] M.J. Kearns, R.E. Schapire and L.M. Sellie. Toward Efficient Agnostic Learning. In *Proceedings of Computational Learing Theory*, 17(2/3), pp. 341-352 (1992).

[33] J. Lasserre, C.M. Bishop and T. Minka. Principled hybrids of generative and discriminative models. In *Proceedings of the 2006 IEEE Conference on Computer Vision and Pattern Recognition*, 1, pp. 87-94 (2006).

[34] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), pp. 2278-2324 (1998).

[35] P.M. Long, R.A. Servedio and H.U. Simon. Discriminative Learning can Succeed where Generative Learning Fails. *Information Processing Letters* (2007), 103(4), pp. 131-135 (2007).

[36] E. Mossel and S. Roch. Learning Nonsingular Phylogenies and Hidden Markov Models. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, Baltimore (STOC05), MD, USA, pp. 366-376 (2005).

[37] A.Y. Ng and M.I. Jordan. On discriminative vs generative classifiers: A comparison of logistic regression and naive bayes. In *Neural Information Processing Systems*, Vancouver, Canada, pp. 841-848 (2001).

[38] N. Palmer and P.W. Goldberg. PAC Classification via PAC Estimates of Label Class Distributions. *Tech rept. 411, Dept. of Computer Science, University of Warwick* (2004). [Available from arXiv as cs.LG/0607047.]

[39] N. Palmer and P.W. Goldberg. PAC-Learnability of Probabilistic Deterministic Finite State Automata in terms of Variation Distance. In *Proceedings of ALT 05*, LNAI 3734, pp. 157-170 (2005).

[40] N. Palmer and P.W. Goldberg. PAC-Learnability of Probabilistic Deterministic Finite State Automata in terms of Variation Distance. *Theoretical Computer Science*, 387(1), pp. 18-31 (2007).

[41] D. Ron, Y. Singer and N. Tishby. On the Learnability and Usage of Acyclic Probabilistic Finite Automata. *Journal of Computer and System Sciences*, 56(2), pp. 133-152 (1998).

[42] Y.D. Rubinstein and T. Hastie. Discriminative vs Informative Learning. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pp. 49-53 (1997).

[43] S.J. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. *Prentice Hall*, 1st Edition (1995).

[44] P.Y. Simard, D. Steinkraus and J. Platt Best Practice for Convolutional Neural Networks Applied to Visual Document Analysis. *International Conference on Document Analysis and Recogntion (ICDAR)*, pp. 958-962 (2003).

[45] L.G. Valiant. A Theory of the Learnable. *Journal of the Association for Computing Machinery*, 27 pp. 1134-1142 (1984).

[46] V.N. Vapnik and A. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory of Probability and its Applications*, 16(2) pp. 264-280 (1971).

[47] V.N. Vapnik. The Nature of Statistical Learning Theory. *New York: Springer*, 2nd Edition (2000).

[48] X. Zhu. Semi-Supervised Learning Literature Survey. Tech Report no.1530 in Computer Sciences, University of Wisconsin-Madison (2005).