Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2006-057 September 1, 2006

Learning with Online Constraints: Shifting Concepts and Active Learning

Claire E. Monteleoni

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Learning with Online Constraints: Shifting Concepts and Active Learning

by

Claire E. Monteleoni

A.B. Harvard University, 1998
S.M. Massachusetts Institute of Technology, 2003

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2006

Author . . . .
Department of Electrical Engineering and Computer Science
August 25, 2006

Certified by . . . .
Tommi S. Jaakkola
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning with Online Constraints:
Shifting Concepts and Active Learning

by

Claire E. Monteleoni

Submitted to the Department of Electrical Engineering and Computer Science
on August 25, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

Many practical problems such as forecasting, real-time decision making, streaming data applications, and resource-constrained learning, can be modeled as *learning with online constraints*. This thesis is concerned with analyzing and designing algorithms for learning under the following online constraints:

i) The algorithm has only sequential, or one-at-time, access to data.

ii) The time and space complexity of the algorithm must not scale with the number of observations.

We analyze learning with online constraints in a variety of settings, including *active learning*. The active learning model is applicable to any domain in which unlabeled data is easy to come by and there exists a (potentially difficult or expensive) mechanism by which to attain labels.

First, we analyze a supervised learning framework in which no statistical assumptions are made about the sequence of observations, and algorithms are evaluated based on their regret, i.e. their relative prediction loss with respect to the hindsight-optimal algorithm in a comparator class. We derive a lower bound on regret for a class of online learning algorithms designed to track *shifting concepts* in this framework. We apply an algorithm we provided in previous work, that avoids this lower bound, to an energy-management problem in wireless networks, and demonstrate this application in a network simulation. Second, we analyze a supervised learning framework in which the observations are assumed to be iid, and algorithms are compared by the number of prediction mistakes made in reaching a target generalization error. We provide a lower bound on mistakes for Perceptron, a standard online learning algorithm, for this framework. We introduce a modification to Perceptron and show that it avoids this lower bound, and in fact attains the optimal mistake-complexity for this setting.

Third, we motivate and analyze an online active learning framework. The observations are assumed to be iid, and algorithms are judged by the number of label queries to reach a target generalization error. Our lower bound applies to the active learning setting as well, as a lower bound on labels for Perceptron paired with any active learning rule. We provide a new online active learning algorithm that avoids the lower bound, and we upper bound its label-complexity. The upper bound is optimal and also bounds the algorithm's total errors (labeled and unlabeled). We analyze the algorithm further, yielding a label-complexity bound under relaxed assumptions.

Using optical character recognition data, we empirically compare the new algorithm to an online active learning algorithm with data-dependent performance guarantees, as well as to the combined variants of these two algorithms.

Thesis Supervisor: Tommi S. Jaakkola
Title: Associate Professor of Electrical Engineering and Computer Science

To Grandpa Bernie and Nonno Leo.

Acknowledgments

I am extraordinarily grateful to Tommi Jaakkola, my thesis supervisor, and my research advisor throughout my MIT career. Thank you for generously sharing your expertise and your lucid view of all the areas of mathematics and of machine learning that we have discussed over the years. I feel extremely lucky to have been your student.

My thesis committee also deserves many thanks. I am very grateful that Piotr Indyk, a pivotal contributor to the literature on streaming algorithms and computational geometry, agreed to join my thesis committee and share his expertise. Thank you for stimulating research discussions, and course lectures. Sanjoy Dasgupta of UC San Diego has been an invaluable influence on my recent and ongoing research on active learning, and I would like to sincerely thank him for agreeing to serve on my thesis committee, despite the physical distance.

Much of this thesis is the product of my collaborations with various talented people. Again I thank Tommi Jaakkola, this time for productive and stimulating collaborations. It was a wonderful experience, after taking Hari Balakrishnan's course on computer networks, to have the opportunity to collaborate with him, as well as the talented Nick Feamster. I am extraordinarily grateful to Adam Tauman Kalai for hosting me on a summer internship and sharing his rich expertise on randomized and online algorithms, leading to an excellent and fun collaboration. Recently I have greatly enjoyed and benefited from an ongoing collaboration with Matti Kääriäinen: many thanks.

I feel particularly fortunate to have entered into a fascinating research conversation with Sanjoy Dasgupta at the 2004 Conference on Learning Theory. This conversation has led to a great collaboration and a stimulating visit to UCSD, and has in fact never ceased. Thank you so much, and see you in La Jolla!

There are many other people who have affected my research career in graduate school. These include helpful and influential professors, students who helped in various ways with my research, and in answering technical questions, and colleagues who have given me career advice along the way. I would like to thank MIT professors Michael Collins, David Karger and Santosh Vempala. I am extremely grateful for the help of the following current and former MIT graduate students and postdocs: John Barnett, Adrian Corduneanu, Nick Harvey, Jessica Howe, Adam Klivans, Ronny Krashinsky, April Rasala Lehman, Alantha Newman, Luis Ortiz, Alex Park, Luis Perez-Breva, Alexander Rakhlin, Jason Rennie, Nati Srebro and Hiuzhen (Janey) Yu. Outside of MIT, I would like to thank Ran Gilad-Bachrach and David McCallester.

I am also very grateful to people who have influenced my career prior to graduate school, people who have indirectly influenced my research and career choices during graduate school, and people who have influenced me during graduate school in spheres outside of my career. The first group includes several earlier mentors whose advice and encouragement still echoes in my head and whose careers have shaped mine by example. Thank you for believing in me; you know who you are. The second group includes graduate student colleagues with whom I have interacted, such as problem set collaborators and course TAs, as well as professors whom I have learned from in courses and meetings, colleagues during my summer internship and those with whom I have interacted at conferences. In particular, at MIT I would like to thank the Theory and Machine Learning groups at CSAIL, and the CSAIL staff, as well as several current and former members of the Center for Biological and Computational Learning, and the Department of Mathematics. I would also like to thank the Toyota Technological Institute at Chicago, and the summer research interns of 2004. I thank the Sherman Cafe, in Somerville, for being my home away from lab where much of my work was carried out, and the CSAIL Reading Room for providing a quiet place to think, as well as endless chocolate.

I don't know where to start in thanking the third group. Let me just say that I send my heartfelt gratitude to all my dear friends, as well as a sincere apology that I am not listing you all by name. I will just mention a few people who have been extremely supportive during graduate school: Alantha, Alex, Anitha, Anna, Chudi, Dan, Daria, Dean, Elodie, Flora, Funke, Ginger, Karen, Lamine, Laurie, Mary, Naoka, Nick, Patty, Paula, Qasba, Robin, Sarah, Sasha, Steve, Steven and Tammy. I would also like to thank countless other friends at MIT, in the Boston area, New York, California, Italia, and throughout the world, who have been a steady source of inspiration, entertainment and support for me. Thank you. Throughout my graduate school career, Rambax MIT has served as my weekly source of laughter, polyrhythms, and friendship, and provided me the opportunity of an unforgettable trip to Senegal (*neex na torop!*). To all of you, I send a deeply grateful *jerejef waay!* Taking after a famous computer science thesis [Kal01], in order to avoid any omissions, I would also like to thank anyone who is taking the time to read these acknowledgements.

Finally, I profusely thank my family. I send gratitude to my wonderful and supportive aunts, uncles and cousins (*forza cugini!*) spread across at least three continents. I would like to thank Grandma and Grandpa, who deeply value learning and language. I thank Nonno, in memoriam, for his strong belief in education, and Nonna who approved of my work in her own particular way: remarking with delight, upon being presented with a copy of my Masters thesis, on the various nested subheadings such as Section 1.2.3, etc.

My immediate family has been infinitely supportive: a constant source of love, entertainment, and cheerleading! Thank you for sitting through an hour-long answer, involving diagrams, every time you asked me about my current research. Thanks for having the patience and interest to actually reach the point of understanding in these discussions. It is with utmost warmth and gratitude that I thank my dear brother, Paul. From the beginning, my parents have instilled in me a fascination with and a propensity for both languages and visual art, surprisingly coming in handy for my research. Thank you, thank you, thank you, dearest Dad and Mom.

And now I guess I can finally say it: *Crepi il lupo.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning, a dynamic subfield of artificial intelligence (AI), produces tools and techniques currently in effective use and high demand in a broad range of research and applications. In the past few decades, machine learning and statistical pattern recognition algorithms have impacted not only computer science, but also a range of fields from economics to the health sciences, and have potential in many more. Within computer science, machine learning algorithms have been applied to caching, bug detection, and recent problems in computer networks. Designers of Web search-engines currently rely on machine learning expertise for intelligent search techniques, mining data from Web pages and automatically indexing the Web. Machine learning has been used in economic and financial applications such as portfolio management strategies that adapt to the current market climate. Moreover, machine learning and statistical pattern recognition tools have been revolutionizing discovery in the natural sciences, as evidenced by the recent emergence of the fields of bioinformatics, and its efficacy for analyzing the genome, and computational chemistry, and its successes in the discovery of new medications.

It is currently the state of the art, however, that machine learning practitioners face a myriad of choices when applying a technique or tool. These choices begin with very basic ones, such as which algorithm to use, or which model class to consider for hypotheses. Even beyond basic decisions however, there remains a series of complex choices, such as how to set the parameters of an algorithm, and how many data points ought to be used for training and testing. Often, there are no formal methods available by which to make these decisions. Yet the decisions required in applying a machine learning technique do in fact depend on some fundamental quantities and complexity tradeoffs inherent to the problem of learning. For example, there is a basic tradeoff between the number of training examples used and the level of error obtained by the learned classifier. Similarly, there is a tradeoff between the complexity (in parameter space) of the model, and its ability to generalize to examples that were not present in the training set. Formalizing the tradeoffs that define the complexity of machine learning, and designing algorithms that exploit them, are the goals of a research field at the intersection of theoretical computer science and machine learning, best described by the terms *theoretical machine learning* and *computational learning theory.*

The individual contribution of this dissertation to the fields of theoretical machine learning, and computational learning theory, is focused on several constraints to the learning problem that we have chosen because they are well motivated by fundamental questions of AI, they are relevant to practical applications, and they address open problems in the literature. All the work that follows addresses learning with two online constraints: data is received in a sequential fashion, and the learner is constrained against computation and memory that scales with the amount of seen data. The subproblems addressed include learning when there are no statistical assumptions on the observations, and learning in the active setting in which the data is unlabeled and the learner can choose to pay for labels. In the rest of this introduction we will introduce the various frameworks studied, and then outline our contributions.

1.1 Learning with online constraints

This thesis is concerned with applying online constraints to the problem of machine learning. In particular, there are two types of online constraints defining all the results we report. The first concerns the algorithm's access to data observations. We are concerned with models in which the observations are received in a sequential fashion, i.e. one at a time. Once an observation has been seen, it might not ever be observed again. The second constraint concerns allowable algorithmic solutions to the learning problem we define. The learner has constraints on computation and memory entailing that it cannot solve the problem via batch learning: it is constrained against increasing its memory usage and computation time with the number of seen examples.

1.1.1 Online access to observations

The first constraint, that the observations be received sequentially, defines the sequential or online learning framework: training examples are received one at a time, and the learner must make a prediction at each time-step. This requirement is well motivated from an AI perspective, in emulating human cognition. Humans are not usually granted the luxury of learning a concept from a batch of labeled examples. Instead we receive observations in a sequential fashion, and must update our beliefs online. The sequential framework effectively models many practical problems. Two categories of applications that are well served by the sequential model are problems of temporal forecasting, such as predicting the stock market, weather, or usage patterns and burstiness of the internet, and streaming data applications. In forecasting problems, the online model is useful because not only are observations received in a sequence, but also it is often the case that predictions are needed almost immediately after each data observation, and the data may vary with time. In streaming applications, the data is received in a sequential fashion and is often of extremely high-dimension, in which case online access to the data may be the only practical model.

1.1.2 Online constraint on time and space complexity

The second constraint, which limits time and memory usage from scaling with the number of observations, is motivated in part by the real-world limits on time and memory faced by any computational learner, and in part by an effort to best match the solution to the problem: to reflect the sequential nature of the problem in the design of algorithms that solve it. In other words, the sequential framework raises interesting computational issues: tradeoffs in complexity and resources for computational learners. We require that the learner cannot store all previously seen examples and then apply a batch learning algorithm to them, but must instead intelligently summarize its observations. Without this constraint, the problem would be reduced to that of batch learning, which already has an abundant literature. Practical motivations include computation on small devices for which the memory limit is easily reached, as well as learning under non-stationarity where memory of the distant past is less useful for current predictions. Additionally, the time complexity of the belief update step should be constrained against scaling with the number of past examples, in order for the algorithm to be effective in the online setting. This constraint has practical motivation in any system that must predict in real-time.

1.2 Supervised learning framework

Online learning can be studied in the *supervised learning* framework, meaning all the examples are labeled. The previous section outlined some of the practical motivations for supervised online learning, which is typically just called online learning. We will study two such frameworks which model two different scenarios with two different measures for judging an algorithm's performance, involving different assumptions and analysis techniques.

1.2.1 Non-stochastic setting and regret bounds

First we study a universal prediction setting in which no statistical assumptions are made on the observation sequence. By *non-stochastic*, we denote the lack of statistical assumptions. The observations could even be generated online by an adaptive adversary. Since no sampling assumptions can be made about the sequence to be predicted, algorithms can only be judged by relative performance measures. The analysis of algorithms is therefore focused on establishing bounds on the *regret*, or the difference between the cumulative loss of the algorithm and the loss of the best method in an appropriately defined comparator class, with respect to hindsight knowledge of the observed sequence. In this framework we study *shifting algorithms*: a general class of algorithms that model the observations as being non-stationary, but generated from a shifting sequence of stationary distributions.

1.2.2 Iid assumption and mistake bounds

Motivated by a desire to bound a quantity that is intuitively more absolute and definitive than the notion of regret, we then study a supervised online learning analysis setting that permits us to bound the final error of the hypothesis attained. In order to do so, we add a statistical assumption on the generation of the sequence of observations. We assume the sequence of observations is *iid*, the abbreviation for "independently, identically distributed," meaning that it results from independent random draws from a fixed probability distribution over the input space. Algorithms for the sequential iid framework can be compared by their *mistake-complexity*: the number of prediction mistakes they make before converging on an accurate model. This convergence can be analyzed with respect to the error rate of the hypothesis on the full input distribution. If the concept class over which learning is performed contains a perfect classifier for the problem, then this error rate is actually the generalization error of the hypothesis.

1.3 Active learning framework

Online learning can also be studied in an *active learning* framework. In many machine learning applications, such as speech recognition, medical diagnosis and Web page classification, access to labeled data is much more limited or expensive than access to unlabeled samples from the same data-generating distribution. It is often realistic to model this scenario as active learning. Since active learning allows for intelligent choices of which examples to label, often the *label-complexity*, the number of labeled examples required to learn a concept via active learning, is significantly lower than the PAC sample complexity. PAC refers to the "Probably Approximately Correct" learning theoretic analysis framework, originally proposed by [Val84], and well explained in [KV94]. The PAC sample complexity of a concept is an upper bound on the number of labeled examples, sampled iid from a fixed input distribution, such that with high probability with respect to the sampling, the function generating the labels of the examples can be approximated to within a fixed error rate on the input distribution. Here we describe the specific online active learning framework studied.

1.3.1 Active learning in the PAC-like selective sampling model

The active learning model is applicable in any domain in which unlabeled data is easy to come by and there exists a (potentially difficult or expensive) mechanism by which to obtain their labels. While the query learning model has been well studied theoretically (see e.g. [Ang01]), it is often unrealistic in practice, as it requires access to labels for the entire input space. It has been shown in domains such as text and OCR that the synthetic examples on which the learner has the most uncertainty may be difficult even for a human to label [LG94]. In *selective sampling* (originally introduced by [CAL94]) the learner receives unlabeled data, sampled iid from a fixed input distribution, and may request certain labels to be revealed, at a constant cost per label.

1.3.2 Online active learning

Selective sampling can be modeled in an online or sequential setting, in which unlabeled examples are received one at a time and the learner must make a one-time choice whether to pay for the current label. We will use the terms "sequential selective sampling" and "online active learning" interchangeably. We motivate a framework involving both online constraints: an algorithm must perform sequential selective sampling, thus respecting the first constraint, and obey the second constraint in that neither the time nor space complexity scales with the number of seen labeled examples, or mistakes. Algorithms for sequential selective sampling that also respect the online constraints on time and memory we consider to be strongly online active learners, though with a slight overload of terminology we will also refer to them simply as online active learners.

In an iid framework that is both active and sequential, interesting issues arise. Beyond just minimizing the number of mistakes needed to learn a concept to a fixed error rate on the full input distribution, in active learning the goal is to minimize the number of labels that the algorithm needs to check, in doing so, i.e. the label-complexity in this setting. A distinction now exists between mistakes and errors: mistakes are a subset of total errors on which the algorithm requests labels, and thus receives feedback on its erroneous predictions. Thus *error-complexity* can be analyzed as a separate quantity from mistake-complexity for active sequential algorithms.

1.3.3 Practical motivations

Sequential active learning with online constraints has well motivated real-world applications such as OCR on small devices. As of 2004, a quarter of US physicians were using handheld computers.[1] In the 2004 US presidential election, several major political organizations equipped canvassers going door-to-door with handheld computers to collect neighborhood voting data. Limited computing power may constrain the OCR training of these handhelds to be online. In the selective sampling setting, the device may occasionally ask the user to provide a label for a written character, for example by entering it through the keypad. Human usage would favor algorithms that minimize the number of such correction events during the learning process.

Document filtering is a problem that has been modeled using active learning: the filtering mechanism implements the choice of whether to query a label, which amounts to forwarding the document to the human user and thus receiving feedback from the user as to the document's relevance. Email filtering is an increasingly important problem, as electronic information flow, both relevant and irrelevant (such as spam) continues to increase. With many users receiving email on handheld devices that may have memory and computation constraints, online email filtering is poised to become an increasingly necessary application of online active learning.

[1] McAlearney A. S., Schweikhart S. B., Medow M. A., Doctors' experience with handheld computers in clinical practice: qualitative study. *British Medical Journal.* 328(7449):1162. 2004.

1.4 Outline of contributions

The organization of our contributions is as follows. The first part of this thesis concerns supervised online learning. In Chapter 2, we consider a scenario in which there are no assumptions on the observation sequence, and algorithms are judged in terms of their regret: their relative loss with respect to the hindsight optimal algorithm in a comparator class. In this chapter we provide a lower bound on regret for a broad class of online learning algorithms for this setting, and apply an algorithm we introduced in previous work, that avoids this lower bound, to a problem in wireless networks, in simulation. We continue considering supervised online learning in Chapter 3, through Section 3.5, focusing instead on a scenario in which the observations are assumed to be iid and algorithms are judged by the number of mistakes to reached a fixed error rate on the input distribution. In these sections we provide a lower bound on mistakes for standard Perceptron, and introduce a Perceptron variant for which we provide a new upper bound on mistakes.

In the remainder of the thesis we consider an active setting, retaining the iid assumption, in which algorithms are judged by the number of labels to reached a fixed error rate on the input distribution. The lower bound of Section 3.4 holds for labels as well, and in the remainder of Chapter 3 we give an online active learning algorithm with upper bounds on label queries and total errors. In Chapter 4 we analyze the algorithm from Chapter 3 in various additional ways, and then apply it to optical character recognition, along with an online active learning algorithm from the literature and several variants combining the two algorithms, as well as random sampling.

Chapter 2

Learning shifting concepts

This chapter is based on joint work with Tommi Jaakkola. Sections 2.2–2.4 are based on work that originally appeared in [MJ03]. Section 2.5 is based on work that is also joint with Hari Balakrishnan and Nick Feamster [MBFJ04].

In this chapter we study a supervised online learning framework involving no statistical assumptions. This framework can be used to model, regression, estimation, or classification. As in the typical online learning setting, the learner receives examples, (x_t, y_t), one at a time. We study a setting in which the learner has access to a set of "experts,"[1] and their predictions on each observation, but possesses no other *a priori* information relating to the observation sequence. In this chapter we are concerned with cumulative prediction loss, i.e. loss on every example counts, as opposed to Chapters 3 and 4, in which we are only concerned with the final error attained. The objective in this setting is to design algorithms whose prediction loss can be upper bounded with respect to the best (in an appropriately chosen comparison class) algorithm that has hindsight access to the observation sequence, over a finite, known, time horizon T. All algorithms in this chapter respect the two online constraints as defined in the introduction.

The motivation of our previous work in [Mon03], which we summarize in Section 2.2, was to improve online learning in the non-stochastic case, by removing prior assumptions. Previous algorithms for this setting, designed to track shifting concepts, are parameterized by the switching-rate, or rate of concept shift, requiring a prior assumption as to the level of non-stationarity of the observation sequence. We designed an algorithm to learn this parameter online, simultaneous to the original learning task, and showed that its regret is upper bounded by $O(\log T)$. Our analysis also yielded a regret upper bound for an existing class of algorithms, including the shifting algorithms, discussed above.

In Section 2.3 we derive the optimal learning discretization for our algorithm. In Section 2.4 we provide a lower bound on regret for the class of shifting algorithms discussed above, which can be $\Omega(T)$, depending on the observation sequence. The lower bound illustrates the asymptotic advances made by our algorithm.

[1] The term "expert" is arbitrary: the "experts" need not have any true expertise.

In Section 2.5 we apply our algorithm to energy management for mobile wireless devices of the 802.11 standard, in a network simulation. In our `ns-2` simulations, our application saved 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average latency by a factor of 1.02, and not more than 1.2.

2.1 Related work

This chapter relates to the literature on shifting algorithms. The ability to shift emphasis from one "expert" to another, in response to changes in the observations, is valuable in many applications. When given access to a set of experts whose prediction mechanisms are unknown to the learner, the learner may choose to quickly identify a single best expert to rely on, thus modeling a static concept, as in an algorithm due to Littlestone and Warmuth [LW89], or switch from one expert to another in response to perceived changes in the observation sequence, thus modeling shifting concepts, as in an algorithm due to Herbster and Warmuth [HW98]. Both of these algorithms make modeling assumptions about the switching dynamics of the observation sequence.

Many algorithms developed for universal prediction on the basis of a set of experts have clear performance guarantees (e.g., [LW89; HKW98; HW98; Vov99]). The performance bounds characterize the regret relative to the best expert, or best sequence of experts, chosen in hindsight. Algorithms with such relative loss guarantees have also been developed for adaptive game playing [FS99], online portfolio management [HSSW96], paging [BBK99] and the k-armed bandit problem [ACBFS02]. The form of these algorithms involves multiplicative weight updates, reminiscent of Winnow, a canonical online learning algorithm due to [Lit88]. Other relative performance measures for universal prediction involve comparing across systematic variations in the sequence [FV99].

Our goal of removing the switching-rate as a parameter to the class of algorithms considered in [HW98] is similar to Vovk's in [Vov99], though the approach and the comparison class for the bounds differ.

2.2 Regret framework and review of our previous work

In this section we explain the regret framework and summarize our previous work that appeared in [Mon03].

2.2.1 Preliminaries

The learner has access to n experts, $a_1, \ldots, a_n$. Each expert makes a prediction at each time-step over a finite (known) time period $t = 1, \ldots, T$, and each expert's prediction is observed by the learner. We denote the i^{th} expert at time t as $a_{i,t}$ since the algorithm may not have any information as to how the experts arrive at their predictions and what information is available to facilitate the predictions. The

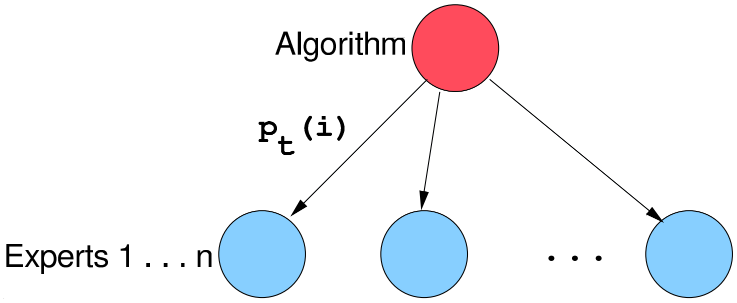


Figure 2-1: The algorithm maintains a distribution over the experts, in order to inform its own predictions.

prediction mechanisms of the experts are unknown; they may vary from one expert to another and may change over time. In this framework, x_t is simply the vector of the experts' predictions at time t.[2] However we will operate only upon the prediction losses, as explained below. We denote the non-negative prediction loss of expert i at time t as $L(i, t)$, where the loss, a function of t, naturally depends on the observation $y_t \in \mathcal{Y}$ at time t. We consider here algorithms that provide a distribution $p_t(i)$, $i = 1, \ldots, n$, over the experts at each time point. The prediction loss of such an algorithm is denoted by $L(p_t, t)$. Figure 2-1 is a schematic of these dependencies.

For the purpose of deriving learning algorithms such as `Static-expert` and `Fixed-share` described in [HW98], we associate the loss of each expert with a predictive probability so that $-\log p(y_t|y_{t-1}, \ldots, y_1, i) = L(i, t)$. We define the loss of any probabilistic prediction to be the log-loss:

$$L(p_t, t) = -\log \sum_{i=1}^{n} p_t(i)\, p(y_t|i, y_1, \ldots, y_{t-1}) = -\log \sum_{i=1}^{n} p_t(i) e^{-L(i,t)} \tag{2.1}$$

Many other definitions of the loss corresponding to $p_t(\cdot)$ can be bounded by a scaled log-loss [HKW98; HW98]. We omit such modifications here as they do not change the essential nature of the algorithms nor their analysis.

The algorithms combining expert predictions can be now derived as simple Bayesian estimation methods calculating the distribution $p_t(i) = P(i|y_1, \ldots, y_{t-1})$ over the experts on the basis of the observations seen so far. $p_1(i) = 1/n$ for any such method as any other initial bias could be detrimental in terms of relative performance guarantees. The Bayesian algorithm updating $p_t(\cdot)$ is defined as follows:

$$p_t(i; \alpha) = \frac{1}{Z_t} \sum_{j=1}^{n} p_{t-1}(j; \alpha) e^{-L(j,t-1)} p(i|j; \alpha) \tag{2.2}$$

where Z_t normalizes the distribution. This is analogous to forward propagation in a

[2] In contrast, in the k-armed bandit problem (e.g. [ACBFS02]), the learner only views the loss of one of the experts per time-step: the arm (expert) chosen.

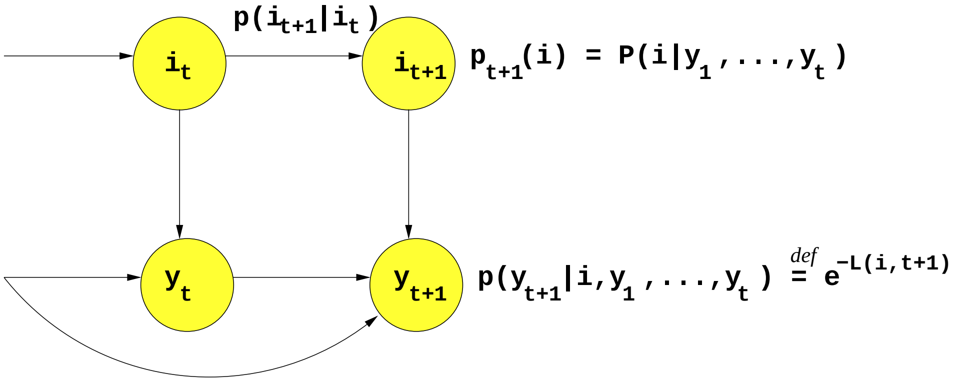

p(i_{t+1}|i_t)
i_t
i_{t+1}
p_{t+1}(i) = P(i|y_1 ,...,y_t)
y_t
y_{t+1}
p(y_{t+1}|i,y_1 ,...,y_t) def= e^{-L(i,t+1)}

Figure 2-2: A generalized Hidden Markov Model (HMM) of probability of the next observation, given past observations, and the current best expert.

generalized HMM (allowing observation dependence on past), such as the one shown in Figure 2-2, in which we model the identity of the best expert for predicting the current observation as a hidden state variable. Updating $p_t(\cdot)$ involves assumptions about how the optimal choice of expert can change with time, $p(i|j;\alpha)$. For simplicity, we consider here only a Markov dynamics where α parameterizes the one-step transition probabilities, and could be an arbitrary transition matrix. To derive previous algorithms, we can use a scalar $0 \leq \alpha \leq 1$ to model the switching-rate between which expert is currently best at prediction. We define

$$P(i|j;\alpha) = \begin{cases} (1-\alpha) & i = j \\ \frac{\alpha}{n-1} & i \neq j \end{cases} \tag{2.3}$$

which corresponds to the `Fixed-share` algorithm, and yields the `Static-expert` algorithm when $\alpha = 0$.

While we have made various probabilistic assumptions (e.g., conditional independence of expert predictions) in deriving the algorithm, the algorithms can be used in a context where no such statistical assumptions about the observation sequence or the experts are warranted. The performance guarantees in this chapter do not require these assumptions.

2.2.2 Upper bound on regret for shifting algorithms

The existing upper bound on the relative loss of the `Fixed-share` algorithm [HW98] is expressed in terms of the loss of the algorithm relative to the loss of the best k-partition of the observation sequence, where the best expert is assigned to each segment. Here is a guarantee which is similar in spirit, but which characterizes the regret relative to the best `Fixed-share` algorithm, parameterized by α^*, where α^* is chosen in hindsight after having seen the observation sequence. The proof technique is different from [HW98] and gives rise to simple guarantees for a wider class of prediction methods, along with a lower bound on this regret.

Lemma 1 (Mon03) *Let* $L_T(\alpha) = \sum_{t=1}^{T} L(p_{t;\alpha}, t)$, $\alpha \in [0,1]$, *be the cumulative loss of the* ***`Fixed-share`*** *algorithm on an arbitrary sequence of observations. Then for any* α, α^**:*

$$L_T(\alpha) - L_T(\alpha^*) = -\log\left[E_{\hat{\alpha}\sim Q}\, e^{(T-1)[D(\hat{\alpha}\|\alpha^*)-D(\hat{\alpha}\|\alpha)]}\right] \tag{2.4}$$

Proof: The cumulative log-loss of the Bayesian algorithm can be expressed in terms of negative log-probability of all the observations:

$$L_T(\alpha) = -\log[\sum_{\vec{s}} \phi(\vec{s})p(\vec{s};\alpha)] \tag{2.5}$$

where $\vec{s} = \{i_1, \ldots, i_T\}$, $\phi(\vec{s}) = \prod_{t=1}^{T} e^{-L(i_t,t)}$, and $p(\vec{s};\alpha) = p_1(i_1)\prod_{t=2}^{T} p(i_t|i_{t-1};\alpha)$.

Consequently, $L_T(\alpha) - L_T(\alpha^*)$

$$\begin{aligned}
&= -\log\frac{\sum_{\vec{s}}\phi(\vec{s})p(\vec{s};\alpha)}{\sum_{\vec{r}}\phi(\vec{r})p(\vec{r};\alpha^*)} = -\log\left[\sum_{\vec{s}}\left(\frac{\phi(\vec{s})p(\vec{s};\alpha^*)}{\sum_{\vec{r}}\phi(\vec{r})p(\vec{r};\alpha^*)}\right)\frac{p(\vec{s};\alpha)}{p(\vec{s};\alpha^*)}\right] \\
&= -\log\left[\sum_{\vec{s}} Q(\vec{s};\alpha^*)\frac{p(\vec{s};\alpha)}{p(\vec{s};\alpha^*)}\right] = -\log\left[\sum_{\vec{s}} Q(\vec{s};\alpha^*)e^{\log\frac{p(\vec{s};\alpha)}{p(\vec{s};\alpha^*)}}\right] \\
&= -\log\left[\sum_{\vec{s}} Q(\vec{s};\alpha^*)e^{(T-1)\left(\hat{\alpha}(\vec{s})\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha}(\vec{s}))\log\frac{1-\alpha}{1-\alpha^*}\right)}\right]
\end{aligned}$$

where $Q(\vec{s};\alpha^*)$ is the posterior probability over the choices of experts along the sequence, induced by the hindsight-optimal switching-rate α^*, and $\hat{\alpha}(\vec{s})$ is the empirical fraction of non-self-transitions in the selection sequence $\vec{s}$. This can be rewritten as the expected value of $\hat{\alpha}$ under distribution Q. □

Upper and lower bounds on regret are obtained by optimizing Q in $\mathcal{Q}$, the set of all distributions over $\hat{\alpha} \in [0,1]$, of the expression for regret.

The upper bound follows from solving: $\max_{Q\in\mathcal{Q}}\left\{-\log\left[E_{\hat{\alpha}\sim Q}\, e^{(T-1)[D(\hat{\alpha}\|\alpha^*)-D(\hat{\alpha}\|\alpha)]}\right]\right\}$ subject to the constraint that α^* has to be the hindsight-optimal switching-rate, i.e. that: (C1) $\quad \frac{d}{d\alpha}(L_T(\alpha) - L_T(\alpha^*))_{|\alpha=\alpha^*} = 0$

Theorem 1 (Mon03) *Let* $L_T(\alpha^*) = \min_\alpha L_T(\alpha)$ *be the loss of the best* ***`Fixed-share`*** *algorithm chosen in hindsight. Then for any* $\alpha \in [0,1]$, $L_T(\alpha) - L_T(\alpha^*) \leq (T-1)\, D(\alpha^*\|\alpha)$, *where* $D(\alpha^*\|\alpha)$ *is the relative entropy between Bernoulli distributions defined by* α^* *and* α.

The bound vanishes when $\alpha = \alpha^*$ and, unlike previous work, it does not depend directly on the number of experts. The dependence on n may appear indirectly through α^*, however. While the regret for `Fixed-share` algorithms is proportional to T, this dependence vanishes for a learning algorithm that is guaranteed to find α such that $D(\alpha^*\|\alpha) \leq O(1/T)$, as we will show in Section 2.2.3.

Theorem 1 follows, as a special case, from an analogous result (Mon03) for algorithms based on arbitrary first-order Markov transition dynamics. In the general case, the regret bound is: $(T-1)\max_i D(P(j|i,\alpha^*) \parallel P(j|i,\alpha))$, where α, α^* are now transition matrices, and $D(\cdot\|\cdot)$ is the relative entropy between discrete distributions. Here is the essence of the proof for the scalar case of Theorem 1.[3]

Proof: Constraint (C1) can be expressed simply as $\frac{d}{d\alpha} L_T(\alpha)_{|\alpha=\alpha^*} = 0$, which is equivalent to $E_{\hat{\alpha}\sim Q}\{\hat{\alpha}\} = \alpha^*$. Taking the expectation outside the logarithm, in Equation 2.4, results in the upper bound. □

2.2.3 Algorithm `Learn-`α and upper bound on regret

In addition to facing the lower bound on regret which we will provide in Section 2.4, the algorithms described above take α as a parameter. Setting α entails an assumption as to the level of non-stationarity of the observation sequence. In contrast, `Learn-`α, the algorithm from (Mon03) which we will describe here, learns the switching-rate, α, simultaneously to updating the probability weighting over the experts.

As illustrated in Figure 2-3, `Learn-`α is a hierarchical algorithm that maintains m "α-experts". The α-experts are `Fixed-share` sub-algorithms each running with a different switching-rate, α, that each maintain a distribution, $p_{t,j}(i) = p_{t;\alpha_j}(i)$, over the original experts given in the problem. `Learn-`α tracks the best "α-expert" using the `Static-expert` algorithm, i.e. it updates its distribution over the `Fixed-share` sub-algorithms with a model which assumes no switching, as specified in Section 2.2.1.

Since the cumulative loss $L_t(\alpha)$ of each `Fixed-share` algorithm running with switching parameter α can be interpreted as a negative log-probability, the posterior distribution over the switching-rate becomes

$$p_t(\alpha) = P(\alpha|y_{t-1},\ldots,y_1) \propto e^{-L_{t-1}(\alpha)} \tag{2.6}$$

assuming a uniform prior over $\alpha \in [0,1]$. As a predictive distribution $p_t(\alpha)$ does not include the observation at the same time point.

We will consider a finite resolution version of this algorithm, allowing only m possible choices for the switching-rate, $\{\alpha_1,\ldots,\alpha_m\}$. For a sufficiently large m and appropriately chosen values $\{\alpha_j\}$, we expect to be able to always find $\alpha_j \approx \alpha^*$ and suffer only a minimal additional loss due to not being able to represent the hindsight-optimal value exactly.

Let $p_{t,j}(i)$ be the distribution over experts defined by the j^{th} `Fixed-share` algorithm corresponding to α_j, and let $p_t^{top}(j)$ be the top-level algorithm producing a weighting over such `Fixed-share` experts. The top-level algorithm is given by

$$p_t^{top}(j) = \frac{1}{Z_t} p_{t-1}^{top}(j) e^{-L(p_{t-1,j},t-1)} \tag{2.7}$$

[3] For more details and the full proof for general transition matrices, we refer to the reader to [Mon03].

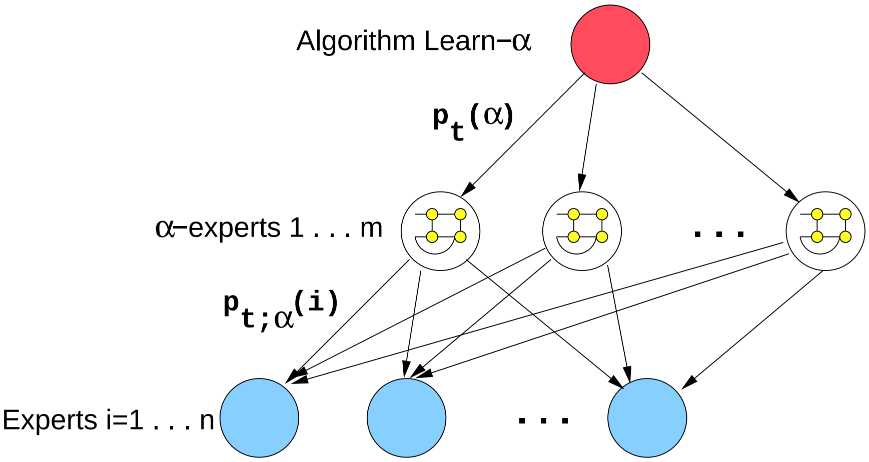


Figure 2-3: The hierarchy of experts and "α-experts," `Fixed-share` algorithms running with different settings of α, maintained by the `Learn-`α algorithm.

where $p_1^{top}(j) = 1/m$, and the loss per time-step becomes

$$L^{top}(p_t^{top}, t) = -\log \sum_{j=1}^{m} p_t^{top}(j) e^{-L(p_{t,j},t)} = -\log \sum_{j=1}^{m} \sum_{i=1}^{n} p_t^{top}(j) p_{t,j}(i) e^{-L(i,t)} \qquad (2.8)$$

as is appropriate for a hierarchical Bayesian method.

Upper bound on regret for `Learn-`α

Here is the extension to Theorem 1 providing an analogous guarantee for the `Learn-`α algorithm.

Corollary 1 (Mon03) *Let L_T^{top} be the cumulative loss of the hierarchical `Learn-`α algorithm using $\{\alpha_1, \ldots, \alpha_m\}$. Then*

$$L_T^{top} - L_T(\alpha^*) \leq \log(m) + (T-1) \min_{j=1,\ldots,m} D(\alpha^* \| \alpha_j) \qquad (2.9)$$

The hierarchical algorithm involves two competing goals that manifest themselves in the regret: 1) the ability to identify the best `Fixed-share` expert, which degrades for larger m, and 2) the ability to find α_j whose loss is close to the optimal α for that sequence, which improves for larger m. The additional regret arising from having to consider a number of non-optimal values of the parameter, in the search, comes from the relative loss bound for the `Static-Expert` algorithm, i.e. the relative loss associated with tracking the best single expert [HW98; LW89]. This regret is simply $\log(m)$ in our context. More precisely, the corollary follows directly from successive application of that single expert relative loss bound, and then our `Fixed-share` relative loss bound (Theorem 1):

$$\begin{aligned} L_T^{top} - L_T(\alpha^*) &\leq \log(m) + \min_{j=1,\ldots,m} L_T(\alpha_j) \qquad (2.10)\\ &\leq \log(m) + (T-1) \min_{j=1,\ldots,m} D(\alpha^* \| \alpha_j) \qquad (2.11)\end{aligned}$$

2.3 Optimal discretization for learning α

Here we show the derivation of the optimal choice of the discrete set $\{\alpha_1, \ldots, \alpha_m\}$, based on optimizing the upper bound on `Learn-`α's relative loss. We start by finding the smallest discrete set of switching-rate parameters so that any additional regret due to discretization does not exceed $(T-1)\delta$, for some threshold δ. In other words, we find $m = m(\delta)$ values $\alpha_1, \ldots, \alpha_{m(\delta)}$ such that

$$\max_{\alpha^* \in [0,1]} \min_{j=1,\ldots,m(\delta)} D(\alpha^* \| \alpha_j) = \delta \qquad (2.12)$$

The resulting discretization, a function of δ, can be found algorithmically as follows. First, we set α_1 so that $\max_{\alpha^* \in [0,\alpha_1]} D(\alpha^* \| \alpha_1) = D(0 \| \alpha_1) = \delta$ implying that $\alpha_1 = 1 - e^{-\delta}$. Each subsequent α_j is found conditionally on α_{j-1} so that

$$\max_{\alpha^* \in [\alpha_{j-1},\alpha_j]} \min\{D(\alpha^* \| \alpha_{j-1}), D(\alpha^* \| \alpha_j)\} = \delta \qquad (2.13)$$

The maximizing α^* can be solved explicitly by equating the two relative entropies giving

$$\alpha^* = \log(\frac{1-\alpha_{j-1}}{1-\alpha_j}) \left(\log(\frac{\alpha_j}{\alpha_{j-1}} \frac{1-\alpha_{j-1}}{1-\alpha_j}) \right)^{-1} \qquad (2.14)$$

which lies within $[\alpha_{j-1}, \alpha_j]$ and is an increasing function of the new point α_j. Substituting this α^* back into one of the relative entropies we can set α_j so that $D(\alpha^* \| \alpha_{j-1}) = \delta$. The relative entropy is an increasing function of α_j (through α^*) and the solution is obtained easily via, e.g., bisection search. The iterative procedure of generating new values α_j can be stopped after the new point exceeds $1/2$; the remaining levels can be filled-in by symmetry so long as we also include $1/2$. The resulting discretization is not uniform but denser towards the edges; the spacing around the edges is $O(\delta)$, and $O(\sqrt{\delta})$ around $1/2$.

For small values of δ, the logarithm of the number of resulting discretization levels, or $\log m(\delta)$, closely approximates $-1/2 \log \delta$. We can then optimize the regret bound (2.9): $-1/2 \log \delta + (T-1)\delta$, yielding $\delta^* = 1/(2T)$, and $m(\delta^*) = \sqrt{2T}$. Thus we will need $O(\sqrt{T})$ settings of α.

Optimized regret bound for `Learn-`α

The optimized regret bound for `Learn-`$\alpha(\delta^*)$ is thus (approximately) $\frac{1}{2} \log T + c$, which is comparable to analysis of universal coding for word-length T [KT81]. The optimal discretization for learning the parameter is not affected by n, the number of original experts. Unlike regret bounds for Fixed-share, the value of the bound does not depend on the observation sequence. And notably, in comparison to the lower bound on Fixed-share's performance, which we will prove in the next section, Learn-α's regret is at most logarithmic in T.

2.4 A lower bound on regret for shifting algorithms

The relative losses obviously satisfy $L_T(\alpha) - L_T(\alpha^*) \geq 0$ providing a trivial lower bound. Any non-trivial lower bound on the regret cannot be expressed only in terms of α and α^*, but needs to incorporate some additional information about the losses along the observation sequence. We express the lower bound on the regret as a function of the relative quality β^* of the minimum α^*:

$$\beta^* = \frac{\alpha^*(1-\alpha^*)}{T-1} \frac{d^2}{d\alpha^2} L_T(\alpha)_{|\alpha=\alpha^*} \tag{2.15}$$

where the normalization guarantees that $\beta^* \leq 1$. $\beta^* \geq 0$ for any α^* that minimizes $L_T(\alpha)$.

The lower bound is found by solving: $\min_{Q \in \mathcal{Q}} \left\{ -\log \left[E_{\hat{\alpha} \sim Q}\, e^{(T-1)[D(\hat{\alpha}\|\alpha^*) - D(\hat{\alpha}\|\alpha)]} \right] \right\}$ subject to both constraint (C1) and (C2) $\frac{d^2}{d\alpha^2}(L_T(\alpha))_{|\alpha=\alpha^*} = \frac{\beta^*(T-1)}{\alpha^*(1-\alpha^*)}$

Theorem 2 *Let β^* and α^* be defined as above based on an arbitrary observation sequence, and $q_1 = [1 + \frac{T-1}{1-\beta^*} \frac{1-\alpha^*}{\alpha^*}]^{-1}$ and $q_0 = [1 + \frac{T-1}{1-\beta^*} \frac{\alpha^*}{1-\alpha^*}]^{-1}$. Then*

$$L_T(\alpha) - L_T(\alpha^*) \geq -\log \left[E_{\hat{\alpha} \sim Q}\, e^{(T-1)[D(\hat{\alpha}\|\alpha^*) - D(\hat{\alpha}\|\alpha)]} \right] \tag{2.16}$$

where $Q(1) = q_1$ and $Q((\alpha^* - q_1)/(1-q_1)) = 1 - q_1$ whenever $\alpha \geq \alpha^*$; $Q(0) = q_0$ and $Q(\alpha^*/(1-q_0)) = 1 - q_0$ otherwise.

The upper and lower bounds agree for all $\alpha, \alpha^* \in (0,1)$ when $\beta^* \to 1$. In other words, $\beta^* \to 1$ results in a guaranteed loss specified by the matching upper and lower bounds. Thus there may exist observation sequences on which Fixed-share, using $\alpha \neq \alpha^*$, must incur regret linear in T.

Proof: The proof is similar to the upper bound case except that we minimize the expression for $L_T(\alpha) - L_T(\alpha^*)$ with respect to distributions Q subject to the constraints imposed by α^* and β^*. I.e. we minimize the regret, the expression from Lemma 1:

$$\begin{aligned} L_T(\alpha) - L_T(\alpha^*) &= -\log E_{\hat{\alpha} \sim Q} \left[\exp \left\{ T' \big(\hat{\alpha} \log \frac{\alpha}{\alpha^*} + (1-\hat{\alpha}) \log \frac{1-\alpha}{1-\alpha^*} \big) \right\} \right] \\ &= -\log E_{\hat{\alpha} \sim Q} \big[f(\hat{\alpha}; \alpha, \alpha^*) \big] \end{aligned} \tag{2.17}$$

where $f(\hat{\alpha}; \alpha, \alpha^*) = \exp \left\{ T' \big(\hat{\alpha} \log \frac{\alpha}{\alpha^*} + (1-\hat{\alpha}) \log \frac{1-\alpha}{1-\alpha^*} \big) \right\}$ and $T' = T - 1$. The minimization is carried out with respect to the distributions Q subject to

$$(1) \qquad \frac{d}{d\alpha} L_T(\alpha)_{|\alpha=\alpha^*} = 0 \tag{2.18}$$

$$(2) \qquad \frac{d^2}{d\alpha^2} L_T(\alpha)_{|\alpha=\alpha^*} = \frac{\beta^*(T-1)}{\alpha^*(1-\alpha^*)} \tag{2.19}$$

These constraints can be expressed as expectation constraints involving Q. As in the upper bound context, the first condition is equivalent to $E_{\hat{\alpha}\sim Q}\{\hat{\alpha}\} = \alpha^*$. To simplify the second condition we rely on the following expression for the second derivative, applying the first constraint, $E_{\hat{\alpha}\sim Q}\{\hat{\alpha}\} = \alpha^*$

Claim 1 *The second derivative of the cumulative loss, $L_T(\alpha)$, of the* `Fixed-share` *algorithm, is of the following form, around $\alpha = \alpha^*$, where $\alpha^* = \min_\alpha L_T(\alpha)$.*

$$\frac{d^2}{d\alpha^2} L_T(\alpha)_{|\alpha=\alpha^*} \quad = \quad \frac{T'^2}{\alpha^{*2}(1-\alpha^*)^2}\left[\frac{\alpha^*(1-\alpha^*)}{T'} - E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right]\right] \tag{2.20}$$

The proof is given in Appendix A.

The second constraint is therefore equivalent to

$$E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right] = \frac{(1-\beta^*)\alpha^*(1-\alpha^*)}{T'} \equiv \beta_2^* \tag{2.21}$$

Let $\mathcal{Q}$ be the set of all distributions over $[0,1]$. Using the fact that minimizing $-\log(\cdot)$ is the same as maximizing the argument, we write the optimization problem for Q as follows

$$\max_{Q\in\mathcal{Q}} E_{\hat{\alpha}\sim Q}\, f(\hat{\alpha}; \alpha, \alpha^*) \tag{2.22}$$

subject to $E_{\hat{\alpha}\sim Q}\{\hat{\alpha}\} = \alpha^*$ and $E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right] = \beta_2^*$. We find the maximizing Q in closed form and substitute the result for the expression of the regret to yield the lower bound. Note that we have relaxed the problem slightly by considering the set of all distributions $\mathcal{Q}$ rather than those possibly induced by the observed sequence. The resulting lower bound on the regret will nevertheless be valid but slightly weaker.

We first identify the form that the optimum Q must take by introducing Lagrange multipliers for the two equality constraints. The Lagrange multipliers can take any real values, since they encode equality constraints. We avoid introducing two additional Lagrange multipliers, for the positivity constraint and the constraint that Q must integrate to one, by performing the optimization of Q only over probability distributions, as mentioned above.

$$\begin{aligned} J(Q,\vec{\lambda}) &= E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha};\alpha,\alpha^*)\right] - \lambda_1(E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}\right] - \alpha^*) - \lambda_2\Big(E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right] - \beta_2^*\Big) \\ &= E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha};\alpha,\alpha^*) - \lambda_1(\hat{\alpha}-\alpha^*) - \lambda_2(\hat{\alpha}-\alpha^*)^2 + \lambda_2\beta_2^*\right] \end{aligned} \tag{2.23}$$

where the second step is by linearity of expectation. The original optimization can thus be written as

$$\max_Q \min_\lambda J(Q, \vec{\lambda}) \tag{2.24}$$

This expression can in turn be upper bounded as follows:

$$\max_Q \min_\lambda J(Q, \vec{\lambda}) \leq \min_\lambda \max_Q J(Q, \vec{\lambda}) \tag{2.25}$$

So by optimizing the right hand side, we will be upper bounding the argument to $-\log(\cdot)$ in the expression for regret (2.17), and thus lower bounding the regret. Thus in order to obtain the bound, it is not necessary to invoke Strong Duality.

The optimizing Q will just place mass at the the $\hat{\alpha}$ values that maximize this quantity. So we can define an objective

$$\begin{aligned} G(\hat{\alpha}, \vec{\lambda}) &= f(\hat{\alpha}; \alpha, \alpha^*) - \lambda_1(\hat{\alpha} - \alpha^*) - \lambda_2(\hat{\alpha} - \alpha^*)^2 + \lambda_2 \beta_2^* \\ \frac{\partial}{\partial \hat{\alpha}} G(\hat{\alpha}, \vec{\lambda}) &= f'(\hat{\alpha}; \alpha, \alpha^*) - \lambda_1 - 2\lambda_2 \hat{\alpha} \\ \frac{\partial^2}{\partial \hat{\alpha}^2} G(\hat{\alpha}, \vec{\lambda}) &= f''(\hat{\alpha}; \alpha, \alpha^*) - 2\lambda_2 \end{aligned} \tag{2.26}$$

The maxima of G on the interval will either occur at the endpoints, or at values of $\hat{\alpha}$ in the interval where the first derivative of G vanishes and the second derivative is non-positive. In order to characterize such points, we first consider the form of $f''(\hat{\alpha}; \alpha, \alpha^*)$. $f(\hat{\alpha}; \alpha, \alpha^*)$ is of the form $c_1 \exp\{c_2 \hat{\alpha}\}$ where $c_1 \geq 0$, and $c_2 \in \mathbb{R}$ is determined by the relation of α to α^*. So $f''(\hat{\alpha}; \alpha, \alpha^*) = c_1 c_2^2 \exp\{c_2 \hat{\alpha}\}$, and is therefore convex in $\hat{\alpha}$, strictly positive, and strictly monotonic (increasing for $c_2 > 0$ and decreasing for $c_2 < 0$)[4]. If $\lambda_2 \leq 0$ then $\frac{\partial^2}{\partial \hat{\alpha}^2} G(\hat{\alpha}, \vec{\lambda})$ will be convex in $\hat{\alpha}$, and still strictly positive, so the condition for maxima will not hold. Thus solutions will be such that $\lambda_2 > 0$. Since the second derivative is strictly monotonic, there will be at most one value of $\hat{\alpha}$ on the interval at which it is zero. I.e. for any fixed $c_1 \geq 0$, $c_2 \in \mathbb{R}$, and $\lambda_2 > 0$, there will exist one value of $\hat{\alpha}$ that solves:

$$0 = \frac{\partial^2}{\partial \hat{\alpha}^2} G(\hat{\alpha}, \vec{\lambda}) = c_1 c_2^2 \exp\{c_2 \hat{\alpha}\} - 2\lambda_2 \tag{2.27}$$

If this value of $\hat{\alpha}$ is in $[0, 1]$ then there can exist a second maximum of G along the interval, aside from a maximizing endpoint (if any). This is because when the second derivative of G is zero, G changes from convex to concave, or vice-versa. The strict monotonicity of the second derivative also entails that any maximum of G on the interval will be attained at a point; not along a subinterval. This provides an upper bound of two on the number of maximizing $\hat{\alpha}$ values along the interval. The trivial cases in which both points are at endpoints of the interval, or there is only one

[4] $c_2 = 0 \Rightarrow \alpha = \alpha*$, in which case there is no regret.

maximizing point, violate the variance constraint, thus yielding trivial bounds, and will be covered implicitly.

Depending on the relation of α to α^* we need to consider just the two points 0 and $a \in [0,1]$, or 1 and $a' \in [0,1]$. Q and the corresponding points can now be solved based on the constraints alone. When $\alpha < \alpha^*$, we place probability mass q_0 on 0 and $1 - q_0$ on a. Thus

$$\begin{aligned} 0 \times q_0 + a(1 - q_0) &= \alpha^* && (2.28) \\ q_0(0 - \alpha^*)^2 + (1 - q_0)(a - \alpha^*)^2 &= \frac{(1 - \beta^*)\alpha^*(1 - \alpha^*)}{T'} && (2.29) \end{aligned}$$

and solving for a and q_0, gives

$$a = \frac{\alpha^*}{1 - q_0}, \quad q_0 = \frac{1}{1 + \frac{T'}{1-\beta^*} \frac{\alpha^*}{1-\alpha^*}} \qquad (2.30)$$

or $Q(0) = q_0$ and $Q(a) = Q(\alpha^*/(1 - q_0)) = 1 - q_0$. Analogously, when $\alpha > \alpha^*$, we place probability mass q_1 on 1 and $(1 - q_1)$ on a':

$$a' = \frac{\alpha^* - q_1}{1 - q_1}, \quad q_1 = \frac{1}{1 + \frac{T'}{1-\beta^*} \frac{1-\alpha^*}{\alpha^*}} \qquad (2.31)$$

The bound follows by substitution. □

2.5 Application to wireless energy management

In this section we discuss an application of `Learn-`α to the problem of managing the tradeoff between energy consumption and performance in wireless devices implementing the IEEE 802.11 standard [IEE99]. We performed a preliminary study for this application in [Mon03]; here we strengthen and validate the application by applying it in network simulation. To save energy, the 802.11 specification proposes a power-saving mode (PSM), where a device can sleep to save energy, periodically waking up to receive packets from a neighbor (e.g., an access point) that may have buffered packets for the sleeping device. Previous work has shown that a fixed polling time for waking up degrades the performance of Web transfers [KB02], because network activity is bursty and time-varying. We apply algorithm `Learn-`α to this problem and show, using `ns-2` simulation and trace analysis, that it is able to adapt well to network activity. Our learning power-saving algorithm, LPSM, guides the learning using a loss function that combines the increased latency from potentially sleeping too long and the wasted use of energy in waking up too soon. In our `ns-2` simulations, LPSM saved 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average latency by a factor of 1.02, and not more than 1.2. LPSM is straightforward to implement within the 802.11 PSM framework.

2.5.1 The 802.11 energy/performance tradeoff

Energy is an important resource in mobile computing systems. Because processing, storage, display activity, and communication all consume energy, energy-saving techniques targeted at improving these subsystems have received significant attention in recent years. Impressive advances in hardware design and operating systems have greatly reduced the energy consumed by the processing and storage subsystems, and have led to the wireless network becoming a significant consumer of energy in many mobile devices. This trend is especially true for handheld mobile devices and nodes in wireless ad hoc and sensor networks.

Most wireless network interfaces consume energy not only while transmitting or receiving data, but also when they are simply awake. Therefore, to save energy, most modern wireless interfaces support a *power saving mode* (PSM). In abstract terms, the PSM primitive allows an interface to be in one of two states, SLEEP or AWAKE. SLEEP is a low-power state, but the interface cannot send or receive data in this state. In contrast, AWAKE allows data flow, but is a higher-power state. Depending on the actual device, these two states may differ in power consumption by between a factor of 10 and 50. For instance, in some current 802.11 cards, the ratio is about a factor of 20 (1 W v. 50 mW) [FN01; CJBM02].

With the PSM primitive, power-saving algorithms can save energy by keeping the wireless interface in the SLEEP state for as long as possible. A SLEEPing device periodically wakes up and polls its neighbors (either an access point in "infrastructure" mode, or a neighboring node in "ad hoc" mode) for packets.[5] To avoid excessive packet loss, the neighbor must therefore buffer packets for each SLEEPing receiver. Then, the neighbor sends these buffered packets when it receives a poll from a waking receiver.

Power-saving algorithms built on top of the PSM primitive introduce a tradeoff between the amount of energy saved and the degree of performance degradation. If a device awakens and finds no data buffered for it, then it could have slept for longer and saved some more energy. On the other hand, if any packets are buffered when the interface awakens, then the latency to obtain those packets would be larger than if the network interface had been awake instead of asleep. This increased latency degrades not just the latency of the on-going data transfers, but often the throughput as well.

This section addresses the problem of designing an algorithm by which a device can decide when to SLEEP and when to be AWAKE. Our goal is to devise an algorithm that manages the tradeoff between energy consumption and data transfer latency in a principled, well-specified way, such that users or application designers can specify their desired operating point. The goal of a managing the tradeoff in a principled manner is motivated in part by Krashinsky and Balakrishnan's work on the Bounded SlowDown (BSD) algorithm [KB02], which demonstrates that the IEEE 802.11's non-adaptive polling time strategy [IEE99] degrades both the latency and the throughput of TCP transfers. Consistent with that work, we focus our algorithm on Web-like workloads because they are currently the dominant workloads for many mobile devices.

[5] This is an abstract model: some implementations first have the neighbor advertise information before the polls occur.

Outline of contributions

We develop a PSM algorithm called **LPSM (Learning PSM)** by applying `Learn-`α to determine a device's sleep/awake schedule. In order to adapt the schedule in response to observations of current network activity, we instantiate each expert as a deterministic setting of the polling time. At each decision epoch, the polling time chosen is the weighted sum of the experts' times, where the weights are updated by `Learn-`α. LPSM makes no assumptions on the distribution of packet arrivals and network activity.

The first contribution of this section is to show how online machine learning can be used to solve the wireless power-saving problem. The key to this solution is to define a loss function that the `Learn-`α algorithm uses in determining how to update the weights of the experts every time the mobile device awakens. If the device awakens and there is no data present, then the weights of the experts are carefully adjusted such that the next sleep time is longer. Conversely, if any packets were present, the opposite adjustment is made.

The second contribution of this section is a performance evaluation of LPSM in trace-driven network simulation and on traces of real-time Web activity. We compare the performance of both the non-adaptive 802.11 PSM and the BSD algorithm to LPSM. In our experiments using a Web-like request/response workload, LPSM saves 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average slowdown of 2%, and not more than 20%. LPSM is straightforward to implement within the 802.11 PSM framework.

The rest of this chapter is organized as follows. Section 2.5.3 gives the LPSM algorithm. Section 2.5.4 presents several results from trace-driven `ns-2` simulations and trace-based analysis of LPSM, and Section 2.5.5 concludes with a discussion of our results.

2.5.2 Previous work related to application

Using trace-driven simulations, i.e. simulations that sample from an empirical probability distribution computed from traces of real-time Web activity, Krashinsky and Balakrishnan [KB02] show that the 802.11 PSM algorithm, which uses a fixed polling interval (typically 100 ms) to wake up and check for data, causes response latency for Web-like transfers to be as bad as 2.2× longer than in the absence of any power-saving algorithm. To better manage the tradeoff in question, they propose BSD, an algorithm that uses an adaptive control loop to change polling time based on network conditions. The algorithm uses a parameter, p, and guarantees that the response latency does not ever exceed $(1+p)$ times the response latency without power-saving. Within that constraint and assuming adversarial traffic arrivals, BSD guarantees that the energy consumption is minimized. In contrast, LPSM does not attempt to guarantee bounded latency under adversarial traffic arrivals; instead, our approach is to explicitly encode a tradeoff between energy and latency and give an online learning algorithm that manages this tradeoff.

Simunic et al. formulate the wireless power-saving problem as policy learning in a Markov Decision Process (MDP) [SBGM00]. Their algorithm is not an online algorithm since the linear programming algorithm used to resolve the policy over any given time period requires access to data over that entire period. They also assume that network activity is stationary. In the MDP there is fixed distribution governing the selection of next states, given the current state and action. For any fixed policy such as the optimal policy in this framework, the network activity is modeled as a Markov process. This model is not an ideal one for a mobile node, since the network activity need not conform to a Markov process of any finite order k.

Simunic et al. refer to Chung et al. [CBM99] for the solution in non-stationary environments. That work proposes "policy interpolation," however it still assumes that the underlying process is Markovian, even though it may initially *appear* non-stationary due to a lack of observations. They then propose to learn the associated MDP parameters sequentially [CBM99]. Another machine learning approach to this problem was proposed by Steinbach, using Reinforcement Learning [Ste02]. This approach also imposes the assumption that network activity has the Markov property which, as discussed above, is unrealistic. These previous learning approaches differ from ours in that LPSM does not make any Markovian or stationarity assumptions, nor require any *a priori* knowledge of the phenomenon being learned. LPSM is also simpler to implement in the 802.11 framework.

We focus on online learning algorithms of the type described in Section 2.2. The related work has been discussed in Section 2.1. We consider algorithms that treat network activity as non-stationary, although possibly composed of variable-length periods that exhibit stationarity in some sense. These algorithms are parameterized by the switching-rate of the non-stationary process. In the context of wireless networks, this value cannot be known by a mobile node *a priori*. Thus we will use `Learn-`α to learn the switching-rate parameter online, simultaneous to learning the target concept: the polling time in this context

2.5.3 Application of Algorithm `Learn-`α

The intuition behind our application of `Learn-`α is as follows. The IEEE 802.11 PSM standard is a deterministic algorithm polling at fixed polling time, $T = 100$ ms, which can be viewed as one "expert," who always claims that 100 ms is the polling time that should be used. Clearly the ability to consult a set of experts, in which each expert is a deterministic algorithm using a different T value as its polling time, would enable a more flexible algorithm compared to operating with just one polling time. LPSM maintains a probability distribution over a set of such deterministic experts. The algorithm computes its polling time as a function all the experts, subject to this distribution. It adaptively updates its distribution over experts, based on current network activity.

Not only is this approach more flexible than that of the 802.11 PSM standard, but also it is a promising alternative to approaches like BSD that are based on an adaptive control framework. While BSD adaptively updates its polling time, T, based on network conditions, it evolves only one T value. In contrast, LPSM maintains and updates a set of n T values simultaneously, instead of just one. Although the form of the updates is different than that of BSD, the LPSM algorithm explores across a range of T values in parallel, which should allow it to choose a better instantaneous T value to manage the tradeoff. Since sequential exploration for the sake of learning can incur loss, LPSM's parallel exploration should allow it to better manage the energy/latency tradeoff than previous approaches. Additionally, since Learn-α allows rapid switching between experts, it is able to handle sudden changes in the observed process, for example a sudden burst of network activity, in this context.

The protocol we propose for a node using LPSM is similar to that of 802.11 PSM in mainly sleeping except for polls. The main difference in using LPSM is that the sleep times would be of variable length. Additionally, after retrieving any packets that may have arrived from the neighbor's buffer, the node will only stay awake if the link continues to be active.

The intuition behind how LPSM updates its distribution over experts is that different experts should gain or lose favor based on current network activity. Upon awakening, if many packets had been buffered, then the algorithm should adjust and sleep for less time. On the other hand if only a few packets were received, the algorithm can sleep for longer, in order to save energy. Below we will present an objective function aimed at simultaneously minimizing energy and slowdown. After each observation, LPSM updates the weight of each expert based on that expert's loss with respect to this objective, which is a measure of how well the polling time that the expert proposes would manage the energy/latency tradeoff under the current network conditions.

Instantiation of experts and prediction function

We apply the Learn-α algorithm exactly as in 2.2.3, unless otherwise stated. To do so there are several quantities we must instantiate. The algorithms of the type discussed in Section 2.2 have performance guarantees with respect to the best expert in the expert set given. These guarantees make no assumptions on the set of experts, as they are worst case guarantees computed as if the expert set is just a black box, and could even contain algorithms that are adversarial. Thus when applying such learning algorithms to a specific problem, we can achieve additional gains from the performance guarantees, by choosing experts that are actually helpful for the problem domain in question. In the wireless energy management problem, we use n experts, each corresponding to a different but fixed polling times, $T_i : i \in \{1 \ldots n\}$. The experts form a discretization over the range of possible polling times.

Unlike many previous problems where online learning has been applied, our problem imposes the constraint that the learning algorithm can only receive observations, and perform learning updates, when the mobile device is awake. Thus our subscript t here signifies only wake times, not every time epoch during which bytes might arrive.

To apply this type of online learning algorithm to this problem, we instantiate the prediction function using the weighted mean. Thus the algorithm's polling time T_t, i.e. its prediction of the current amount of time it ought to sleep for is:

$$T_t = \sum_{i=1}^{n} p_t(i) T_i \tag{2.32}$$

where $p_t(i)$ is its current distribution over experts.

Objective function

The performance bounds on `Learn-`α hold regardless of the choice of loss function, $L(i, t)$. In this application to the wireless power-saving problem, we instantiate the loss function as follows. The objective at each learning iteration is to choose the polling time T_t that minimizes both the energy usage of the node, and the network latency it introduces by sleeping. We define the loss function so as to reflect the tradeoff inherent in these conflicting goals. Specifically, we will design a loss function that is directly proportional to appropriate estimates of these two quantities. It is important to note that the algorithm is modular with respect to this function, so while we suggest several loss functions that are proportional to the energy versus slowdown tradeoff, there are many possible functions. If one wishes to model the tradeoff differently, one need only specify an objective function, `Learn-`α will learn using that objective.

The BSD [KB02] approach to managing this tradeoff uses an objective of:

$$\min E \quad : \quad L \leq (1+p)L_{opt} : p > 0 \tag{2.33}$$

for a scalar threshold L_{opt}, where E is an energy term and L is a latency term. In contrast, our approach is not to apply a threshold on either of these quantities, but instead to propose a function that encodes the tradeoff between latency and energy.

The observation that the learning algorithm receives upon awakening is the number of bytes that arrived while it slept during the previous interval. We denote this quantity as I_t, and the length of time that the node slept upon awakening at time t, as T_t. One model for energy usage is proportional to $\frac{1}{T_t}$. This is based on the design that the node wakes only after an interval T_t to poll for buffered bytes, and the fact that it consumes less energy when asleep than awake. Additionally there is a constant spike of energy needed to change from sleeping to the awake state, so the more often a node polls during a given time horizon, the higher the energy consumption. An alternative model for the energy consumption would be to use the term $\frac{1}{\log T_t}$. This is a larger penalty, in order to account for additional energy used, such as while staying awake when the link is active, which is not reflected in how long the node chooses to sleep for at each learning iteration. For clarity, we will just show the first energy model in our equations below, but we will report on the evaluation of both models.

We model the latency introduced into the network due to sleeping for T_t ms as proportional to I_t. In other words, there is increased latency for each byte that was buffered during sleep, by the amount of its wait-time in the buffer. Since our learning algorithm does not perform measurement or learning while asleep, and only observes the aggregated number of bytes that arrived while it slept, we have to approximate the amount of total latency its chosen sleep time introduced, based on the individual buffer wait-times of each of the I_t bytes. To estimate the average latency that each of the I_t buffered bytes would have experienced, without knowledge of the byte arrival times, we can use the maximal entropy assumption. This models all the bytes as arriving at a uniform rate during the sleep interval, T_t, in which case the average wait-time per byte would be $\frac{T_t}{2}$. Thus the total latency introduced by sleeping for T_t is approximated by the number of bytes that arrived in that time, times the average wait-time per byte, yielding $\frac{T_t}{2} I_t$.

The form of our proposed loss function is thus

$$L(t) = \gamma \frac{T_t I_t}{2} + \frac{1}{T_t} \qquad : \gamma > 0 \tag{2.34}$$

In our weight updates however, we apply this loss function to each expert i, approximating the loss that would have resulted from the algorithm using T_i instead of T_t as its polling time. So the equivalent loss per expert i is:

$$L(i,t) = \gamma \frac{I_t T_i^2}{2 T_t} + \frac{1}{T_i} \tag{2.35}$$

where $\frac{T_i}{T_t}$ scales I_t to the number of bytes that would have arrived had the node slept for time T_i instead of T_t. Note that the objective function is a sum of convex functions and therefore admits a unique minimum.

The parameter $\gamma > 0$ allows for scaling between the units for measuring packet arrivals and the polling time, as well as the ability to encode a preference for the ratio between energy and latency that the particular node, protocol or host network favors. This can also be viewed as follows. The two optimizations we are trying to perform are the conflicting goals of minimizing energy usage, and minimizing additional latency caused by buffering. The tradeoff can be formulated as an energy minimization problem, subject to the constraint that latency be upper bounded by a threshold. In this formulation, γ is the Lagrange multiplier enforcing the latency constraint. To clarify the relation between using latency as a constraint and including it as a term in the loss, note that increasing γ will monotonically increase the effect of latency on the loss, which the algorithm seeks to minimize.

Note that this is one of many possible loss functions that are proportional to the tradeoff that must be optimized for this application.

To summarize, the LPSM algorithm proceeds as shown in Figure 2-4.[6]

2.5.4 Performance evaluation

In this section we study the performance of LPSM with respect to the tradeoff between energy savings and performance degradation using trace-driven simulations. We also compare the performance of LPSM to previously proposed power-saving algo-

[6] Implementation is optimized to be more compact.

```
Algorithm LPSM
Initialization:
$\forall j,\ p_1(j) \leftarrow \frac{1}{m}$
$\forall i, j,\ p_{1,j}(i) \leftarrow \frac{1}{n}$
Upon $t$th wakeup:
$T_t \leftarrow$ number of ms just slept
$I_t \leftarrow$ # bytes stored at neighbor
Retrieve buffered data
For each $i \in \{1 \ldots n\}$:
    $\text{Loss}[i] \leftarrow \gamma \frac{I_t T_i^2}{2T_t} + \frac{1}{T_i}$
For each $j \in \{1 \ldots m\}$:
    $\text{AlphaLoss}[j] \leftarrow -\log \sum_{i=1}^{n} p_{t,j}(i)\, e^{-\text{Loss}[i]}$
    $p_{t+1}(j) \leftarrow p_t(j) e^{-\text{AlphaLoss}[j]}$
    For each $i \in \{1 \ldots n\}$:
        $p_{t+1,j}(i) \leftarrow \sum_{k=1}^{n} p_{t,j}(k)\, e^{-\text{Loss}[k]}\, P(i|k; \alpha_j)$
    Normalize $P_{t+1,j}$
    $\text{PollTime}[j] \leftarrow \sum_{i=1}^{n} p_{t+1,j}(i)\, T_i$
Normalize $P_{t+1}$
$T_{t+1} \leftarrow \sum_{j=1}^{m} p_{t+1}(j)$ PollTime[$j$]
Goto sleep for $T_{t+1}$ ms.
```

Figure 2-4: Conceptual view of Algorithm LPSM.

rithms [KB02]. Additionally, we use loss function units to further study the behavior of LPSM on traces of real-time Web activity.

Network simulation evaluation framework

In this section, we study the performance of LPSM with respect to the energy/latency tradeoff. After describing the setup of our `ns-2` simulation, we compare the performance of LPSM with 802.11 static PSM and BSD [KB02].

The simulation framework that we used to evaluate LPSM is the same as that which has been used to evaluate previous 802.11 power-saving algorithms such as BSD [KB02], [Kra02]. We briefly summarize that framework here. We use a simple 3-node topology: a mobile client accesses a Web server via an 802.11 access point. The bandwidth between the mobile host and the base station is 5 Mbps and the latency is 0.1 ms; the bandwidth between the base station and the Web server is 10 Mbps and the latency is 20 ms. As in previous work [KB02], we do not model the details of the 802.11 MAC protocol. Rather, we model sleep times with some simple modifications to `ns-2` [Kra02]. A sleeping device does not forward any packets, but buffers them until the device wakes up again. A device wakes up whenever it has data to forward to the access point and sleeps for the interval determined by LPSM. It remains awake after polling, only if the link remains active. Since the 802.11 PSM framework assumes that nodes wakeup and poll only on 100 ms boundaries,

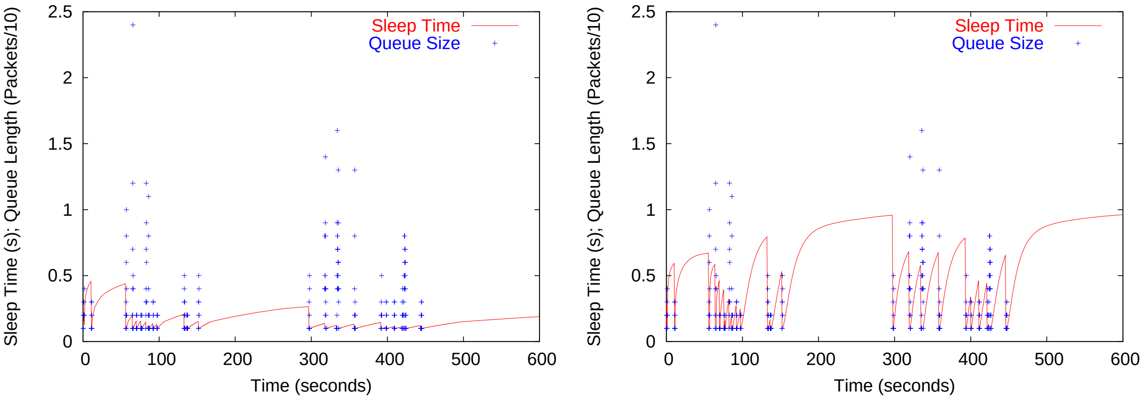


Figure 2-5: Evolution of sleep times with LPSM. a) $1/T$ b) $1/\log T$.

for the sake of synchronization issues between sleeping and awake nodes, we rounded the sleep interval determined by LPSM to the nearest 100 ms. We modeled energy consumption using previous estimates [KB02]: 750 mW when the interface is awake, 50 mW when the interface is asleep, and 1.5 mJ for each beacon.

We report results from an experiment that involved simulating 100 Web transfers from the client to the Web server over approximately 1.5 hours of simulation time. To verify the robustness of our results, we also ran 4 independent experiments of 500 Web transfers, each over approximately 8.5 hours of simulation time. As in previous work [KB02], we modeled Web browsing behavior using the `ns-2` HTTP-based traffic generator, using `FullTcp` connections. In this model, a Web transfer consists of several steps: (1) a client opens a TCP connection and sends a request; (2) after some delay, the server sends a response and several embedded images; the client may open up to 4 parallel TCP connections to retrieve these images; (3) the client waits for some amount of "think time" before making the next request. Also as in previous work [KB02], we randomly selected the parameters for each request based on an empirical distribution [Ber96], and we limited client think time to 1000 seconds.

In these experiments, we configured LPSM with 12 experts spanning the range from 100 ms to 1.2 seconds, at regularly spaced intervals of 100 ms. Thus, the lowest expert, 100 ms, matched the polling time of the current 802.11 PSM standard, and the highest expert was 1.2 seconds. Since a convex combination is upper bounded by its maximum value and lower bounded by its minimum value, we were assured of only using polling times within this range.

As discussed in Section 2.5.3, the learning function is modular with respect to the loss function, as it will seek to minimize whichever loss function one uses to encode the energy/slowdown tradeoff. We experimented with the effects of two different loss functions: (1) the loss function shown in Figure 2-4, with the second term as $1/T$; and (2) a loss function that uses $1/\log T$ as its second term, to penalize energy usage more severely. For the first loss function, we used $\gamma = 1/(1.2 \cdot 10^5)$; for the latter, we used $\gamma = 1/(1.2 \cdot 10^3)$.

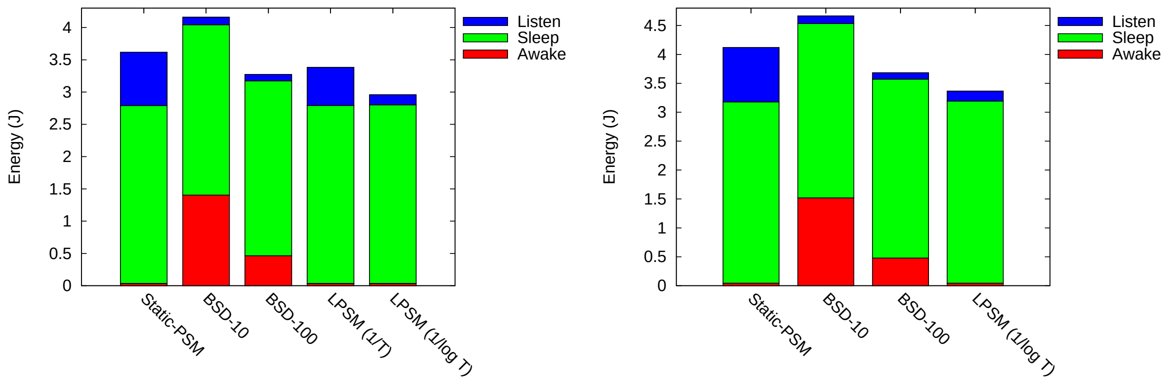


Figure 2-6: Average energy usage per page for various PSM algorithms. a) Results from 100-page trial. b) Results from 4 independent 500-page trials.

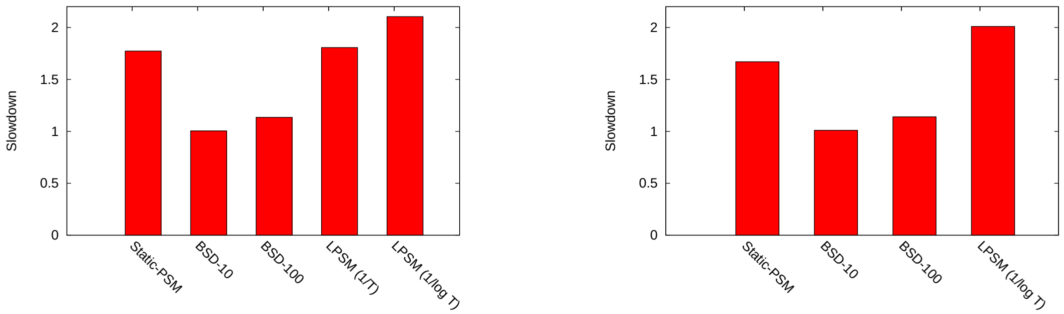


Figure 2-7: Average slowdown over 802.11 without power-saving for various PSM algorithms. a) Results from 100-page trial. b) Results from 4 independent 500-page trials.

Algorithm behavior

Figure 2-5 shows the evolution of sleep times over a portion of the simulation trace. When LPSM discovers that packets are queued at the access point, it quickly reduces the sleep interval to 100 ms. During periods of inactivity, LPSM continually increases the sleep time each time the device wakes up and discovers that no packets are queued for it. These two figures also illustrate how the choice of loss function affects the behavior of the learning algorithm. When the learning algorithm has a loss function that has $1/T$ as its energy term, it backs off much more slowly than when it uses a loss function with $1/\log T$ as its energy term. This makes sense: the latter loss function places relatively more importance on the energy term, the penalty incurred for waking up when no packets are queued for the device.

Performance: energy vs. latency

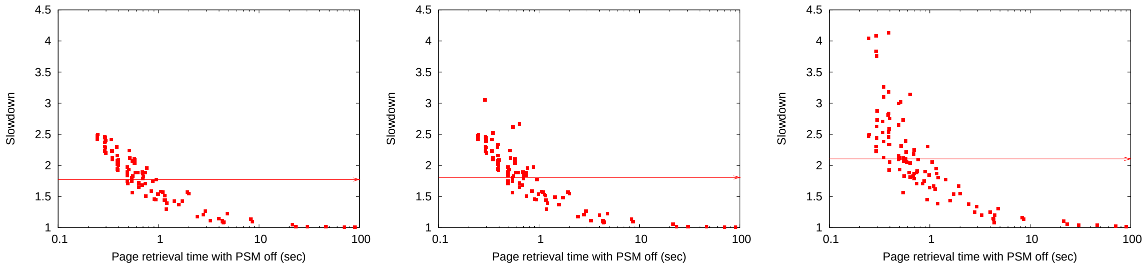
Figures 2-6 a) and 2-7 a) characterize the energy savings and slowdown of LPSM relative to static PSM, which we ran under the same simulation, using the implementation by [Kra02]. As in previous work, slowdown is defined as the ratio of the latency incurred by the PSM algorithm to the latency incurred by using no PSM whatsoever. Our first result is that LPSM consistently decreased per-page energy consumption. For a slight increase, 2%, in slowdown over static PSM, LPSM, with $1/T$ as the energy model, reduces overall energy consumption by about 7% (from 3.62 J to 3.38 J), and energy due to beaconing by almost 30% (from 0.83 J to 0.59 J). On the other hand, LPSM with $1/\log T$ as its energy term, reduces overall energy consumption by nearly 20% (from 3.62 J to 2.95 J) and energy consumption due to beaconing by more than 80% (from 0.83 J to 0.16 J), while increasing slowdown over static PSM by only a factor of 1.19.

To verify the robustness of our results, we also ran longer simulations for LPSM with $1/\log T$ as the energy term. Figures 2-6 b) and 2-7 b) show results averaged over 4 independent 500-page experiments, each one starting the traffic simulator with a different random seed. We also ran static PSM in each of these four simulations, and averaged its results, for a fair comparison. This experiment shows similar results: energy savings of over 18% (from 4.12 J to 3.36 J) and an 82% reduction in energy consumption due to beaconing (from 0.94 J to 0.17 J), while increasing slowdown over static PSM by a factor of only 1.2.

Figures 2-6–2-7 also compare the performace of LPSM with BSD. We ran BSD in the same simulation for both scenarios described above, using the original BSD implementation [Kra02], including averaging over the four runs with the same set of random seeds for traffic generation as used on LPSM. LPSM shows an energy reduction versus most settings of BSD, though higher slowdowns. Notably, the LPSM using $1/\log T$ to model energy, had deeper energy savings than all the previous BSD settings, though it increased slowdown more. It is important to note that LPSM can work well even if the distribution generating the observations of network activity changes with time. Since in simulation, traffic was generated using a stationary distribution, we would expect only better results against BSD in practice. Additionally, in both settings, we ran with m, the number of α-experts in the Learn-α algorithm, optimized for a timescale of 45 minutes. We could expect better results, especially in the longer trials, had it been optimized for the length of the trial. Yet these results lend validity to using LPSM even under resource limitations.

Moreover, LPSM offers several unique advantages over existing approaches. First, because LPSM's determination of appropriate sleep times is based on a loss function, rather than a single parameter, LPSM provides designers sufficiently more flexibility than BSD in exploring the energy/latency tradeoff. It should be possible to simultaneously reduce both energy and slowdown from that of previous approaches, by appropriate choice and calibration of the loss function. Second, because LPSM uses a learning algorithm designed to react well under non-stationarity, we would expect LPSM to perform better than BSD when the distribution generating traffic changes over time, a situation not modeled in this simulation, but which is realistic in practice. These hypotheses deserve further attention and present interesting possibilities for future work.

Figure 2-8 shows the slowdown behavior of various power-saving algorithms for individual Web page downloads. LPSM imposes only a moderate increase in slowdown



(a) Static PSM.

(b) LPSM ($1/T$)

(c) LPSM ($1/\log T$)

Figure 2-8: Average slowdown vs. time to download that page without power-saving. Average slowdown for the entire experiment is shown with a horizontal line. LPSM ($1/T$) imposes only a slight increase in slowdown over static PSM. Using a loss function with the energy term $1/\log T$ saves more energy at the cost of increased slowdown; however, it never increases slowdown over static PSM by more than a factor of 2 for a given page.

over static PSM for various Web pages: only a 2% increase on average, but never more than 20%. Changing the loss function in LPSM to favor energy reduction more will increase the slowdown, as shown in Figure 2-8(c), but even using this loss function never incurs a slowdown of more than a factor of 2 for any given Web page. For longer Web transfers, both LPSM algorithms introduce only negligible slowdown. Some shorter Web transfers impose longer slowdown times than static PSM, but these download times are short anyway, so the additional slowdown that LPSM imposes is not drastic.

Trace-based evaluation framework

We also ran LPSM on traces of real network activity [Ber96]. This evaluation framework differs from the trace-driven simulation in that the network simulation generates traffic from a stationary distribution (created by averaging over the traces [Kra02]). In contrast, running the LPSM algorithm on the traces themselves would allow it to observe real network activity, which could potentially exhibit non-stationarity. Here we present some qualitative results, in units of loss and the weights maintained over experts, to further illustrate the behavior of Learn-α, with respect to individual experts, the best expert and the best sequence of experts, in this problem domain.

We used publicly available traces of network activity from a UC Berkeley home dial-up server that monitored users accessing HTTP files from home [Ber96][7]. Because the traces only provided the start and end times, and number of bytes transferred for each connection, per connection we smoothed the total number of bytes uniformly over 10 ms intervals spanning its duration. In the network trace experiment results below, we ran the algorithm with 10 experts spanning the range of 1000 ms at regularly

[7] These are also the traces we used to synthetically generate Web traffic for the ns-2 simulations

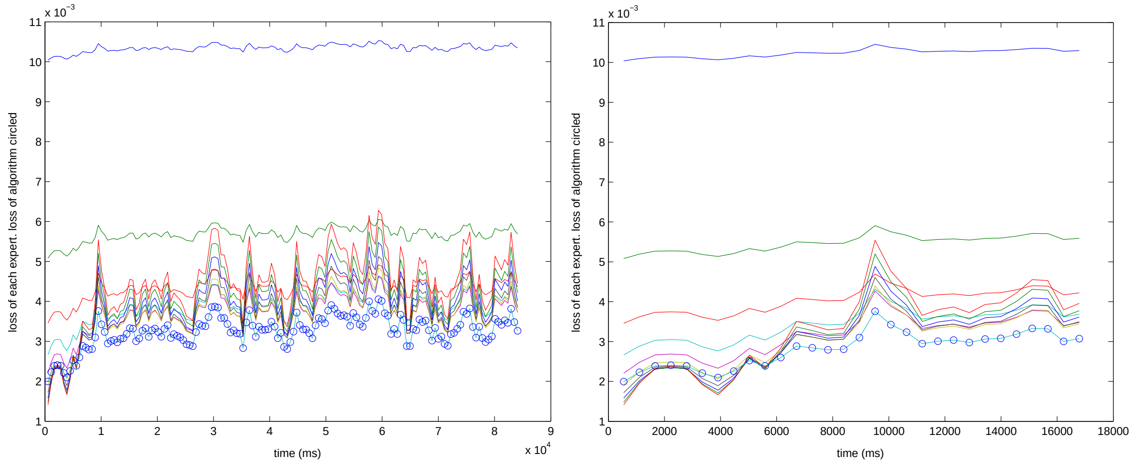


Figure 2-9: These figures show the loss of each expert as a function of time. The circled path is the loss of the algorithm. The right figure zooms in on the earlier iterations.

spaced intervals of 100 ms, and $\gamma = 1.0 \times 10^{-7}$, calibrated to attain polling times within the range of the existing protocol.

Behavior of online learning algorithms

Figure 2-9 shows the loss, on one trace, of each of the experts as a function of time, measured at LPSM's polling times. The circled path is the loss of LPSM. Since the algorithm allows for switching between the experts, it is able to maintain loss close to or better than the loss of the best current expert, even though our results show that the identity of the best current expert changes with time. The graph on the right, which zooms in on the early phases of the run, shows that it takes some time before the algorithm is doing as well as the best current expert. Note however that since the weights start as uniform, the initial polling time is actually the mean of the polling times of each of the experts, so even in the early iterations the algorithm already beats the worse half of the experts, and still tracks the average expert.

Figure 2-10 shows the evolution of the distribution that the algorithm maintains over experts. As expected, the algorithm changes its weighting over the experts, based on which experts would currently be performing best in the observed network activity. Changes in the burstiness level of the arriving bytes are reflected in shifts in which expert currently has the highest weight, starting even in the early learning iterations. The figure on the right, which zooms in on the earlier phases of learning, shows that the algorithm's preference for a given expert can easily decrease and increase in light of network activity. For example after several periods of being the worst expert, after iteration 1600, an expert is able to regain weight as the best expert, as its weight never went to zero.

These results also confirm that for real network traffic, no single deterministic setting of the polling time works well all the time. The algorithm can do better than

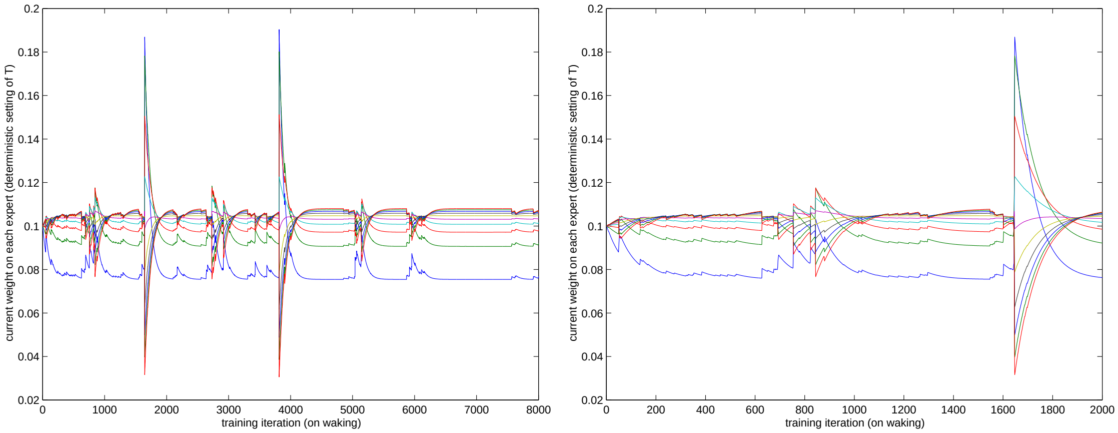


Figure 2-10: These figures show the weights that the algorithm maintains on each expert, per training iteration. The right figure zooms in on the earlier iterations.

any single expert, as it is a convex combination of all the experts, and the updates of the weighting allow the algorithm to track the expert that is currently best. Note that these online adjustments of which expert the algorithm currently favors are closely reflected in the evolution of sleep times graphed in Figure 2-5.

Competitive analysis

To perform competitive analysis of the learning algorithm, we compare it to the hindsight best expert, as well as to the current best expert. Figure 2-11 illustrates how the learning algorithm does in relation to the best fixed expert computed in hindsight, and to the current best expert at each time epoch at which the algorithm does a learning update. The algorithm learns to track and eventually do better than the best fixed expert, in the first figure. The scenario in the second figure is an approximation to the best k-partition, i.e. the best partition of the trace into k parts and choice of the best expert for each part, where k is the actual number of switches in the sequence between which expert is currently best at predicting the observations. If the best expert actually switches at every time step, k can be as large as the length of the full time horizon of the trace. Note that as k increases, the optimal k-partition is harder to track. Here the algorithm does not beat the loss of the sequence of best experts, but is at least able to "track" it, as per the bounds [LW89; HW98]. We see that the algorithm's performance is never too far away from that of the current best expert.

In both comparisons, the performance appears to beat the associated performance guarantees, as there is negligible regret (the instantaneous regret is sometimes even negative). This is unsurprising as the performance guarantees are with respect to an unknown, perhaps even adversarial set of experts. In applying the algorithm however, we were able to *choose* the set of experts to best apply to this problem domain. Additionally since the experts we chose are just a discretization over the parameter to be learned (the polling time), and since we instantiate prediction as a

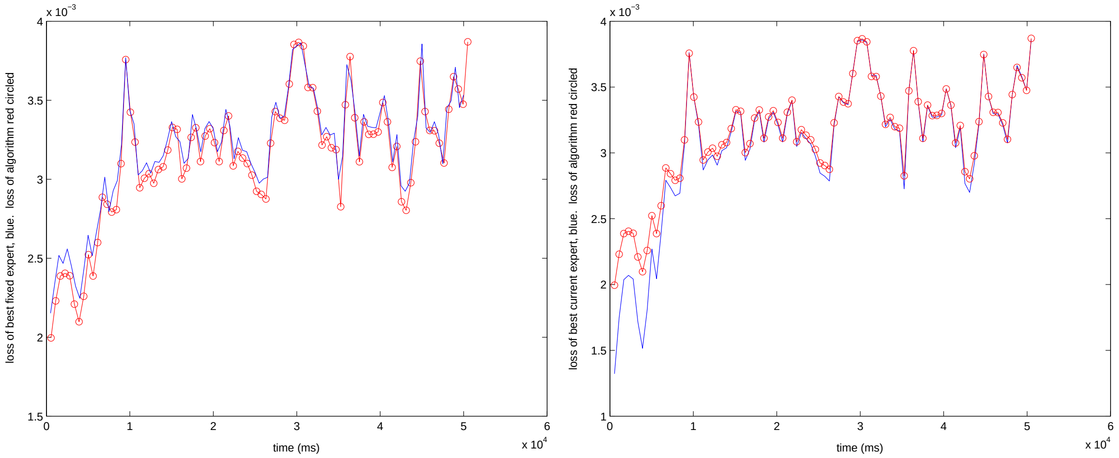


Figure 2-11: Competitive Analysis. Loss of the algorithm (circled) versus time. Solid is loss of the best fixed expert (left), and loss of the current best expert per training epoch (right).

convex combination of the experts, the algorithm can track the best continuous value of this parameter, in the discretized range.

2.5.5 Discussion

We proposed the use of online learning to manage the well-known energy/performance tradeoff in wireless networks, and applied `Learn-`α to the 802.11 wireless LAN PSM. The idea behind LPSM is to set up n "experts," each corresponding to a particular deterministic value of the sleep cycle, with a device sleeping for a duration equal to a weighted sum of these deterministic times. The weights of these n experts are updated according to a loss function, which penalizes the energy consumed and the slowdown introduced by sleeping for a certain amount of time, under the current network conditions. The method of updating the weighting depends upon the algorithm's model of the level of non-stationarity of the observed process: current network activity, in this case. Unlike previous online learning algorithms, `Learn-`α does not take this quantity as a parameter that must be set beforehand. Instead it learns this quantity online: the "switching-rate" of current network activity, simultaneous to learning the best current polling time.

Our experimental results, based on trace-driven simulation and trace-based analysis of Web client traces, show that for a Web-like request/response workload, LPSM (using the particular loss function we chose) can save 7%-20% more energy than 802.11 in power-saving mode, with an associated increase in average slowdown by a factor of at most 1.2.

Since the LPSM implementation we suggest is the same as 802.11 PSM, other than a polling time that changes adaptively, and remaining awake while the link is active, integration into 802.11 PSM would be relatively straightforward. In order to help synchronize between sleeping and awake nodes, LPSM chooses sleep durations that are rounded to the nearest 100 ms multiple of the time computed by Learn-α.

The complexity of the algorithm, can be $O(mn)$, or $O(m+n)$ if parallel computation is supported, since the α-expert updates can be run in parallel. In comparison to related work, this is slightly more complex, but note that n, the number of candidate polling times, can be chosen as a small constant, without degrading performance, as it just defines the level of discretization for a fixed range of possible sleep times. To obtain tight performance guarantees, the optimal number of α-experts is computed based on the timescale along which one would like to benchmark performance. However m can also be held constant, when computation is an issue, and our empirical results verified that this did not significantly degrade performance.

We note that the loss function we use in LPSM may be improved in the future, to obtain a different tradeoff between energy and latency. In our idealized model, we assume that bytes arrive uniformly during the sleep interval. If the protocol for delivering packets to a node were to maintain the times at which the packets originally arrived, the loss function could be made more accurate. The fact that LPSM did rather well despite this simplifying assumption, augurs well for its performance when finer-grained information on arrivals is available. Another extension to LPSM would be to estimate the energy that would be consumed in retrieving the buffered bytes, and factor that into the loss function as well. Note that the algorithm is modular, in that any objective function that is proportional to the energy/performance tradeoff may be used. The implementation methods mentioned above, which would not add too much complexity, should further improve the performance of LPSM over 802.11 PSM. We note that previous approaches to this problem did not save significant energy compared to 802.11 PSM, yet even our initial application of this algorithm was able to save substantial energy.

Finally, due to the nature of Learn-α's performance guarantees for learning shifting concepts, as discussed in Section 2.2.3, in the future we would be very interested to observe Learn-α's performance, as well as that of previous approaches, in simulations that are non-stationary.

Chapter 3

Learning with {mistake, label, error}-complexity guarantees

This chapter is based on joint work with Sanjoy Dasgupta and Adam Tauman Kalai, that originally appeared in [DKM05].

This chapter studies learning with online constraints, under the iid assumption, first in the supervised case and then in an active learning setting. We start by showing a lower bound on mistakes of $\Omega(\frac{1}{\epsilon^2})$ for the Perceptron algorithm to learn linear separators within generalization error ϵ, with respect to data distributed uniformly over the unit sphere in $\mathbb{R}^d$. We then present a modification of the Perceptron update and provide an upper bound of $\tilde{O}(d \log \frac{1}{\epsilon})$ on its mistake-complexity for this case. Our lower bound implies that in an active learning setting, using any active learning rule, the Perceptron algorithm needs $\Omega(\frac{1}{\epsilon^2})$ labels to learn linear separators within generalization error ϵ, with respect to this input distribution. We then present a simple selective sampling algorithm for this problem, which combines the modified Perceptron update with an adaptive filtering rule for deciding which points to query. For data distributed uniformly over the unit sphere in $\mathbb{R}^d$, we show that our algorithm reaches generalization error ϵ after asking for just $\tilde{O}(d \log \frac{1}{\epsilon})$ labels. It is non-trivial to match a mistake bound with a label bound since a label is required for every mistake, and random label choices may hit only a small fraction of mistakes. Additionally, we attain a matching bound for the total errors (labeled and unlabeled) made by the algorithm before reaching generalization error ϵ.

The exponential improvement in label-complexity that we provide over the usual sample complexity of supervised learning has previously been demonstrated only for algorithms that do not respect online constraints, for example the computationally more complex query-by-committee algorithm.

3.1 Introduction

In many machine learning applications, unlabeled data is abundant but labeling is expensive. This distinction is not captured in the standard PAC or online models of supervised learning, and has motivated the field of *active learning*, in which the labels of data points are initially hidden, and the learner must pay for each label it wishes revealed. If query points are chosen randomly, the number of labels needed to reach a target generalization error ϵ, at a target confidence level $1-\delta$, is similar to the sample complexity of supervised learning. The hope is that there are alternative querying strategies which require significantly fewer labels.

To date, the single most dramatic demonstration of the potential of active learning is perhaps Freund et al.'s analysis of the query-by-committee (QBC) learning algorithm [FSST97]. In their *selective sampling* model, the learner observes a stream of unlabeled data and makes spot decisions about whether or not to ask for a point's label. They show that if the data is drawn uniformly from the surface of the unit sphere in $\mathbb{R}^d$, and the hidden labels correspond perfectly to a homogeneous (i.e., through the origin) linear separator from this same distribution, then it is possible to achieve generalization error ϵ after seeing $\tilde{O}(\frac{d}{\epsilon}\log\frac{1}{\epsilon})$ points and requesting just $\tilde{O}(d\log\frac{1}{\epsilon})$ labels:[1] an exponential improvement over the usual $\tilde{\Theta}(\frac{d}{\epsilon})$ sample complexity of learning linear separators in a supervised setting [Lon95; Lon03].[2] This remarkable result is tempered somewhat by the complexity of the QBC algorithm, which involves random sampling from intermediate version spaces; the complexity of the update step scales (polynomially) with the number of updates performed, so it does not respect the online constraints on time and memory, of concern in this thesis.

In this chapter, we show how a simple modification of the Perceptron update can be used to achieve the same sample complexity bounds (within $\tilde{O}$ factors), under the same streaming model and the same uniform input distribution. Unlike QBC, we do not assume a distribution over target hypotheses, and our algorithm does not need to store previously seen data points, only its current hypothesis.

Our algorithm has the following structure.

```
Set initial hypothesis $v_0 \in \mathbb{R}^d$
For $t = 0, 1, 2, \ldots$
  Receive unlabeled point $x_t$
  Make a prediction SGN($v_t \cdot x_t$)
  *Filtering step:* Decide whether to ask for $x_t$'s label
  If label $y_t$ is requested:
    *Update step:* Set $v_{t+1}$ based on $v_t, x_t, y_t$
    Adjust filtering rule
  else:    $v_{t+1} = v_t$
```

Update step.

It turns out that the regular Perceptron update, originally introduced by [Ros58], that is,

$$\text{if } (x_t, y_t) \text{ is misclassified then } v_{t+1} = v_t + y_t x_t$$

[1] In this thesis, the $\tilde{O}$ notation is used to suppress terms in $\log d, \log\log\frac{1}{\epsilon}$ and $\log\frac{1}{\delta}$.

[2] This label-complexity can be seen to be optimal by counting the number of spherical caps of radius ϵ that can be packed onto the surface of the unit sphere in $\mathbb{R}^d$.

cannot yield an error rate better than $\Omega(1/\sqrt{l_t})$, where l_t is the number of labels queried up to time t, no matter what filtering scheme is used. In particular:

Theorem 3 *Consider any sequence of data points $x_0, x_1, x_2, \ldots$ which is perfectly classified by some linear separator $u \in \mathbb{R}^d$. If θ_t is the angle between u and v_t, then for any $t \geq 0$, if $\theta_{t+1} \leq \theta_t$ then $\sin\theta_t \geq 1/(5\sqrt{l_t + \|v_0\|^2})$.*

This holds regardless of how the data is produced. When the points are distributed uniformly over the unit sphere, $\theta_t \geq \sin\theta_t$ (for $\theta_t \leq \frac{\pi}{2}$) is proportional to the error rate of v_t, yielding a lower bound of $\Omega(\frac{1}{\epsilon^2})$ on the number of labels to reach error ϵ.

So instead we use a slightly modified update rule:

$$\text{if } (x_t, y_t) \text{ is misclassified then } v_{t+1} = v_t - 2(v_t \cdot x_t)x_t$$

(where x_t is assumed normalized to unit length). Note that the update can also be written as $v_{t+1} = v_t + 2y_t|v_t \cdot x_t|x_t$, since updates are only made on mistakes, in which case $y_t \neq \texttt{SGN}(v_t \cdot x_t)$, by definition. Thus we are scaling the standard Perceptron's additive update by a factor of $2|v_t \cdot x_t|$ to avoid oscillations caused by points close to the hyperplane represented by the current hypothesis. The same rule, but without the factor of two, has been used in previous work [BFKV96] on learning linear classifiers from noisy data, in a batch setting. We are able to show that our formulation has the following generalization performance in a supervised setting.

Theorem 4 *When the modified Perceptron algorithm is applied in a sequential supervised setting, with data points x_t drawn independently and uniformly at random from the surface of the unit sphere in $\mathbb{R}^d$, then with probability $1-\delta$, after $O(d(\log\frac{1}{\epsilon}+\log\frac{1}{\delta}))$ mistakes, its generalization error is at most ϵ.*

This contrasts favorably with the $\tilde{O}(\frac{d}{\epsilon^2})$ mistake bound of the Perceptron algorithm, and a more recent variant, on the same distribution [Bau97; Ser99]. As a lower bound for standard Perceptron, Theorem 3 also applies in the supervised case, as it holds for all filtering rules, including viewing all the labels. The bound on labels, $\Omega(\frac{1}{\epsilon^2})$, lower bounds mistakes as well, as we show in Section 3.4.

The PAC sample complexity of the problem under the uniform distribution is $\tilde{\Theta}(\frac{d}{\epsilon})$ (lower bound [Lon95], and upper bound [Lon03]). Yet since not all examples yield mistakes, mistake bounds can be lower than sample bounds. A similar statement holds in the active learning case: upper bounds on label-complexity can be lower than sample bounds, since the algorithms are allowed to filter which samples to label.

Filtering step.

Given the limited information the algorithm keeps, a natural filtering rule is to query points x_t when $|v_t \cdot x_t|$ is less than some threshold s_t. The choice of s_t is crucial. If it is too large, then only a miniscule fraction of the points queried will actually be misclassified – almost all labels will be wasted. On the other hand, if s_t is too small, then the waiting time for a query might be prohibitive, and when an update is actually made, the magnitude of this update might be tiny.

Therefore, we set the threshold adaptively: we start s high, and keep dividing it by two until we reach a level where there are enough misclassifications amongst the points queried. This filtering strategy makes possible our main theorem, for the active learning setting, again for data from the uniform distribution over the unit sphere in $\mathbb{R}^d$.

Theorem 5 *With probability $1 - \delta$, if the active modified Perceptron algorithm is given a stream of $\tilde{O}(\frac{d}{\epsilon} \log \frac{1}{\epsilon})$ unlabeled points, it will request $\tilde{O}(d \log \frac{1}{\epsilon})$ labels, make $\tilde{O}(d \log \frac{1}{\epsilon})$ errors (on all points, labeled or not), and have final error $\leq \epsilon$.*

3.2 Related work

Our approach relates to the literature on selective sampling, originally introduced by [CAL94]. Several selective sampling algorithms for learning linear separators (or their probabilistic analogues) have been proposed in the literature, and some have been shown to work in practice, for example Lewis and Gale's sequential algorithm for text classification [LG94], which has batch access to the remaining unlabeled data points at each iteration. Several of these are similar in spirit to our heuristic, in that they query points with small margins, such as Tong and Koller's [TK01] active learning algorithms that use a support vector machine (SVM) as the underlying classifier and work well in practice. These algorithms are not online, and to our knowledge they lack formal guarantees.

Among the active learning algorithms that have been shown to obey theoretical guarantees, several schemes with provable upper bounds on label-complexity are actually intractable. Dasgupta provided a general result in a non-Bayesian, realizable setting [Das05] for a scheme that requires exponential storage and computation. In a non-Bayesian, agnostic setting, a recent label-complexity upper bound for learning linear separators under the uniform input distribution, relies on an scheme that is not computationally efficient [BBL06].

Several formal guarantees have been shown for active learning algorithms that can actually be implemented. Under Bayesian assumptions, Freund et al. [FSST97] gave an upper bound on label-complexity for learning linear separators under the uniform, using Query By Committee [SOS92], a computationally complex algorithm that has recently been simplified to yield encouraging empirical results [GBNT05]. Cesa-Bianchi et al. provided regret bounds on a selective sampling algorithm for learning linear thresholds [CBCG03] from a stream of iid examples corrupted by random class noise whose rate scales with the examples' margins. Recently an upper bound on label-complexity of, $\tilde{O}(d^2 \log \frac{1}{\epsilon})$, which is close to ours for small d, has been shown by [BBL06] for an algorithm due to [CAL94], in the setting we consider. The algorithms discussed above do not respect online constraints on time and memory however.

In terms of algorithms for selective sampling that adhere to both of the online constraints of concern in this thesis, and that have been analyzed formally, we are primarily aware of work by Cesa-Bianchi, Gentile and Zaniboni (CBGZ) [CBGZ04].

Their algorithm conforms to roughly the same template as ours but differs in both the update and filtering rule – it uses the regular Perceptron update and it queries points x_t according to a fixed, randomized rule which favors small $|v_t \cdot x_t|$. The authors make no distributional assumptions on the input and they show that in terms of worst-case hinge-loss bounds, their algorithm does about as well as one which queries *all* labels. The actual fraction of points queried varies from data set to data set. In contrast, our objective is to achieve a target generalization error with minimum label-complexity, although we do also obtain a mistake bound and a bound on total errors (both labeled and unlabeled) under our distributional assumption.

It is known that active learning does not always give a large improvement in the sample complexity of learning linear separators. For instance, in our setting where data is distributed uniformly over the unit sphere, recent work has shown that if the target linear separator is allowed to be non-homogeneous, then the number of labels required to reach error ϵ is $\Omega(1/\epsilon)$, no matter what active learning scheme is used [Das04]. This lower bound also applies to learning homogeneous linear separators with respect to an arbitrary distribution. In the fully agnostic setting, Kääriäinen provided a lower bound of $\Omega(\frac{\eta^2}{\epsilon^2})$, where η is the error rate of the best hypothesis in the concept class [Kää06].

Finally, there is a rich body of theory on a related model in which it is permissible to create query points synthetically; a recent survey by Angluin [Ang01] summarizes key results.

3.3 Preliminaries

In our model, all data x_t lie on the surface of the unit ball in $\mathbb{R}^d$, which we will denote as S:

$$S = \left\{ x \in \mathbb{R}^d \;\middle|\; \|x\| = 1 \right\}.$$

Their labels y_t are either -1 or $+1$, and the target function is a half-space $u \cdot x \geq 0$ represented by a unit vector $u \in \mathbb{R}^d$ which classifies all points perfectly, that is, $y_t(u \cdot x_t) > 0$ for all t, with probability one. This is called the *realizable setting*: the target classifier is in the concept class, i.e. there exists a half-space (u) which can correctly classify all the data.

For any vector $v \in \mathbb{R}^d$, we define $\hat{v} = \frac{v}{\|v\|}$ to be the corresponding unit vector.

Our lower bound (Theorem 3) holds regardless of how the data are generated; thereafter we will assume that the data points x_t are drawn independently from the uniform distribution over S. Under the uniform input distribution, any hypothesis $v \in \mathbb{R}^d$ has error

$$\epsilon(v) = P_{x \in S}[\texttt{SGN}(v \cdot x) \neq \texttt{SGN}(u \cdot x)] = \frac{\arccos(u \cdot \hat{v})}{\pi}.$$

We will refer to the error rate of a hypothesis v as its *generalization error*, since in the realizable case the target itself has zero error.

For a hypothesis v_t, we will denote the angle between u and v_t by θ_t, and we

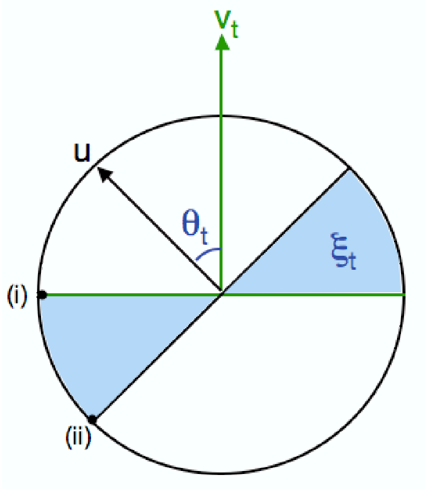


Figure 3-1: The projection of the error region ξ_t onto the plane defined by u and v_t.

will define the error region of v_t as $\xi_t = \{x \in S \,|\, \mathtt{SGN}(v_t \cdot x) \neq \mathtt{SGN}(u \cdot x)\,\}$. Figure 3-1 provides a schematic of the projection of the error region onto the plane defined by u and v_t.

We will use the term *margin*, in the context of learning half-spaces, to denote simply the distance from an example to the separator in question, as opposed to the standard use of this term (as the minimum over examples of this distance with respect to the target separator). For example, we will denote the margin of x with respect to v as $|x \cdot v|$.

We will use a few useful inequalities for θ on the interval $(0, \frac{\pi}{2}]$.

$$\frac{4}{\pi^2} \leq \frac{1-\cos\theta}{\theta^2} \quad \leq \quad \frac{1}{2}, \tag{3.1}$$

$$\frac{2}{\pi}\theta \leq \sin\theta \quad \leq \quad \theta \tag{3.2}$$

Equation (3.1) can be verified by checking that for θ in this interval, $\frac{1-\cos\theta}{\theta^2}$ is a decreasing function, and evaluating it at the endpoints.

We will also make use of the following lemma.

Lemma 2 *For any fixed unit vector a and any $\gamma \leq 1$,*

$$\frac{\gamma}{4} \leq P_{x \in S}\left[|a \cdot x| \leq \frac{\gamma}{\sqrt{d}}\right] \leq \gamma \tag{3.3}$$

The proof is deferred to the appendix.

3.4 A lower bound on {mistakes, labels} for the Perceptron update

Here we prove Theorem 3 simultaneously for the supervised setting, in which we lower bound mistakes for Perceptron, and the active learning setting, in which we lower bound labels for Perceptron paired with any active learning rule. We prove a lower bound on updates, which implies a mistake bound for Perceptron since it only performs updates on mistakes. This also serves as a lower bound on labels in the active learning setting, since a label is required in order to identify a mistake, i.e. as explained in Section 1.3.2, mistakes are errors that have been labeled.

Consider standard Perceptron, an algorithm of the following form:

```
Pick some $v_0 \in \mathbb{R}^d$
Repeat for $t = 0, 1, 2, \ldots$:
  Get some $(x, y)$ for which $y(v_t \cdot x) \le 0$
  $v_{t+1} = v_t + yx$
```

On any update,

$$v_{t+1} \cdot u = v_t \cdot u + y(x \cdot u). \tag{3.4}$$

Thus, if we assume for simplicity that $v_0 \cdot u \ge 0$ (we can always just start count when this first occurs) then $v_t \cdot u \ge 0$ always, and the angle between u and v_t is always acute. Since we denote this angle by θ_t, we can write,

$$\|v_t\| \cos\theta_t = v_t \cdot u.$$

The update rule also implies

$$\|v_{t+1}\|^2 = \|v_t\|^2 + 1 + 2y(v_t \cdot x). \tag{3.5}$$

Thus $\|v_t\|^2 \le t + \|v_0\|^2$ for all t. In particular, this means that Theorem 3 is an immediate consequence of the following lemma.

Lemma 3 *Assume $v_0 \cdot u \ge 0$ (i.e., start count when this first occurs). Then*

$$\theta_{t+1} \le \theta_t \quad \Rightarrow \quad \sin\theta_t \ge \min\left\{\frac{1}{3}, \frac{1}{5\|v_t\|}\right\}.$$

Proof: Figure 3-1 shows the unit circle in the plane defined by u and v_t. The dot product of any point $x \in \mathbb{R}^d$ with either u or v_t depends only upon the projection of x into this plane. The point is misclassified when its projection lies in the shaded region, ξ_t. For such points, $y(u \cdot x)$ is at most $\sin\theta_t$ (point (i)) and $y(v_t \cdot x)$ is at least $-\|v_t\| \sin\theta_t$ (point (ii)).

Combining this with equations (3.4) and (3.5), we get

$$\begin{aligned} v_{t+1} \cdot u &\le v_t \cdot u + \sin\theta_t \\ \|v_{t+1}\|^2 &\ge \|v_t\|^2 + 1 - 2\|v_t\| \sin\theta_t \end{aligned}$$

To establish the lemma, we first assume $\theta_{t+1} \leq \theta_t$ and $\sin\theta_t \leq \frac{1}{5\|v_t\|}$, and then conclude that $\sin\theta_t \geq \frac{1}{3}$.

$\theta_{t+1} \leq \theta_t$ implies

$$\cos^2\theta_t \ \leq\ \cos^2\theta_{t+1} \ =\ \frac{(u \cdot v_{t+1})^2}{\|v_{t+1}\|^2} \ \leq\ \frac{(u \cdot v_t + \sin\theta_t)^2}{\|v_t\|^2 + 1 - 2\|v_t\| \sin\theta_t}.$$

The final denominator is positive since $\sin\theta_t \leq \frac{1}{5\|v_t\|}$. Rearranging,

$$(\|v_t\|^2 + 1 - 2\|v_t\| \sin\theta_t) \cos^2\theta_t \ \leq\ (u \cdot v_t)^2 + \sin^2\theta_t + 2(u \cdot v_t)\sin\theta_t$$

and using $\|v_t\| \cos\theta_t = (u \cdot v_t)$:

$$(1 - 2\|v_t\| \sin\theta_t) \cos^2\theta_t \ \leq\ \sin^2\theta_t + 2\|v_t\| \sin\theta_t \cos\theta_t$$

Again, since $\sin\theta_t \leq \frac{1}{5\|v_t\|}$, it follows that $(1 - 2\|v_t\| \sin\theta_t) \geq \frac{3}{5}$ and that $2\|v_t\| \sin\theta_t \cos\theta_t \leq \frac{2}{5}$. Using $\cos^2 = 1 - \sin^2$, we then get

$$\frac{3}{5}(1 - \sin^2\theta_t) \ \leq\ \sin^2\theta_t + \frac{2}{5}$$

which works out to $\sin^2\theta_t \geq \frac{1}{8}$, implying $\sin\theta_t > \frac{1}{3}$. □

The problem is that the Perceptron update can be too large. In $\mathbb{R}^2$ (e.g. Figure 3-1), when θ_t is tiny, the update will cause v_{t+1} to overshoot the mark and swing too far to the other side of u, unless $\|v_t\|$ is very large: to be precise, we need $\|v_t\| = \Omega(1/\sin\theta_t)$. But $\|v_t\|$ grows slowly, at best at a rate of $\sqrt{t}$. If $\sin\theta_t$ is proportional to the error of v_t, as in the case of data distributed uniformly over the unit sphere, this means that the Perceptron update cannot stably maintain an error rate $\leq \epsilon$ until $t = \Omega(1/\epsilon^2)$.

3.5 A modified Perceptron update

We now describe the modified Perceptron algorithm. Using a simple modification to the standard Perceptron update yields the fast convergence we will prove subsequently. Unlike with standard Perceptron, this modification ensures that $v_t \cdot u$ is increasing, i.e., the error of v_t is monotonically decreasing. Another difference from the standard update (and other versions) is that the magnitude of $\|v_t\| = 1$, which is convenient for our analysis.

The modified Perceptron algorithm is shown in Figure 3-2. We now show that the norm of v_t stays at one. Note that $\|v_1\| = 1$ and

$$\|v_{t+1}\|^2 = \|v_t\|^2 + 4(v_t \cdot x_t)^2 \|x_t\|^2 - 4(v_t \cdot x_t)^2 = 1$$

by induction. In contrast, for the standard Perceptron update, the magnitude of v_t

```
Inputs:  dimensionality $d$ and desired number of updates
(mistakes) $M$.
  Let $v_1 = x_1 y_1$ for the first example $(x_1, y_1)$.
  For $t = 1$ to $M$:
    Let $(x_t, y_t)$ be the next example with $y(x \cdot v_t) < 0$.
    $v_{t+1} = v_t - 2(v_t \cdot x_t)x_t$.
```

Figure 3-2: The modified Perceptron algorithm. The standard Perceptron update, $v_{t+1} = v_t + y_t x_t$, is in the same direction (note $y_t = -\texttt{SGN}(v_t \cdot x_t)$) but different magnitude (scaled by a factor of $2|v_t \cdot x_t|$).

is important and normalized vectors cannot be used.

With the modified update, the error can only decrease, because $v_t \cdot u$ only increases:

$$v_{t+1} \cdot u = v_t \cdot u - 2(v_t \cdot x_t)(x_t \cdot u) = v_t \cdot u + 2|v_t \cdot x_t||x_t \cdot u|.$$

The second equality follows from the fact that v_t misclassified x_t. Thus $v_t \cdot u$ is increasing, and the increase can be bounded from below by showing that $|v_t \cdot x_t||x_t \cdot u|$ is large. This is a different approach from previous analyses.

Blum et al. [BFKV96] used an update similar to ours, but without the factor of two. In general, one can consider modified updates of the form $v_{t+1} = v_t - \alpha(v_t \cdot x_t)x_t$. When $\alpha \neq 2$, the vectors v_t no longer remain of fixed length; however, one can verify that their corresponding unit vectors $\hat{v}_t$ satisfy

$$\hat{v}_{t+1} \cdot u = (\hat{v}_t \cdot u + \alpha|\hat{v}_t \cdot x_t||x_t \cdot u|)/\sqrt{1 - \alpha(2-\alpha)(\hat{v}_t \cdot x_t)^2},$$

and thus any choice of $\alpha \in [0, 2]$ guarantees non-increasing error. Blum et al. used $\alpha = 1$ to guarantee progress in the denominator (their analysis did not rely on progress in the numerator) as long as $\hat{v}_t \cdot u$ and $(\hat{v}_t \cdot x_t)^2$ were bounded away from 0. Their approach was used in a batch setting as one piece of a more complex algorithm for noise-tolerant learning. In our sequential framework, we can bound $|\hat{v}_t \cdot x_t||x_t \cdot u|$ away from 0 in expectation, under the uniform distribution, and hence the choice of $\alpha = 2$ is most convenient, but $\alpha = 1$ would work as well. Although we do not further optimize our choice of the constant α, this choice itself may yield interesting future work, perhaps by allowing it to be a function of the dimension. The identical update to ours was previously proposed for learning Linear Theshold Units: hyperplanes over discrete-valued vector inputs, in [HK99] which tests the algorithm empirically, and discusses mistake bounds in a different analysis context.

3.5.1 An upper bound on mistakes for the modified Perceptron

How large do we expect $|v_t \cdot x_t|$ and $|u \cdot x_t|$ to be for an error (x_t, y_t)? As we shall see, in d dimensions, one expects each of these terms to be on the order of $d^{-1/2} \sin \theta_t$, where $\sin\theta_t = \sqrt{1-(v_t\cdot u)^2}$. Hence, we might expect their product to be about $(1-(v_t\cdot u)^2)/d$, which is how we prove the following lemma.

Note, we have made little effort to optimize constant factors.

Lemma 4 *For any v_t, with probability at least $\frac{1}{3}$,*

$$1 - v_{t+1}\cdot u \le (1 - v_t\cdot u)\left(1 - \frac{1}{50d}\right).$$

There exists a constant $c > 0$, such that with probability at least $\frac{63}{64}$, for any v_t,

$$1 - v_{t+1}\cdot u \le (1 - v_t\cdot u)\left(1 - \frac{c}{d}\right).$$

Proof: We show only the first part of the lemma. The second part is quite similar. We will argue that each of $|v_t\cdot x_t|$,$|u\cdot x_t|$ is "small" with probability at most 1/3. This means, by the union bound, that with probability at least 1/3, they are both sufficiently large.

As explained in Section 3.3, the error rate of v_t is θ_t/π, where $\cos\theta_t = v_t\cdot u$, and the error region is denoted as $\xi_t = \{x\in S\,|\texttt{SGN}(v_t\cdot x)\neq\texttt{SGN}(u\cdot x)\}$. By Lemma 2, for an x drawn uniformly from the sphere,

$$P_{x\in S}\left[|v_t\cdot x| \le \frac{\theta_t}{3\pi\sqrt{d}}\right] \le \frac{\theta_t}{3\pi}.$$

Using $P[A|B]\le P[A]/P[B]$, we have,

$$P_{x\in S}\left[|v_t\cdot x| \le \frac{\theta_t}{3\pi\sqrt{d}} \;\middle|\; x\in\xi_t\right] \le \frac{P_{x\in S}[|v_t\cdot x|\le\frac{\theta_t}{3\pi\sqrt{d}}]}{P_{x\in S}[x\in\xi_t]} \le \frac{\theta_t/(3\pi)}{\theta_t/\pi} = \frac{1}{3}$$

Similarly for $|u\cdot x|$, and by the union bound the probability that $x\in\xi_t$ is within margin $\frac{\theta}{3\pi\sqrt{d}}$ from either u or v_t is at most $\frac{2}{3}$. Since the updates only occur if x is in the error region, we now have a lower bound on the expected magnitude of $|v_t\cdot x||u\cdot x|$.

$$P_{x\in S}\left[|v_t\cdot x||u\cdot x| \ge \frac{\theta_t^2}{(3\pi\sqrt{d})^2}\middle| x\in\xi_t\right] \ge \frac{1}{3}.$$

Hence, we know that with probability at least 1/3, $|v_t\cdot x||u\cdot x| \ge \frac{1-(v_t\cdot u)^2}{100d}$, since $\theta_t^2 \ge \sin^2\theta_t = 1-(v_t\cdot u)^2$ and $(3\pi)^2 < 100$. In this case,

$$\begin{aligned} 1 - v_{t+1}\cdot u &\le 1 - v_t\cdot u - 2|v_t\cdot x_t||u\cdot x_t| \\ &\le 1 - v_t\cdot u - \frac{1-(v_t\cdot u)^2}{50d} \\ &\le (1 - v_t\cdot u)\left(1 - \frac{1+v_t\cdot u}{50d}\right) \end{aligned}$$

□

Finally, we give a high-probability bound, i.e. Theorem 4, stated here with proof.

Theorem 4 *With probability* $1-\delta$*, after* $M = O(d(\log \frac{1}{\epsilon} + \log \frac{1}{\delta}))$ *mistakes, the generalization error of the modified Perceptron algorithm is at most* ϵ*.*

Proof: By the above lemma, we can conclude that, for any vector v_t,

$$E_{x_t \in \xi_t}[1 - v_{t+1} \cdot u] \leq (1 - v_t \cdot u)\left(1 - \frac{1}{3(50d)}\right).$$

This is because with $\geq 1/3$ probability it goes down by a factor of $1 - \frac{1}{50d}$ and with the remaining $\leq 2/3$ probability it does not increase. Hence, after M mistakes,

$$E[1 - v_M \cdot u] \leq (1 - v_1 \cdot u)\left(1 - \frac{1}{150d}\right)^M \leq \left(1 - \frac{1}{150d}\right)^M,$$

since $v_1 \cdot u \geq 0$. By Markov's inequality,

$$P\left[1 - v_M \cdot u \geq \left(1 - \frac{1}{150d}\right)^M \delta^{-1}\right] \leq \delta.$$

Finally, using (3.1) and $\cos\theta_M = v_M \cdot u$, we see $P[\frac{4}{\pi^2}\theta_M^2 \geq (1 - \frac{1}{150d})^M \delta^{-1}] \leq \delta$. Using $M = 150d \log(1/\epsilon\delta)$ gives $P[\frac{\theta_M}{\pi} \geq \epsilon] \leq \delta$ as required. □

The additional factor of $\frac{1}{\epsilon}$ in the bound on unlabeled samples ($\tilde{O}(\frac{d}{\epsilon} \log \frac{1}{\epsilon})$) follows by upper bounding the number of unlabeled samples until an update: when the hypothesis has error rate ϵ, the waiting time (in samples) until an update is $\frac{1}{\epsilon}$, in expectation.

3.6 An active modified Perceptron and {label, error} upper bounds

The ideal objective in designing an active learning rule that minimizes label-complexity would be to query for labels only on points in the error region, ξ_t. However without knowledge of u, the algorithm is unaware of the location of ξ_t. The intuition behind our active learning rule is to approximate the error region, given the information the algorithm does have: v_t. As shown in Figure 3-3, the labeling region $\mathbb{L}$ is simply formed by thresholding the margin of a candidate example with respect to v_t.

The active version of the modified Perceptron algorithm is shown in Figure 3-4. The algorithm is similar to the algorithm of the previous section, in its update step. For its filtering rule, we maintain a threshold s_t and we only ask for labels of examples with $|v_t \cdot x| \leq s_t$. Approximating the error region is achieved by choosing the threshold, s_t, adaptively, so as to manage the tradeoff between $\mathbb{L}$ being too large, causing many labels to be wasted without hitting ξ_t (and thus yielding updates), and $\mathbb{L}$ only containing points with very small margins with respect to v_t, since our

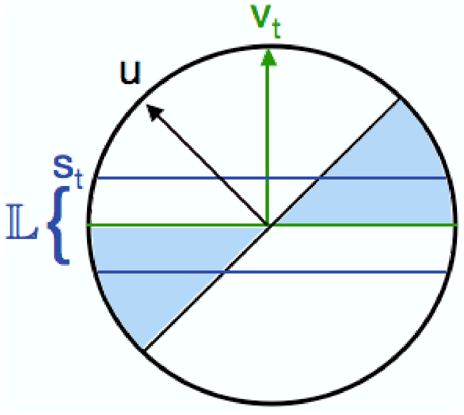


Figure 3-3: The active learning rule is to query for labels on points x in $\mathbb{L}$ which is defined by the threshold s_t on $|v_t \cdot x|$.

update step will make very small updates on such points. We decrease the threshold adaptively over time, starting at $s_1 = 1/\sqrt{d}$ and reducing it by a factor of two whenever we have a run of labeled examples on which we are correct.

For Theorem 5, we select values of R, L that yield ϵ error with probability at least $1 - \delta$. The idea of the analysis is as follows:

Definition 1 *We say the* t*th update is "good" if,*

$$1 - v_{t+1} \cdot u \leq (1 - v_t \cdot u)\left(1 - \frac{c}{d}\right).$$

(The constant c *is from Lemma 4.)*

1. (Lemma 5) First, we argue that s_t is not too small (we do not decrease s_t too quickly). Assuming this is the case, then 2 and 3 hold.

2. (Lemma 7) We query for labels on at least an expected $1/32$ of all *errors*. In other words, some errors may go undetected because we do not ask for their labels, but the number of mistakes total should not be much more than 32 times the number of updates we actually perform.

3. (Lemma 8) Each update is *good* (Definition 1) with probability at least $1/2$.

4. (Theorem 5) Finally, we conclude that we cannot have too many label queries, updates, or total errors, because half of our updates are good, $1/32$ of our errors are updates, and about $1/R$ of our labels are updates.

We first lower-bound s_t with respect to our error, showing that, with high probability, the threshold s_t is never too small.

```
Inputs:  Dimensionality $d$, maximum number of labels $L$,
and patience $R$.
  $v_1 = x_1 y_1$ for the first example $(x_1, y_1)$.
  $s_1 = 1/\sqrt{d}$
  For $t = 1$ to $L$:
    Wait for the next example $x$ :  $|x \cdot v_t| \leq s_t$ and query its label.
    Call this labeled example $(x_t, y_t)$.
    If $(x_t \cdot v_t) y_t < 0$, then:
      $v_{t+1} = v_t - 2(v_t \cdot x_t) x_t$
      $s_{t+1} = s_t$
    else:
      $v_{t+1} = v_t$
      If predictions were correct on $R$ consecutive labeled
      examples (i.e.  $(x_i \cdot v_i) y_i \geq 0 \ \forall i \in \{t - R + 1, t - R + 2, \ldots, t\}$),
      then set $s_{t+1} = s_t/2$, else $s_{t+1} = s_t$.
```

Figure 3-4: An active version of the modified Perceptron algorithm.

Lemma 5 *With probability at least* $1 - L\left(\frac{3}{4}\right)^R$, *we have:*

$$s_t \geq \sqrt{\frac{1 - (u \cdot v_t)^2}{16d}} \text{ for } t = 1, 2, \ldots, L, \text{ simultaneously.} \tag{3.6}$$

Before proving this lemma, it will be helpful to show the following lemma. As before, let us define $\xi_t = \{x \in S | (x \cdot v_t)(x \cdot u) < 0\}$.

Lemma 6 *For any* $\gamma \in \left(0, \sqrt{\frac{1-(u \cdot v_t)^2}{4d}}\right]$,

$$P_{x_t \in S}\left[x_t \in \xi_t \;\middle|\; |x_t \cdot v_t| < \gamma\right] \geq \frac{1}{4}$$

Proof: Let x be a random example from S such that $|x \cdot v_t| < \gamma$ and, without loss of generality, suppose that $0 \leq x \cdot v_t \leq \gamma$. Then we want to calculate the probability we err, i.e. $u \cdot x < 0$. We can decompose $x = x' + (x \cdot v_t) v_t$ where $x' = x - (x \cdot v_t) v_t$ is the component of x orthogonal to v_t, i.e. $x' \cdot v_t = 0$. Similarly for $u' = u - (u \cdot v_t) v_t$. Hence,

$$u \cdot x = (u' + (u \cdot v_t) v_t) \cdot (x' + (x \cdot v_t) v_t) = u' \cdot x' + (u \cdot v_t)(x \cdot v_t)$$

In other words, we err iff $u' \cdot x' < -(u \cdot v_t)(x \cdot v_t)$. Using $u \cdot v_t \in [0, 1]$ and since $x \cdot v_t \in [0, \sqrt{(1 - (u \cdot v_t)^2)/(4d)}]$, we conclude that if,

$$u' \cdot x' < -\sqrt{\frac{1 - (u \cdot v_t)^2}{4d}} \tag{3.7}$$

then we must err. Also, let $\hat{x}' = \frac{x'}{\|x'\|}$ be the unit vector in the direction of x'. It is straightforward to check that $\|x'\| = \sqrt{1-(x \cdot v_t)^2}$. Similarly, for u we define $\hat{u}' = \frac{u'}{\sqrt{1-(u \cdot v_t)^2}}$. Substituting these into (3.7), we must err if, $\hat{u}' \cdot \hat{x}' < -1/\sqrt{4d(1-(x \cdot v_t))^2}$, and since $\sqrt{1-(x \cdot v_t)^2} \geq \sqrt{1-1/(4d)}$, it suffices to show that,

$$P_{x \in S}\left[\hat{u}' \cdot \hat{x}' < \frac{-1}{\sqrt{4d(1-1/(4d))}} \,\middle|\, 0 \leq x \cdot v_t \leq \gamma\right] \geq \frac{1}{4}$$

What is the probability that this happens? Well, one way to pick $x \in S$ would be to first pick $x \cdot v_t$ and then to pick $\hat{x}'$ uniformly at random from the set $S' = \{\hat{x}' \in S | \hat{x}' \cdot v_t = 0\}$, which is a unit sphere in one fewer dimensions. Hence the above probability does not depend on the conditioning. By Lemma 2, for any unit vector $a \in S'$, the probability that $|\hat{u}' \cdot a| \leq 1/\sqrt{4(d-1)}$ is at most 1/2, so with probability at least 1/4 (since the distribution is symmetric), the signed quantity $\hat{u}' \cdot \hat{x}' < -1/\sqrt{4(d-1)} < -1/\sqrt{4d(1-1/(4d))}$. □

We are now ready to prove Lemma 5.

Proof [of Lemma 5]: Suppose that condition (3.6) fails to hold for some t's. Let t be the smallest number such that (3.6) fails. By our choice of s_1, clearly $t > 1$. Moreover, since t is the smallest such number, and $u \cdot v_t$ is increasing, it must be the case that $s_t = s_{t-1}/2$, that is we just saw a run of R labeled examples (x_i, y_i), for $i = t-R, \ldots, t-1$, with no mistakes, $v_i = v_t$, and

$$s_i = 2s_t < \sqrt{\frac{1-(u \cdot v_t)^2}{4d}} = \sqrt{\frac{1-(u \cdot v_i)^2}{4d}}. \tag{3.8}$$

Such an event is highly unlikely, however, for any t. In particular, from Lemma 6, we know that the probability of (3.8) holding for any particular i and the algorithm not erring is at most 3/4. Thus the chance of having any such run of length R is at most $L(3/4)^R$. □

Lemma 6 also tells us something interesting about the fraction of errors that we are missing because we do not ask for labels. In particular,

Lemma 7 *Given that $s_t \geq \sqrt{(1-(u \cdot v_t)^2)/(16d)}$, upon the tth update, each erroneous example is queried with probability at least 1/32, i.e.,*

$$P_{x \in S}\left[|x \cdot v_t| \leq s_t \,\middle|\, x \in \xi_t\right] \geq \frac{1}{32}.$$

Proof: Using Lemmas 6 and 2, we have

$$\begin{aligned} P_{x\in S}\left[x\in\xi_t \wedge |x\cdot v_t|\le s_t\right] &\ge P_{x\in S}\left[x\in\xi_t\wedge|x\cdot v_t|\le\sqrt{\frac{1-(u\cdot v_t)^2}{16d}}\right] \\ &\ge \frac{1}{4}P_{x\in S}\left[|x\cdot v_t|\le\sqrt{\frac{1-(u\cdot v_t)^2}{16d}}\right] \\ &\ge \frac{1}{64}\sqrt{1-(u\cdot v_t)^2}=\frac{1}{64}\sin\theta_t \\ &\ge \frac{\theta_t}{32\pi} \end{aligned}$$

For the last inequality, we have used (3.2). However, $P_{x\in S}[x\in\xi_t]=\theta_t/\pi$, so we are querying an error $x\in\xi_t$ with probability at least $1/32$, i.e., the above inequality implies,

$$P_{x\in S}\left[|x\cdot v_t|\le s_t \;\middle|\; x\in\xi_t\right]=\frac{P_{x\in S}\left[x\in\xi_t\wedge|x\cdot v_t|\le s_t\right]}{P_{x\in S}[x\in\xi_t]}\ge\frac{\theta_t/(32\pi)}{\theta_t/\pi}=\frac{1}{32}.$$

□

Next, we show that the updates are likely to make progress.

Lemma 8 *Assuming that* $s_t\ge\sqrt{(1-(u\cdot v_t)^2)/(16d)}$*, a random update is good with probability at least* $1/2$*, i.e.,*

$$P_{x_t\in S}\left[(1-v_{t+1}\cdot u)\le(1-v_t\cdot u)\left(1-\frac{c}{d}\right)\;\middle|\;|x\cdot v_t|\le s_t\wedge x_t\in\xi_t\right]\ge\frac{1}{2}.$$

Proof: By Lemma 7, each error is queried with probability 1/32. On the other hand, by Lemma 4 of the previous section, 63/64 of all errors are good. Since we are querying at least 2/64 fraction of all errors, at least half of our queried errors must be good. □

We now have the pieces to guarantee the convergence rate of the active algorithm, thereby proving Theorem 5. This involves bounding both the number of labels that we query as well as the number of total errors, which includes updates as well as errors that were never detected.

Theorem 5 *With probability* $1-\delta$*, using* $L=O\big(d\log\big(\frac{1}{\epsilon\delta}\big)(\log\frac{d}{\delta}+\log\log\frac{1}{\epsilon})\big)$ *labels and making a total number of errors of* $O\big(d\log\big(\frac{1}{\epsilon\delta}\big)(\log\frac{d}{\delta}+\log\log\frac{1}{\epsilon})\big)$*, the final error of the active modified Perceptron algorithm will be* ϵ*, when run with the above* L *and* $R=O(\log\frac{d}{\delta}+\log\log\frac{1}{\epsilon})$.

Proof: Let U be the number of updates performed. We know, by Lemma 5 that with probability $1-L(\frac{3}{4})^R$,

$$s_t\ge\frac{\sin\theta_t}{4\sqrt{d}}\ge\frac{\theta_t}{2\pi\sqrt{d}} \tag{3.9}$$

for all t. Again, we have used (3.2). By Lemma 8, we know that for each t which is an update, either (3.9) fails or

$$E[1 - u \cdot v_{t+1} | v_t] \leq (1 - u \cdot v_t) \left(1 - \frac{c}{2d}\right).$$

Hence, after U updates, using Markov's inequality,

$$P\left[1 - u \cdot v_L \geq \frac{4}{\delta}\left(1 - \frac{c}{2d}\right)^U\right] \leq \frac{\delta}{4} + L\left(\frac{3}{4}\right)^R.$$

In other words, with probability $1 - \delta/4 - L(3/4)^R$, we also have

$$U \leq \frac{2d}{c} \log \frac{4}{\delta(1 - u \cdot v_L)} \leq \frac{2d}{c} \log \frac{\pi^2}{\delta \theta_L^2} = O\left(d \log \frac{1}{\delta \epsilon}\right),$$

where for the last inequality we used (3.1). In total, $L \leq R\,(U + \log_2 1/s_L)$. This is because once every R labels we either have at least one update or we decrease s_L by a factor of 2. Equivalently, $s_L \leq 2^{U - L/R}$. Hence, with probability $1 - \delta/4 - L(3/4)^R$,

$$\frac{\theta_L}{2\pi\sqrt{d}} \leq s_L \leq 2^{O(d \log(1/\delta\epsilon)) - L/R}$$

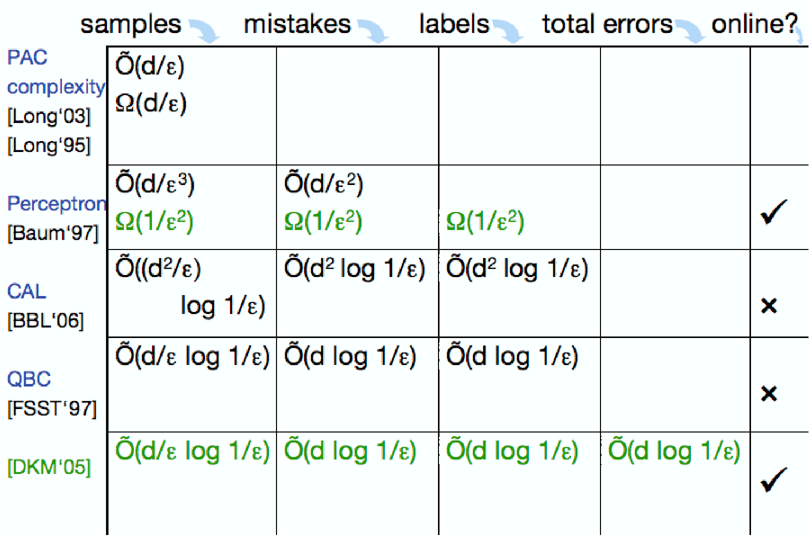
Working backwards, we choose $L/R = \Theta(d \log \frac{1}{\epsilon\delta})$ so that the above expression implies $\frac{\theta_L}{\pi} \leq \epsilon$, as required. We choose,

$$R = 10 \log \frac{2L}{\delta R} = \Theta\left(\log \frac{d \log \frac{1}{\epsilon\delta}}{\delta}\right) = O\left(\log \frac{d}{\delta} + \log\log \frac{1}{\epsilon}\right).$$

The first equality ensures that $L(3/4)^R \leq \delta/4$. Hence, for the L and R chosen in the theorem, with probability $1 - \frac{3}{4}\delta$, we have error $\theta_L/\pi < \epsilon$. Finally, either condition (3.9) fails or each error is queried with probability at least 1/32. By the multiplicative Chernoff bound, if there were a total of $E > 64U$ errors, then with probability $\geq 1 - \delta/4$, at least $E/64 > U$ would have been caught and used as updates. Hence, with probability at most $1 - \delta$, we have achieved the target error using the specified number of labels and incurring the specified number of errors. □

3.7 Conclusions and open problems

Table 3.1 details the related work discussed above, with the results of this chapter summarized in green. Algorithm names are stated in blue (CAL denotes [CAL94]), with their analyses cited in black. Only the Perceptron and the algorithm introduced in this chapter conform to the online constraints of interest in this thesis. Significant reductions in {mistake, label}-complexity are evident between Perceptron and our algorithm, in the supervised and active cases.



	samples	mistakes	labels	total errors	online?
PAC complexity [Long'03] [Long'95]	$\tilde{O}(d/\varepsilon)$ $\Omega(d/\varepsilon)$				
Perceptron [Baum'97]	$\tilde{O}(d/\varepsilon^3)$ $\Omega(1/\varepsilon^2)$	$\tilde{O}(d/\varepsilon^2)$ $\Omega(1/\varepsilon^2)$	$\Omega(1/\varepsilon^2)$		✓
CAL [BBL'06]	$\tilde{O}((d^2/\varepsilon) \log 1/\varepsilon)$	$\tilde{O}(d^2 \log 1/\varepsilon)$	$\tilde{O}(d^2 \log 1/\varepsilon)$		×
QBC [FSST'97]	$\tilde{O}(d/\varepsilon \log 1/\varepsilon)$	$\tilde{O}(d \log 1/\varepsilon)$	$\tilde{O}(d \log 1/\varepsilon)$		×
[DKM'05]	$\tilde{O}(d/\varepsilon \log 1/\varepsilon)$	$\tilde{O}(d \log 1/\varepsilon)$	$\tilde{O}(d \log 1/\varepsilon)$	$\tilde{O}(d \log 1/\varepsilon)$	✓

Table 3.1: The contributions of Chapter 3 in context.

While the theoretical study of active learning is still in its infancy, the one nontrivial scenario in which active learning has been shown to give an exponential improvement in sample complexity is that of learning a linear separator for data distributed uniformly over the unit sphere. In this chapter, we have demonstrated that this particular case can be solved by a much simpler algorithm than was previously known: in fact, an online algorithm. It is possible that our algorithm can be molded into something of more general applicability, and so it would be interesting to study its behavior under different circumstances, for instance a different distributional assumption. The uniform input distribution is interesting to study, in that most of the data is close to the decision boundary, but a more common assumption would be to make the two classes Gaussian, or to merely stipulate that they are separated by a margin. Open problems include obtaining performance guarantees for our algorithm, or appropriate variants, in such settings.

Chapter 4

Online active learning: further analysis and application

Section 4.6 is based on joint work with Matti Kääriäinen that is currently in submission [MK06].

In this chapter we analyze the online active learning algorithm of Chapter 3 under several different assumptions, beyond those in Chapter 3. In Section 4.3 we perform a version space analysis of the hypothesis yielded from the algorithm in Chapter 3, and show that it need not remain in the version space. This motivates a "target region" approach to general active learning, in contrast to the version space approach in much of the related work. We introduce and discuss this notion in Section 4.4. We then relax the distributional assumptions from Chapter 3 in various ways, motivated in part by an open problem in active learning which we present in Section 4.5.1. We study the algorithm under relaxed distributional assumptions both theoretically, in Section 4.5.2, providing a label-complexity upper bound when the input distribution is λ-similar to uniform, and empirically, in Section 4.6.

In Section 4.6, in order to empirically assess the algorithm's performance when the distributional and separability assumptions are dropped, we compare its practical performance to another recently proposed online active learning algorithm, with guarantees in a different analysis setting [CBGZ04]. We perform an empirical evaluation of these two algorithms, and their combined variants along with random sampling, on optical character recognition (OCR) data, an application that we argue to be appropriately served by online active learning. We compare the performance between the algorithm variants and show significant reductions in label-complexity over random sampling.

4.1 Related work

Since much of this chapter pertains to the algorithm from Chapter 3, most of the related work has already been discussed in Section 3.2. In this chapter we will additionally make use of the following notions introduced in previous work. The first reference, of which we are aware, to the "region of uncertainty," a notion we use in this chapter, is due to Cohn, Atlas and Ladner (CAL94). We will use a pseudometric over the space of hypotheses that is consistent with recent work by [Kää05; Das05].

Of the algorithms with label-complexity bounds discussed in Chapter 3, the analysis of the algorithm given in that chapter, and the analysis by [BBL06] of the algorithm due to [CAL94], were only performed with respect to the uniform input distribution. Along with analyzing the uniform input distribution case, [FSST97] showed a label-complexity bound of $\tilde{O}(\frac{d}{\lambda}\log\frac{1}{\epsilon})$, when the input distribution is "λ-similar" to uniform, a notion we will define below. Dasgupta also performed such an analysis on his (intractable) scheme for upper bounding label-complexity, yielding a bound of $\tilde{O}(d\log\frac{1}{\lambda}\log^2\frac{1}{\epsilon})$. None of these algorithms, except for the one we introduced in Chapter 3, respect the online constraints of concern in this thesis, so in Section 4.5.2 we will analyze this algorithm under input distributions that are "λ-similar" to uniform.

4.2 Preliminaries

Except when stated, we use the notation and assumptions of Chapter 3. Those definitions are given in Section 3.3. We will additionally define several terms.

The terms defined below apply not only to learning half-spaces but also to general concept learning, since we will occasionally analyze more general cases in this chapter. We consider a selective sampling framework, in which unlabeled samples are drawn independently, in a stream, from an arbitrary fixed distribution, D, referred to as the *input distribution*, which is defined over an input space X. The concept class over which learning is performed is denoted by H. The *version space*, V_t, is defined as the set of hypotheses still consistent with the labelings of all t previously labeled examples. In the realizable case, this is the set of hypotheses whose empirical error after t labeled examples is zero. Denoting a hypothesis v's prediction as $v(x)$, we define the distance between two hypotheses $f, g \in H$ as the probability, under the input distribution, that their classifications disagree, i.e.

$$d(f,g) = P_{x\sim D}[f(x) \neq g(x)].$$

Using this notation, one can express the error rate of hypothesis v_t as $\epsilon_t = d(v_t, u)$. We will assume this notion of distance (which is in fact a pseudo-metric) when referring, even implicitly, to distance. For example a ball, $B(z, r)$ is defined with respect to this pseudo-metric.

We define the *region of uncertainty* as

$$U = \{x \mid \exists f, g \in V \;:\; f(x) \neq g(x)\}.$$

The region of uncertainty can be maintained exactly for certain convex concept classes such as half-spaces, or in general by allowing access to a supervised batch learner.

We will make use of the following notion of λ-*similarity*, as defined in [FSST97].

Definition 2 *A distribution D is λ-similar to a distribution Q on domain X, if for all $A \subseteq X$, the following holds for some $\lambda \leq 1$:*

$$\lambda \leq \frac{P_{x\sim Q}[A]}{P_{x\sim D}[A]} \leq \frac{1}{\lambda}$$

4.3 Version space analysis of DKM algorithm

Here we analyze the algorithm from Chapter 3 (which we will hereafter refer to as DKM) to ascertain whether its hypothesis can leave the version space. We answer this question in the affirmative: it is possible for the hypothesis to misclassify a data point for which it has already observed the label, even in the realizable case. We provide an example in which the hypothesis attained does not correctly classify all the training data and thus is not in the version space. A related observation is that the algorithm's error region, from which it makes label queries, can actually contain points whose labels would already be known by an analogous batch learner.

4.3.1 DKM hypothesis can exit version space

Here we show that even in the realizable setting, and even under the uniform distribution (though our example holds with no assumptions on the input sequence) DKM's hypothesis need not be consistent with the training data. Since it can violate a training data constraint, the hypothesis is thus not always in the version space. This is possible both in the active setting, as well as the supervised setting.

We first discuss a condition that entails that the hypothesis need not be in the version space, and is useful to consider, as it sheds light on potential room for improvement in the algorithm. This is the situation in which the algorithm's error region overlaps with the seen region, the complement of the uncertainty region. As a result, the algorithm's hypothesis misclassifies points that it has either already seen, or that a batch algorithm would be able to label just by convexity from having seen, and stored, other labeled points. Additionally, this situation could cause the algorithm to request labels on a larger region than needed, since only a subset of the algorithm's error region actually intersects with the uncertainty region.

Theorem 6 *There exists a linearly separable sequence of examples such that the DKM online active learning algorithm, when run on that sequence, will yield a hypothesis, v_t, whose current error region, ξ_t, is not contained in the current uncertainty region, U_t.*

Proof: It will suffice to provide a counterexample, i.e. an $x \in \xi_t$, the error region of DKM's hypothesis, such that $x \notin U_t$, the region of uncertainty. We will provide a counterexample in $\mathbb{R}^2$. We will give u, x_0, x_1 and x_2 such that initializing DKM with x_0 and then updating DKM's hypothesis on x_1 and x_2 yields a hypothesis v_3 that misclassifies a point $x \notin U_3$, i.e. the uncertainty region after the three label updates.

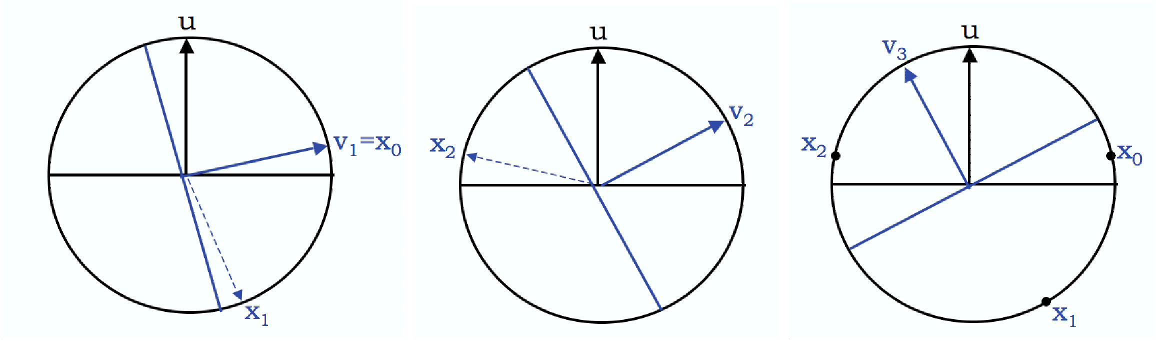


Figure 4-1: Target separator indicated by its normal vector, u. Hypothesis v is initialized with x_0, and updated with x_1, yielding v_2. Next update on x_2 yields v_3. One of the training examples, x_0, is now misclassified.

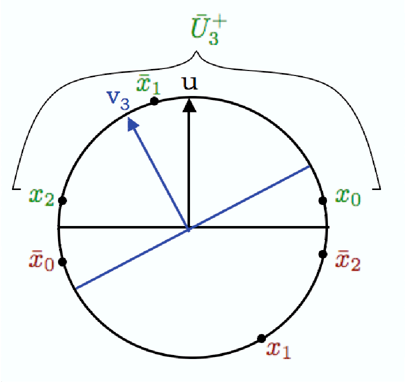


Figure 4-2: The indicated area is the positive "seen" region. It overlaps with the error region: the wedge between the separators indicated by u and v_3.

Since our subscripts only index label queries, these examples can arrive interspersed with arbitrarily many unlabeled examples that fail DKM's active learning rule.

Without loss of generality (as there is spherical symmetry), we will assume $u = (0, 1)$. Consider that x_0, the point that initializes v_1, is $(\frac{\sqrt{2+\sqrt{3}}}{2}, \frac{\sqrt{2-\sqrt{3}}}{2})$. So $v_1 = y_0 x_0 = \texttt{SGN}(u \cdot x_0) x_0 = (\frac{\sqrt{2+\sqrt{3}}}{2}, \frac{\sqrt{2-\sqrt{3}}}{2})$. Let the next point in the stream that fulfills DKM's labeling rule be $x_1 = (\frac{\sqrt{2+\sqrt{3}}-\sqrt{3}}{2\sqrt{2-\sqrt{2+\sqrt{3}}}}, \frac{\sqrt{2-\sqrt{3}}-1}{2\sqrt{2-\sqrt{2+\sqrt{3}}}})$. DKM will choose to label this point since $v_1 \cdot x_1 = \frac{\sqrt{2-\sqrt{2+\sqrt{3}}}}{2} \leq \frac{1}{\sqrt{2}} = \frac{1}{\sqrt{d}}$. This point will yield an update because it is a mistake: $u \cdot x_1 = \frac{\sqrt{2-\sqrt{3}}-1}{2\sqrt{2-\sqrt{2+\sqrt{3}}}} < 0$ whereas $v_1 \cdot x_1 > 0$. The DKM update will yield $v_2 = v_1 - 2(v_1 \cdot x_1)x_1 = (\frac{\sqrt{3}}{2}, \frac{1}{2})$.

If the next point point fulfilling the labeling criterion is $x_2 = (-\frac{\sqrt{2+\sqrt{3}}}{2}, \frac{\sqrt{2-\sqrt{3}}}{2})$, it will be labeled, since $|v_2 \cdot x_2| \leq \frac{1}{\sqrt{d}} = \frac{1}{\sqrt{2}}$. This point will yield an update because it is a mistake: $u \cdot x_2 = \frac{\sqrt{2-\sqrt{3}}}{2} > 0$, whereas $v_2 \cdot x_2 = -\frac{1}{\sqrt{2}} < 0$. The DKM update will yield: $v_3 = v_2 - 2(v_2 \cdot x_2)x_2 = (-\frac{1}{2}, \frac{\sqrt{3}}{2})$. A schematic of this example is provided in Figure 4-1.

Although v_3 now correctly classifies x_2, it will incorrectly classify points that are no longer in the uncertainty region. When learning a half-space, the uncertainty region can reduce from occupying the whole input space as soon as d points have been seen (assuming the points are in general position, i.e. d points that do not lie in a lower dimensional subspace). We will denote the complement of the uncertainty region as, $\bar{U}_3$, and refer to it as the "seen" region (after three labels), where the seen region is actually composed of two convex regions that are antipodal reflections of eachother: the region of seen positive examples, $\bar{U}_3^+$, and the region of seen negative examples, $\bar{U}_3^-$.

In this example in $\mathbb{R}^2$, as shown in Figure 4-2, after having seen the points x_0, x_1 and x_2, $\bar{U}_3^+$ is the (shorter) arc along the surface of the ball in $\mathbb{R}^2$ from x_0 to x_2, and $\bar{U}_3^-$ is its antipodal reflection, connecting $\bar{x}_0$ to $\bar{x}_2$, where $\bar{x}$ is the antipodal reflection of x. However the error region after these three labels, $\xi_3 = \{x \in \mathbb{R}^d, \|x\| = 1 \bigm| \texttt{SGN}(u \cdot x) \neq \texttt{SGN}(v_3 \cdot x)\}$ is the (shorter) arc from $(1, 0)$ to $(\frac{\sqrt{3}}{2}, \frac{1}{2})$. Thus the arc from $x_0 = (\frac{\sqrt{2+\sqrt{3}}}{2}, \frac{\sqrt{2-\sqrt{3}}}{2})$ to $(\frac{\sqrt{3}}{2}, \frac{1}{2})$ is simultaneously in ξ_3 and $\bar{U}_3^+$, and thus not in U, the uncertainty region. □

While we just chose one example that was exact to write down in rational numbers, such a scenario is sufficiently general. The example can be easily modified to hold for v_t, where t is an arbitrary number of updates, including the value of the bound in Chapter 3. Our example remains unchanged as x_0 descends arbitrarily close to $(1, 0)$ from above (and x_1 is instantiated appropriately to attain the same value for v_2). However, although it is easy to show parts of the "seen" region that are misclassified, we have not yet ascertained whether for a training point to be violated it had to have been the point initializing v.

DKM need not be in version space

Theorem 6 implies the following corollary.

Corollary 2 *There exists a linearly separable sequence of examples such that the DKM online active learning algorithm, when run on that sequence, will yield a hypothesis, v_t, that misclassifies at least one training data point.*

Given Theorem 6, we can prove the Corollary by applying Lemma 9, below, that shows that the condition from Theorem 6 implies that DKM need not be in the version space.

Lemma 9 is general in that it applies to arbitrary concept learning (as opposed to just linear separators) in the realizable case. Thus for this lemma, we generalize the definition of the error region to $\xi_t = \{x \in X \,|v_t(x) \neq u(x)\}$, which in the linear separator learning context is $\xi_t = \{x \in S \,|\texttt{SGN}(v_t \cdot x) \neq \texttt{SGN}(u \cdot x)\}$, consistent with its definition in Section 3.3. The following two lemmas make no reference to the particular algorithm yielding the hypothesis.

Lemma 9 *When learning an arbitrary concept in the realizable setting, if the error region of a hypothesis is not contained in the uncertainty region of the hypothesis, then the hypothesis is not in the version space. Or:*

$$\exists x \in \xi_t : x \notin U_t \Rightarrow v_t \notin V_t.$$

Proof: We will prove the contrapositive, i.e.

$$v_t \in V_t \Rightarrow (x \in \xi_t \Rightarrow x \in U_t).$$

By the definition of the error region ξ_t, if $x \in \xi_t$ then $v_t(x) \neq u(x)$. U_t is defined as all points in the input space such that there exists at least one pair of classifiers in the current version space whose classifications on that point disagree. In the realizable setting $u \in V_t$ for all t. So when $v_t \in V_t$, one pair of classifiers in the version space that disagree on the classification of $x \in \xi_t$ is the pair u, v_t, and thus $x \in U_t$. □

An example in which DKM misclassifies a point it has actually already been trained on occurs in the proof of Theorem 6. The final hypothesis, v_3, will misclassify x_0, since $u \cdot x_0 = \frac{\sqrt{2-\sqrt{3}}}{2} > 0$ whereas $v_3 \cdot x_0 = \frac{\sqrt{3(2-\sqrt{3})}-\sqrt{2+\sqrt{3}}}{4} < 0$, and x_0 was one of the previously seen labeled training points. This can be observed by comparing the classifications of u and v_3 of point x_0, in Figure 4-2.

We note here that, at least in the case of learning linear separators, the converse of Lemma 9 does not hold. We will show this in the following lemma.

Lemma 10 *When learning linear separators in the realizable setting, if a hypothesis is not in the version space, it is possible for its error region to be contained in its uncertainty region, i.e. $v_t \notin V_t$ need* not *imply $\exists x \in \xi_t, x \notin U_t$.*

Proof: In $\mathbb{R}^d$, it takes at least d labeled examples before the seen region will have non-zero surface area in $\mathbb{R}^d$. So any hypothesis that misclassifies a training example (and is thus outside the version space) after fewer than d labeled examples will have its error contained in the uncertainty region, as the uncertainty region will still be the entire input space. □

4.3.2 Discussion

In comparison to previous work, the hypothesis updated by the Query by Committee algorithm is always in the version space, as it is the Gibbs classification. However this algorithm is not online and its analysis relies on Bayesian assumptions. The hypothesis generated by standard Perceptron can leave the version space. We have not yet checked this on all recent variants such as [CBGZ04; CBCG05].

The observation that DKM's error region is not a subset of the uncertainty region seems to leave room for improvement in label-complexity. The algorithm is designed to query for labels on a constant fraction of the error region. However if parts of the error region are redundant, in that they are not even in the uncertainty region, it would seem the algorithm could make due with fewer labels. At least in the uniform case however, DKM already attains the optimal label-complexity, in addition to respecting online constraints. The only suggestion we can put forth in order to perhaps decrease label-complexity in the general case would require knowledge of the "seen" region. This can be implemented by keeping the convex hull of all seen mistakes (for certain convex concepts), or via reduction to supervised batch learning, in general. It seems unlikely for a method augmented in this way to still meet online constraints.

As we will discuss further in the next section, our results suggest that, following DKM's example, it may not be necessary to track the version space in the process of performing active learning

4.4 Target region vs. version space approach

Regardless of its possible applicability to online active learning, we will take this section to consider the possible implications of the observation of Section 4.3 on the design of general active learning algorithms, even in the batch case. The discussion here will leave the concept class and input distribution as general as possible. Our results in Section 4.3 suggest that in order to attain a hypothesis within some distance (in terms of probability of disagreement) from the target, the current hypothesis need not always be in the version space, i.e. consistent with all training examples seen so far (in the online setting). Moreover, even the final concept (or the one learned in a batch setting) can err on a fraction of the training examples if the fraction is small.

First we note that the observation above may aid in improvements to the upper bound on label-complexity for active learning of arbitrary concepts under arbitrary input distributions, in the batch setting, due to [Das05], since the proof idea involves cutting the discretization of the version space based on labeled examples. Thus the final hypothesis chosen cannot misclassify any of the training examples. Since it may be the case that some hypotheses within error ϵ from the target do err on some of the training examples, including such hypotheses in the search could reduce the number of label queries needed to find a hypothesis ϵ-close to the target.

4.4.1 Target region approach

The conclusions of our version space analysis motivate the notion of the *target region.* For a fixed $\epsilon \leq 1$, we define the target region as $H^* = \{h \in H \bigm| d(h,u) < \epsilon\}$. The target region contains u, but could potentially contain other hypotheses in h, so taking the limit of the number of seen labeled examples to infinity, the target region contains the version space (which is then just $B(u,0)$) as a subset. In Theorem 5 in Chapter 3, we gave a t after which DKM outputs a hypothesis in H^*, and in Section 4.3 we showed that for any t, the hypothesis need not be in V_t. Since, at least in the case when t goes to infinity, H^* can be a superset of V, this might motivate an active learning algorithm aimed at tracking H^* as opposed to V. We will discuss this issue in light of designing a batch active learning algorithm, as adding the online constraint would likely only make the problem harder.

An important issue in designing an efficient algorithm along these lines would of course be implementation: how to maintain either an approximation to H^*, or a set that contains H^*, even just with high probability, efficiently. Setting that problem aside though, we analyze whether for a finite t, H^* and V_t possess any subset or superset relation, as such a relation would lend value to this approach.

4.4.2 Subset analysis of target region and version space

Here we analyze the relation of H^* and V_t and show that for any finite t, neither $H^* \subseteq V_t$ nor $V_t \subseteq H^*$ need hold. Our proofs are algorithm-independent: they are not concerned with whether a learning algorithm could yield the hypothesis, h used in the proof, and they are shown in the supervised case without addressing whether an active learning algorithm could yield, as a labeled sequence, the sequence of examples used in the proof. Thus when analyzing a *specific* algorithm, some subset relation may actually hold between H^* and V_t.

Lemma 11 *For any fixed $\epsilon \leq 1$, there can exist a concept class H and an input distribution D such that for any finite t, $H^* \subseteq V_t$ need not hold.*

Proof: We will show that for any finite t, there can exist $h \in H^*$, and a series of t examples, such that $h \notin V_t$. By the definition of h, if there exists an h such that $0 < d(h,u) < \epsilon$, then $h \in H^*$. Since $d(h,u) > 0$ is equivalent to $P_{x \sim D}[h(x) \neq u(x)] > 0$, with constant probability any x is misclassified by h. Thus for any t, any sequence of t examples could contain an x that h misclassifies, in which case $h \notin V_t$. □

Section 4.3 provided a sequence of labeled examples that illustrate a special case of Lemma 11, that of learning linear separators, when h and the active learning rule are specified by the update and filtering rules from Chapter 3, respectively.

Lemma 12 *For any fixed $\epsilon \leq 1$, there can exist a concept class H and an input distribution D such that for any finite t, $V_t \subseteq H^*$ need not hold.*

Proof: We will show that for any finite t, there can exist $h \notin H^*$, and a series of t examples, such that $h \in V_t$. The version space, V_t, by definition, contains any $h \in H$ that has made zero mistakes on the t seen labeled examples. By definition of H^*, if there exists an h such that $\epsilon < d(h, u) < 1$ then $h \notin H^*$. Since $d(h, u) < 1$ is equivalent to $P_{x \sim D}[h(x) \neq u(x)] < 1$, then for such an h, we have that $P_{x \sim D}[h(x) = u(x)] > 0$. For any t there exists at least one sequence of examples $x_1, \ldots, x_t$ such that with non-zero probability $h(x_i) = u(x_i)$ for all $i, \in \{1 \ldots, t\}$, for example an iid sequence of draws from D: $P_{x_i \sim D}[h(x_i) = u(x_i)] > 0$ for each x_i, and by independence, the probability of the correct classification of the whole sequence is just the product of the individual probabilities, which are each non-zero. In the case of that sequence being observed, since h correctly classifies all examples in the sequence, $h \in V_t$. □

As mentioned above, when analyzing a particular algorithm, some subset relation may actually hold between these two sets, at least with some probability. If that were the case, the goal in designing an algorithm would be to approximate or approximately contain H^* efficiently. Currently we are not aware of any tractable methods to approximate the version space, unless it has special properties, such as being spherical with high probability, which occurs for example when the concept class is linear separators and the input distribution is uniform. In that case Perceptron approximates the version space by its center, as does the variant in Chapter 3, which combined with the active learning mechanism of Chapter 3 yields the label-complexity guarantees provided therein.

As to whether a method that approximates or approximately contains H^*, if tractable, would yield label-complexity savings over a version space method, it is important to note that by uniform convergence, the version space converges to $B(u, 0)$ at a rate $\frac{1}{t}$, in a realizable, supervised iid setting. The rate of convergence of some approximation or superset of H^* to H^* could not be faster than this, since the rate of convergence of frequencies to probabilities increases with the value of the probability, and is at best $\frac{1}{t}$, for probabilities close to zero. So in approximating $H^* = B(u, \epsilon)$, convergence to hypotheses with distance greater than zero from u can only be slower than convergence to $B(u, 0)$.

4.5 Relaxing distributional assumptions on DKM

In this section and the next, we relax the distributional assumptions from Chapter 3 in various ways, and study the algorithms from Chapter 3 theoretically in this section, and empirically in Section 4.6.

One motivation for attaining performance guarantees under distributional assumptions that are more relaxed than those of Chapter 3, is that such an analysis could potentially provide a preliminary answer to an open problem we recently proposed in [Mon06], and present here. This is a general problem in active learning so it could be solved by the analysis of a batch algorithm; however, solving it via the analysis of an online algorithm such as DKM would suffice, and would provide even more efficiency than required by the open problem.

4.5.1 Motivation: open problem in active learning

Here we describe an open problem concerning efficient algorithms for general active learning. The purpose of this open problem is to probe to what extent the PAC-like selective sampling model of active learning helps in yielding label-complexity savings beyond PAC sample complexity. So we seek to pose the simplest problem such that if active learning is a useful model, it should be solvable. By useful we mean that studying the model yields efficient algorithms with label-complexity bounds less than PAC. In order to simplify the problem we remove the aspects that could be solved via unsupervised learning. Thus we seek to pinpoint the active learning problem only, in order to determine its difficulty.

While the analysis of selective sampling is still in its infancy, we focus here on one of the (seemingly) simplest problems that remain open. Given a pool of unlabeled examples, drawn iid from an arbitrary input distribution known to the learner, and oracle access to their labels, the objective is to achieve a target error rate with minimum label-complexity, via an *efficient* algorithm. No prior distribution is assumed over the concept class, however the problem remains open even under the realizability assumption: there exists a target hypothesis in the concept class that perfectly classifies all examples, and the labeling oracle is noiseless.[1] As a precise variant of the problem, we consider the case of learning homogeneous half-spaces in the realizable setting: unlabeled examples, x_t, are drawn i.i.d. from a known distribution D over the surface of the unit ball in $\mathbb{R}^d$ and labels y_t are either -1 or $+1$. The target function is a half-space $u \cdot x \geq 0$ represented by a unit vector $u \in \mathbb{R}^d$ such that $y_t(u \cdot x_t) > 0$ for all t. Predictions are of the form $v(x) = \texttt{SGN}(v \cdot x)$.

Problem: Provide an algorithm for active learning of half-spaces, such that (with high probability with respect to D and any internal randomness):

1. After L label queries, the algorithm's hypothesis v obeys $P_{x \sim D}[v(x) \neq u(x)] < \epsilon$.

2. L is at most the PAC sample complexity of the supervised problem, $\tilde{O}(\frac{d}{\epsilon} \log \frac{1}{\epsilon})$, and for a general class of input distributions, L is significantly lower.

3. Running time is at most poly$(d, \frac{1}{\epsilon})$.

The assumption that D is known reflects the attempt to extract what is difficult about active learning from the larger problem that involves unsupervised learning. D

[1] In the general setting, the target is the member of the concept class with minimal error rate on the full input distribution, with respect to the (possibly noisy) oracle.

could either be "known" approximately, via an initial unsupervised learning phase, or known exactly, in a new model: there is infinite unlabeled data for computing D, but the learner only has oracle access to labels on a finite subset. This notion is somewhat analogous to a semi-supervised model, although it is still less constrained in that within the subset with oracle access to labels, the learner has the freedom to only query labels on an even smaller subset. In this case we are not concerned with the total running time being polynomial, but the time to choose queries, and make appropriate updates. Even these operations are intractable in existing schemes currently providing label-complexity upper bounds under general distributions.

Left unspecified by this problem is the definition of a suitably "general class of input distributions." While the standard PAC bound for the specific problem variant above is $\tilde{O}(\frac{d}{\epsilon}\log\frac{1}{\epsilon})$, Dasgupta provided a lower bound on label-complexity of $\Omega(\frac{1}{\epsilon})$ [Das04], so there does not appear to be a significant amount of slack. However, a pathological distribution (confined to a lower dimensional subspace) yields the lower bound. Thus could the class of input distributions be defined in such a way so as to avoid the lower bound? In the uniform case, the PAC complexity is $\tilde{\Theta}(\frac{d}{\epsilon})$ [Lon95; Lon03], however in the active learning model, we have shown in Chapter 3 a label-complexity upper bound of $\tilde{O}(d\log\frac{1}{\epsilon})$. This is the type of asymptotic savings we would hope for in a more general case, however it remains to be defined what would be considered sufficiently general in terms of input distributions.

Other open variants

Along with the simple version stated here, the following variants remain open. It is clearly also an open problem when D is unknown to the learner. The agnostic setting, under certain scenarios could be studied, however, the fully agnostic setting faces the lower bound of [Kää06]. An analogous goal could be defined for other concept classes, or for an algorithm that can learn general concepts. Along the lines of this thesis, it would be interesting to additionally apply online constraints to this problem. I.e., not only must data access be sequential, but the storage and time complexity of the online update must not scale with the number of seen labels or mistakes.

State of the art

The state of the art falls into several categories. Though we have touched on most of the related works in Chapter 3 (Section 3.2) or in Section 4.1, here we characterize them with respect to the open problem.

Recent work has provided several negative results. In Chapter 3 we showed that standard Perceptron requires $\Omega(\frac{1}{\epsilon^2})$ labels under the uniform, using any active learning rule. Dasgupta [Das05] provided a general lower bound for learning half-spaces of $\Omega(\frac{1}{\epsilon})$ labels, when the size of the unlabeled sample is bounded. Kääriäinen provided a lower bound of $\Omega(\frac{\eta^2}{\epsilon^2})$, where η is the generalization error of the best hypothesis in the concept class. [Kää06].

Several of the positive results to date have been based on intractable algorithms. Dasgupta [Das05] gave a general upper bound on labels for selective sampling to learn arbitrary concepts under arbitrary input distributions, which for half-spaces under distributions λ-similar to uniform is $\tilde{O}(d \log \frac{1}{\lambda} \log^2 \frac{1}{\epsilon})$. The algorithm achieving the bound is intractable: exponential storage and computation are required, as well as access to an exponential number of functions in the concept class (not just their predictions). Similarly, recent work by Balcan, Beygelzimer and Langford [BBL06] provides an upper bound on label-complexity of $\tilde{O}(d^2 \log \frac{1}{\epsilon})$ for learning half-spaces under the uniform, in a certain agnostic scenario, via an intractable algorithm.

Several selective sampling algorithms have been shown to work in practice, e.g. [LG94]. Some lack performance guarantees, or have been analyzed under different assumptions than those of this problem (e.g. the regret framework [CBGZ04]). Under a Bayesian assumption, Freund et al. [FSST97] gave a bound on label-complexity for QBC of $\tilde{O}(d \log \frac{1}{\epsilon})$ for learning half-spaces under the uniform, and $\tilde{O}(\frac{d}{\lambda} \log \frac{1}{\epsilon})$ under "λ-similar" to uniform. Not only is the algorithm computationally complex, requiring sampling from the version space, but also the Bayesian assumption is significantly stronger than the realizability assumption considered here.

There have also been some positive results for efficient algorithms for this setting, however to date the analyses have only been performed with respect to the uniform input distributions such as the $\tilde{O}(d^2 \log \frac{1}{\epsilon})$ bound of [BBL06] in the realizable case, and our $\tilde{O}(d \log \frac{1}{\epsilon})$ label-complexity upper bound in Chapter 3. With regard to the larger goals of this thesis, only the latter algorithm is online.

Possible approaches

It is important to note that solving this open problem might only require a new analysis, not necessarily a new algorithm. It may be the case, for example, that some algorithms previously only analyzed in the regret framework, could yield the desired label-complexity guarantees with the appropriate analysis. If this were the case, then perhaps the solution to this open problem could actually be with an algorithm that is online in the sense we are concerned with in this thesis.

4.5.2 A label-complexity upper bound under λ-similar to uniform

As a first step towards relaxing the distributional assumption, we extend the analysis from Chapter 3 to input distributions that are λ-similar to uniform.

We make use of Definition 2. In particular, a distribution D is λ-similar to the uniform distribution, U, if:

$$\begin{aligned} \lambda &\leq \frac{U[A]}{P_{x \sim D}[A]} \leq \frac{1}{\lambda} \\ \lambda \, P_{x \sim D}[A] &\leq U[A] \leq \frac{1}{\lambda} P_{x \sim D}[A] \end{aligned}$$

This provides the following relations:

$$P_{x\sim D}[A] \;\leq\; \frac{1}{\lambda}U[A] \tag{4.1}$$

$$P_{x\sim D}[A] \;\geq\; \lambda\, U[A] \tag{4.2}$$

We will now use these relations to prove a label-complexity upper bound on the algorithm in Chapter 3, for input distributions λ-*similar* to uniform. As in Chapter 3, the bound will depend on d, the dimension, and ϵ, the final error rate on the input distribution, and in addition the bound will depend on λ.

Theorem 7 *With probability $1-\delta$, the online active learning algorithm stated in Chapter 3, given iid samples from a distribution D that is λ-similar to the uniform, will reach error $\leq \epsilon$ after $\tilde{O}(poly(\frac{1}{\lambda})\, d\log\frac{1}{\epsilon})$ label queries, and make at most $\tilde{O}(poly(\frac{1}{\lambda})\, d\log\frac{1}{\epsilon})$ errors (on all points, labeled or not).*

Proof: We will closely follow the proof of Theorem 5 in Chapter 3. We will need to modify several lemmas. We start by showing that Lemma 6 of Chapter 3 holds exactly as stated however with probability $\frac{\lambda^2}{4}$, as opposed to $\frac{1}{4}$ as in the uniform case, i.e. we will show the following lemma.

Lemma 13 *For any $\gamma \in \left(0, \sqrt{\frac{1-(u\cdot v_t)^2}{4d}}\right]$,*

$$P_{x_t\sim D}\left[x_t \in \xi_t \;\middle|\; |x_t \cdot v_t| < \gamma\right] \geq \frac{\lambda^2}{4}$$

Proof: By the exact initial argument in Lemma 6, we have that it would suffice to show:

$$P_{x\sim D}\left[\hat{u}'\cdot\hat{x}' < \frac{-1}{\sqrt{4d(1-1/(4d))}} \;\middle|\; 0 \leq x\cdot v_t \leq \gamma\right] \geq \frac{\lambda^2}{4}$$

We proceed to expand the left hand side. Noting that

$$P_{x\sim D}[A|B] = \frac{P_{x\sim D}[A,B]}{P_{x\sim D}[B]} \geq \frac{\lambda\, U[A,B]}{\frac{1}{\lambda}U[B]} = \lambda^2 U[A|B]$$

where the inequality follows by applying (4.2) and (4.1) to the numerator and denominator, respectively. By the proof of Lemma 6, we have that

$$P_{x\sim U}\left[\hat{u}'\cdot\hat{x}' < \frac{-1}{\sqrt{4d(1-1/(4d))}} \;\middle|\; 0 \leq x\cdot v_t \leq \gamma\right] \geq \frac{1}{4}$$

Thus

$$P_{x\sim D}\left[\hat{u}'\cdot\hat{x}' < \frac{-1}{\sqrt{4d(1-1/(4d))}} \;\middle|\; 0 \leq x\cdot v_t \leq \gamma\right] \geq \frac{\lambda^2}{4}$$

and we are done, using the mechanics of Lemma 6 mentioned above. □

This leads directly to a version of Lemma 5 from Chapter 3, which we will state here without proof as the proof is identical to the proof of Lemma 5.

Lemma 14 *With probability at least* $1 - L\big(1 - \frac{\lambda^2}{4}\big)^R$*, we have:*

$$s_t \geq \sqrt{\frac{1-(u\cdot v_t)^2}{16d}} \textit{ for } t = 1, 2, \ldots, L, \textit{ simultaneously.}$$

where L *is the total number of label queries.*

We now prove a modification of Lemma 7 from Chapter 3.

Lemma 15 *Given that* $s_t \geq \sqrt{(1-(u\cdot v_t)^2)/(16d)}$*, upon the* t*th update, each erroneous example is queried with probability at least* $\lambda^4/32$*, i.e.,*

$$P_{x\sim D}\left[|x\cdot v_t| \leq s_t \;\middle|\; x\in\xi_t\right] \geq \frac{\lambda^4}{32}.$$

Proof: First we simplify $P_{x\sim D}\left[x\in\xi_t \wedge |x\cdot v_t|\leq s_t\right]$. Applying Lemma 14,

$$P_{x\sim D}\left[x\in\xi_t \wedge |x\cdot v_t|\leq s_t\right] \quad\geq\quad P_{x\sim D}\left[x\in\xi_t \wedge |x\cdot v_t| \leq \sqrt{\frac{1-(u\cdot v_t)^2}{16d}}\right]$$

Now using

$$P_{x\sim D}[A,B] = P_{x\sim D}[B|A]P_{x\sim D}[A] \geq P_{x\sim D}[B|A]\lambda\, U[A]$$

(where the inequality is by (4.2)), as well as Lemma 13, we can continue to simplify the right hand side.

$$\begin{aligned} P_{x\sim D}\left[x\in\xi_t \wedge |x\cdot v_t| \leq \sqrt{\frac{1-(u\cdot v_t)^2}{16d}}\right] &\geq \frac{\lambda^3}{4}P_{x\sim U}\left[|x\cdot v_t| \leq \sqrt{\frac{1-(u\cdot v_t)^2}{16d}}\right] \\ &\geq \frac{\lambda^3\theta_t}{32\pi} \end{aligned}$$

where the last inequality summarizes several steps that follow the proof of Lemma 7 exactly.

Now we can analyze the quantity $P_{x\sim D}\left[x\in\xi_t \;\middle|\; |x\cdot v_t|\leq s_t\right]$.

$$P_{x\sim D}\left[x\in\xi_t \;\middle|\; |x\cdot v_t|\leq s_t\right] = \frac{P_{x\sim D}\left[x\in\xi_t \wedge |x\cdot v_t|\leq s_t\right]}{P_{x\sim D}[x\in\xi_t]} \geq \frac{\frac{\lambda^3\theta_t}{32\pi}}{\frac{1}{\lambda}\frac{\theta_t}{\pi}} = \frac{\lambda^4}{32}$$

where we apply the lower bound on the numerator we just derived, as well as the upper bound on the denominator from (4.1). □

Given Lemma 15, assuming Lemma 13 holds, we query for labels on at least $\frac{\lambda^4}{32}$ of the total errors. In order to show that a *constant* fraction of our label queries yield "good" updates, i.e. updates that decrease the error of the algorithm's hypothesis by a factor we will instantiate below, it will suffice to show that the probability of the update made on a given error being "good" is at least $1 - \frac{\lambda^4}{64}$. Since we are querying labels on at least a $\frac{2\lambda^4}{64}$ fraction of errors, if at most a $\frac{\lambda^4}{64}$ fraction of errors are not good, then at least half of our label queries yield good updates. Therefore we prove a modification of Lemma 4 from Chapter 3.

Lemma 16 *For any v_t, with probability at least $1 - \frac{\lambda^4}{64}$,*

$$1 - v_{t+1} \cdot u \leq (1 - v_t \cdot u)\left(1 - \frac{c\lambda^{12}}{d}\right).$$

where c is a constant.

Proof: We proceed very similarly to the proof of Lemma 4, starting with $P[A|B] \leq P[A]/P[B]$.

$$\begin{aligned} P_{x\sim D}\left[|v_t \cdot x| \leq \frac{\lambda^6 \theta_t}{128\pi\sqrt{d}} \;\middle|\; x \in \xi_t\right] &\leq \frac{P_{x\sim D}[|v_t \cdot x| \leq \frac{\lambda^6\theta_t}{128\pi\sqrt{d}}]}{P_{x\sim D}[x \in \xi_t]} \\ &\leq \frac{\frac{1}{\lambda}P_{x\sim U}[|v_t \cdot x| \leq \frac{\lambda^6\theta_t}{128\pi\sqrt{d}}]}{\lambda\frac{\theta_t}{\pi}} \\ &\leq \frac{\frac{1}{\lambda}\frac{\lambda^6}{128}\frac{\theta_t}{\pi}}{\lambda\frac{\theta_t}{\pi}} = \frac{\lambda^4}{128} \end{aligned}$$

For the second inequality we applied (4.1) and (4.2) to the numerator and denominator, respectively. The last inequality is an application of the band lemma (Lemma 2 from Chapter 3).

Following the proof of Lemma 4, the probability that $|u \cdot x|$ is less than the threshold above is bounded identically. So using the union bound on the probability that either quantity is small, we can now lower bound the size of $2|v_t \cdot x||u \cdot x|$ as follows:

$$P_{x\sim D}\left[2|v_t \cdot x||u \cdot x| \geq \frac{2\lambda^{12}\theta_t^2}{(128\pi)^2 d} \;\middle|\; x \in \xi_t\right] \geq 1 - 2\frac{\lambda^4}{128} = 1 - \frac{\lambda^4}{64}$$

Concluding via the identical argument to the proof of Lemma 4, this implies a multiplicative decrease in error of the algorithm's hypothesis by a factor of $(1 - \frac{c\lambda^{12}}{d})$ that holds with probability $1 - \frac{\lambda^4}{64}$, as desired. This also provides us a bound on the expected error.

$$\begin{aligned} E_{x_t \in \xi_t}[1 - v_{t+1} \cdot u] &\leq (1 - v_t \cdot u)\left(1 - \frac{\lambda^4}{64}\right)\left(1 - \frac{c\lambda^{12}}{d}\right) \\ &\leq (1 - v_t \cdot u)\left(1 - \frac{c\lambda^{12}}{d}\right) \qquad (4.3) \end{aligned}$$

This is because with probability at least $1 - \frac{\lambda^4}{64}$ it goes down by a factor of $(1 - \frac{c\lambda^{12}}{d})$ and with the remaining probability it does not increase. □

To finish the proof of Theorem 7, we follow the logic of the proof of Theorem 5 of Chapter 3. We have now shown that at least half of the label queries yield good updates, and Lemma 15 entails that the number of total errors is at most a factor of $\frac{32}{\lambda^4}$ more than the number of label queries. By definition of the algorithm, about $\frac{1}{R}$ label queries are updates. So for our bounds to hold, it remains to show that there exists an $R = \tilde{O}(poly(\frac{1}{\lambda}))$ that allows our probabilistic guarantees to hold and yields the bound on updates and labels that we claim.

We closely follow the proof of Theorem 5, spelled out here with all the required modifications. Let U be the number of updates performed. We know, by Lemma 14 that with probability $1 - L\big(1 - \frac{\lambda^2}{4}\big)^R$,

$$s_t \geq \frac{\sin \theta_t}{4\sqrt{d}} \geq \frac{\theta_t}{2\pi\sqrt{d}} \qquad (4.4)$$

for all t, making use of (3.2). Using Lemma 16, which was computed so that we query labels on at least half of the errors if (4.4) holds, and making use of equation (4.3), we have that for each t which is an update, with probability $1 - L\big(1 - \frac{\lambda^2}{4}\big)^R$,

$$E[1 - u \cdot v_{t+1} | v_t] \leq (1 - u \cdot v_t)\left(1 - \frac{c\lambda^{12}}{2d}\right).$$

because with probability 1/2 it decreases by a factor of $(1 - \frac{c\lambda^{12}}{d})$ and with the remaining probability it does not increase. Hence, after U updates, using Markov's inequality,

$$P\left[1 - u \cdot v_L \geq \frac{4}{\delta}\left(1 - \frac{c\lambda^{12}}{2d}\right)^U\right] \leq \frac{\delta}{4} + L\left(1 - \frac{\lambda^2}{4}\right)^R.$$

In other words, with probability $1 - \delta/4 - L(1 - \frac{\lambda^2}{4})^R$, we also have

$$U \leq \frac{2d}{c\lambda^{12}} \log \frac{4}{\delta(1 - u \cdot v_L)} \leq \frac{2d}{c\lambda^{12}} \log \frac{\pi^2}{\delta\theta_L^2} = O\left(\frac{d}{\lambda^{12}} \log \frac{1}{\delta\epsilon}\right),$$

where for the last inequality we used (3.1). In total, $L \leq R\,(U + \log_2 1/s_L)$. This is because once every R labels we either have at least one update or we decrease s_L by a factor of 2. Equivalently, $s_L \leq 2^{U-L/R}$. Hence, with probability $1-\delta/4-L(1-\frac{\lambda^2}{4})^R$,

$$\frac{\theta_L}{2\pi\sqrt{d}} \leq s_L \leq 2^{O(\frac{d}{\lambda^{12}}\log(1/\delta\epsilon))-L/R}$$

Working backwards, we choose $L/R = \Theta(\frac{d}{\lambda^{12}}\log\frac{1}{\epsilon\delta})$ so that the above expression implies $\frac{\theta_L}{\pi} \leq \epsilon$, as required. To ensure $L(1-\frac{\lambda^2}{4})^R \leq \delta/4$, we choose,

$$R = \frac{4}{\lambda^2}\log\frac{2L}{\delta R} = \Theta\left(\frac{1}{\lambda^2}\log\frac{d\log\frac{1}{\epsilon\delta}}{\lambda^{12}\delta}\right) = O\left(\frac{1}{\lambda^2}\left(\log\frac{d}{\lambda^{12}\delta} + \log\log\frac{1}{\epsilon}\right)\right).$$

Hence, for the L and R chosen in the theorem, with probability $1-\frac{3}{4}\delta$, we have error $\theta_L/\pi < \epsilon$. Finally, either condition (4.4) fails or each error is queried with probability at least $\lambda^4/32$. By the multiplicative Chernoff bound

$$Pr[E > 2\frac{32}{\lambda^4}U] \leq e^{-\frac{32U}{3\lambda^4}} \leq \frac{\delta}{4} \tag{4.5}$$

So if there were a total of $E > 64U/\lambda^4$ errors, then with probability $\geq 1-\delta/4$, at least $\lambda^4 E/64 > U$ would have been caught and used as updates. Hence, with probability at most $1-\delta$, we have achieved the target error using $\tilde{O}(poly(\frac{1}{\lambda})\, d\log\frac{1}{\epsilon})$ label queries, and incurring at most $\tilde{O}(poly(\frac{1}{\lambda})\, d\log\frac{1}{\epsilon})$ errors (labeled or unlabeled). □

Discussion

We have made no attempt to minimize the exponent on $\frac{1}{\lambda}$ in the label-complexity bound, as the goal was simply to prove the $poly(\frac{1}{\lambda})$ dependence. Tightening the bound is left for future work.

The extent to which Theorem 7 addresses the open problem from Section 4.5.1 depends on the question of how to define a suitably general class of input distributions, and whether λ-similar to uniform could be considered suitably general. It also depends on whether the bound is considered tight enough in terms of its dependence on λ. The only comparable bounds are for methods that are not online. A bound logarithmic in $\frac{1}{\lambda}$ was shown in the hypothetical case in which the method need not be a tractable algorithm [Das05]. Another analysis yields a bound that is linear in $\frac{1}{\lambda}$ [FSST97], however it relies on the QBC algorithm, whose storage and time complexity scales with the number of seen mistakes, and the analysis includes a Bayesian assumption. While the dependence in Theorem 7 is polynomial in $\frac{1}{\lambda}$, as far as we are aware it is the only such upper bound for an efficient algorithm, in the online active learning setting we are considering. To the extent that the class of distributions is considered general, it answers the open problem in that it is polynomial in the salient parameters.

4.6 Comparison of online active learning algorithms in application to OCR

We will now explore the empirical performance of DKM, the algorithm from Chapter 3, when the distributional assumptions, and in several cases even the separability assumption, are violated, by applying it to real data from OCR, an application that we deem particularly appropriate for strongly online active learning. We compare DKM's performance to another state-of-the-art strongly online active learning algorithm due to Cesa-Bianchi, Gentile and Zaniboni (CBGZ) [CBGZ04], which has performance bounds in the individual sequence prediction context. We focus on these algorithms (and their combined variants) as they are the only algorithms of which we are aware that both have some form of theoretical guarantee and perform selective sampling while meeting both of the online constraints of concern in this thesis. In fact, as they are both based on Perceptron variants, they each only store a single vector and their algorithmic form is very light-weight and easy to implement.

The two algorithms have been analyzed under disjoint assumptions but have not been compared theoretically. Additionally, since some of their analysis assumptions are rather limiting, we evaluate them on real data, in order to assess their performance when these assumptions are removed. In doing so, we also illustrate the useful application of online active learning to optical character recognition (OCR). OCR is a particularly appropriate application of online active learning, due to the potential need for online active learning for OCR training on small devices, as we motivated in the introduction to this thesis, in Section 1.3.3. We compare the performance on this problem between the algorithm variants and show significant reductions in label-complexity over random sampling.

4.6.1 Algorithms

The algorithms we consider are both for learning linear separators through the origin, in the sequential selective sampling framework. Each algorithm can be decomposed into two parts: an active learning mechanism, wrapped around a sub-algorithm that implements supervised learning.

Application of DKM to the non-uniform setting

In Figure 4-3 we restate DKM, indicating the variant we apply to the non-uniform setting and providing a higher level of implementation detail than in Chapter 3, for the purpose of comparison to CBGZ. In applying the algorithm from Chapter 3 to the non-uniform setting we changed the initial setting of the active learning threshold. We used $s_1 = \frac{1}{\sqrt{d}}$ in the uniform case based on a fact about uniform random projections, Lemma 2, that need not hold when the distribution is non-uniform. Instead of starting the initial learning threshold so low, we make no assumptions about the input distribution and thus set the initial threshold to the maximum value that $|x \cdot v_t|$ could take, which is one, since the algorithm guarantees that, given input vectors x_t such that $\|x_t\| = 1$, $\|v_t\| = 1$ for all t. Changing the initial active learning threshold

```
Initialization:  $s_1 = 1$, $\tau = 0$, $t = 1$, $v_1 = x_0 y_0$, $\tau = 0$.
**Do**
  Receive $x$.
  Predict $\hat{y} = sign(x \cdot v_t)$.
  **If** $|x \cdot v_t| \leq s_t$ **then**:
    Query the label $y \in \{-1, +1\}$ of $x$, and set $(x_t, y_t) = (x, y)$.
    **If** $(x_t \cdot v_t) y_t < 0$, **then**:
      $v_{t+1} = v_t - 2(v_t \cdot x_t) x_t$
      $s_{t+1} = s_t$
      $\tau = 0$
    **else**:
      $v_{t+1} = v_t$
      $\tau = \tau + 1$
      **If** $\tau \geq R$, **then**:
        $s_{t+1} = s_t / 2$
        $\tau = 0$
      **else**:  $s_{t+1} = s_t$.
    $t = t + 1$.
**Until** $t == L$
```

Figure 4-3: The DKM algorithm applied to the non-uniform case, parameterized by R, the waiting time before halving the active learning threshold.

```
Initialization:  $t = 1$, $v_1 = (0, \ldots, 0)^\top$.
**Do**
  Receive $x$.
  Set $\hat{p} = x \cdot v_t$, and predict $\hat{y} = sign(\hat{p})$.
  Toss a coin with $P(\text{Heads}) = \frac{b}{b + |\hat{p}|}$.
  **If** Heads
    Query the label $y \in \{-1, +1\}$ of $x$.
    **If** $y \neq \hat{y}$, **then**:
      $v_{t+1} = v_t + \eta y x^\top$
    **else**:
      $v_{t+1} = v_t$
    $t = t + 1$.
**Until** $t == L$
```

Figure 4-4: The CBGZ algorithm, parameterized by $b > 0$, and learning rate $\eta > 0$.

might imply that R should also differ from the value given in Chapter 3. Regardless, we have not optimized the constants in Chapter 3, so we tuned R, along with the parameters of the other algorithms, as discussed in the evaluation section.

The CBGZ algorithm

Similar to DKM, the strongly online selective sampling algorithms proposed by Cesa-Bianchi et al. [CBGZ04] are based on augmenting Perceptron-type algorithms with a margin-based filtering rule; for the first-order version used in our experiments, see Figure 4-4. The algorithm queries for a label with probability $b/(b + |\hat{p}|)$, where $\hat{p}$ is the margin of the example with respect to the current hypothesis, and $b > 0$ is a parameter.[2] If a label is queried and the algorithm's prediction $sign(\hat{p})$ is incorrect, a standard Perceptron update is performed.

The main result in [CBGZ04] is a bound on the expected (with respect to the algorithm's randomness) number of mistakes the algorithm makes on arbitrary input sequences. Both this mistake bound and the expected number of label queries depend on b. By optimizing b for the mistake bound, one can match the mistake bound for standard Perceptron. However, this choice of b may result in querying almost all the labels. Another complication is that the optimal choice of b depends on the data, and thus in practice is known only in hindsight. To circumvent this issue, the authors provide and analyze a method for tuning b on the fly, but in practice this adaptive strategy has inferior performance [CBGZ04].

The theoretical results for DKM and CBGZ are not directly comparable; CBGZ provides a bound on labels that is sequence dependent, whereas our upper bound from Chapter 3 on label-complexity as a function of the final error rate attained, is sequence independent, assuming the input distribution is uniform. The lack of unified theoretical results motivates our empirical study of the performance of these algorithms on real data.

4.6.2 Evaluation

Comparison class of algorithms

In designing our evaluation, we considered comparing to the SVM-based active learning algorithms proposed by Tong and Koller [TK01], as they are well-known benchmarks for active learning. However our problem framework is different from their pool-based model in which active learners have unlimited access to all unlabeled data. Although their active learning criteria could potentially also be adapted for the sequential setting, their algorithmic form is less constrained: SVMs do not obey the online constraint on storage and running time that we are also concerned with in this thesis.

Given these two extra degrees of freedom, we would expect SVM-based methods with batch access to the data to outperform all the strongly online algorithms. We

[2] For clarity of exposition, we have also listed η, the learning rate of the Perceptron sub-algorithm, as a parameter, however the CBGZ algorithm is in fact only sensitive to changes in the ratio $\frac{b}{\eta}$.

confirmed this with experiments using an SVM as the sub-algorithm, paired both with random queries and with the Simple active learning heuristic [TK01], with batch access to the remaining unlabeled pool. In both cases the number of labels queried was strictly lower than that of all the online algorithms studied, for each associated error rate. Since random sampling from a pool (drawn iid from the input distribution) is equivalent to a stream of iid draws from the input distribution, the random sampling case fulfills the online constraint on data observation. Thus the ability to break the second online constraint (on time and memory) by running the SVM sub-algorithm, suffices to provide improved performance versus the strongly online algorithms. In fact the gains in performance due to using SVMs instead of online methods were greater than those from using the Simple active learning heuristic [TK01] instead of random sampling. Our conclusion from this study is that the online active learning algorithms studied here and in Chapter 3 seem to be most useful in settings with strict online requirements, such as those upon which we focus in this thesis, while methods without online constraints seem to have superior performance when applicable.

Therefore as an appropriate comparison class, we instead consider only algorithms that are strongly online. Thus we compare all six combinations of the two online active learning rules discussed above, as well as random sampling, paired with two strongly online supervised learning algorithms: Perceptron and the supervised update of DKM. We will denote as DKM^2 the exact algorithm from Chapter 3, i.e. the DKM active learning logic with the DKM supervised update as the sub-algorithm. Running DKM's active learning logic with Perceptron as the sub-algorithm, we refer to as DKMactivePerceptron. We will denote as CBGZ, the CBGZ active learning rule with Perceptron as the sub-algorithm, as specified in [CBGZ04]. For the sake of completeness, we also experimented with combining CBGZ's active learning rule and the DKM update, denoted below as CBGZactiveDKMupate. The random sampling methods simply flip a coin as to whether to query the current point's label, and update using Perceptron (randomPerceptron) or the DKM update (randomDKMupdate). This method is equivalent to performing supervised learning with the sub-algorithm in question, as this method yields a sequence of labeled examples that are simply iid samples from the input distribution.

Experiments

We conducted our evaluation on benchmark data from OCR, since OCR on small devices could stand to benefit from strongly online active learning solutions. Additionally, these datasets are known to be non-uniformly distributed over inputs. We used both MNIST [LeC98] and USPS in order to experiment with multiple datasets and dimensionalities ($d = 784$ for MNIST, $d = 256$ for USPS).

We experimented on 7 binary classification problems, 5 from MNIST and two from USPS, each consisting of approximately 10,000 examples. All the problems but two were linearly separable. (Using svmLight we were unable to find separating hyperplanes for the problem {1,4,7} vs. all other characters, both in the MNIST and USPS versions). Since the algorithms access the data in one pass in a sequential fashion, for each problem we ran 5 runs in which the dataset was uniformly re-

MNIST:	0v1 (0.01)	0vAll (0.05)	4v7 (0.05)	6v9 (0.025)	147vAll (0.15)
DKM^2	28.02±25.26	105.30±42.39	150.02±49.61	163.12±42.45	275.34±72.00
DKMperc	**13.78±5.88**	**57.26±15.92**	**44.00±15.32**	**20.44±11.75**	217.06±75.85
CBGZ_DKM	130.12±116.45	183.78±120.83	194.36±80.20	218.28±94.95	379.16±138.38
CBGZperc	32.78±19.52	62.66±30.48	63.32±30.76	30.66±13.31	**170.02±63.61**
randDKM	87.72±101.84	173.44±114.55	276.92±150.59	367.24±191.25	375.46±164.33
randPerc	83.76±78.47	103.74±76.25	107.98±57.41	104.06±75.10	214.16±93.12

USPS:	0vAll (0.05)	147vAll (0.1)
DKM^2	174.22±63.85	190.72±80.09
DKMperc	**87.56±28.97**	137.86±58.21
CBGZ_DKM	156.08±66.75	193.70±96.35
CBGZperc	115.14±61.09	**116.28±60.68**
randDKM	235.10±129.11	210.40±109.39
randPerc	173.96±98.96	151.32±72.65

Table 4.1: Mean and standard deviation (over 5 runs of 10 fold cross-validation) of the minimum number of labels to reach the test error threshold (in parentheses) for the problem.

permuted, of 10 fold cross-validation.

Several of the algorithms have parameters: Perceptron's learning rate, CBGZ's b and DKM active learning version's R, so each was tuned independently on a separate holdout set for each problem, using 10 fold cross-validation on approximately 2,000 examples. A threshold on test error, ϵ, was chosen for each problem based (qualitatively) on level of separability, and each algorithm's parameters were tuned to minimize the number of labels, averaged over folds, to reach test error ϵ.

In Table 4.1 we report the mean and standard deviations, over all the experiments run, of the minimum number of labels after which each algorithm reached the test error threshold (ϵ listed in parentheses) for that problem. In our discussion of the results, we will indicate which conclusions can be drawn from the mean label values reported in the table, with statistical significance. We tested statistical significance using Wilcoxan signed rank hypothesis testing, which is non-parametric and thus robust, and which takes into account the magnitude of the difference in mean labels, between the algorithms compared, for each problem.

The minimum number of labels was attained by DKMactivePerceptron, in all but two of the problems, in which the minimum was achieved by CBGZ. DKMactivePerceptron also reports the smallest variance on this figure, in all but one problem. Both methods used significantly fewer labels than the random sampling methods. We tested this by assuming as the null hypothesis that the active learning method in question did not reduce label-complexity beyond that of Perceptron with random sampling (the best-performing random sampling method), yielding, for CBGZ, a p-value of 0.0156, entailing that the null hypothesis is rejected for significance levels of 1.56% and higher, and, for DKMactivePerceptron, at significance levels of 3.13% and higher ($p = 0.0313$). The difference in these two active learning algorithms' performance, compared pairwise per problem, was not statistically significant however. Interestingly, the problems for which CBGZ was the top performer are the only two unseparable problems. Although both algorithms use Perceptron as the sub-algorithm, we did not have enough unseparable problems to conclude, with statistical significance, whether CBGZ's active learning rule is better equipped for the non-realizable case than that of DKM.

Similarly, using the DKM active learning rule showed significant improvements over using the DKM update with random sampling. DKM^2 used fewer labels per problem than randomDKMupdate, at significance levels of 1.56% and higher. Moreover, we observe that, compared to the best active learning method for each problem, at the comparison test error threshold, Perceptron with random sampling used more labels by a factor between 1.26–6.08, and for more than half of the problems the factor was 2 or higher.

The DKM supervised update, and methods that used it as their sub-algorithm tended to perform worse than their Perceptron counterparts. This is statistically significant at significance levels of 1.56% and higher, for each pairwise comparison between the Perceptron and DKM updates, with the active learning rule fixed. In the unseparable cases, this may be explained by DKM's update being much more aggressive than Perceptron's, when the point has a large margin: the DKM update adds to its hypothesis the same quantity, $y_t x_t$, as Perceptron, however scaled by a factor of $2|x_t \cdot v_t|$, which could be greater than one, as opposed to the Perceptron's learning rate which is less than one. The comparison may not be completely fair however, as the DKM update was the only algorithm without parameters, and thus the only algorithm not at all tuned per problem. In fact, both of the active learning variants of standard Perceptron were actually doubly tuned per problem, as the Perceptron learning rate was first tuned, and then the active learning parameter (R or b) was tuned for the problem, using the tuned Perceptron as the sub-algorithm. It is also important to note that mistake bounds implying better performance of the DKM update than Perceptron have only been shown under the uniform [Chapter 3], and λ similar to uniform [Section 4.5.2], input distributions, and here the input distribution is known to be highly non-uniform.

For both sub-algorithms, the DKM active learning rule tended to outperform the CBGZ active learning rule; with the DKM update as the sub-algorithm, the DKM active learning rule (DKM^2) used fewer labels than that of CBGZ (CBGZactiveDKMupdate) in all problems but one. As mentioned above, for the Perceptron-based methods, this observation does not have statistical support. However for the algorithms using the DKM update as the sub-algorithm, the advantage of the DKM active learning rule over that of CBGZ is statistically significant at significance levels of 4.69% and higher.

In Figure 4-5 we plot statistical efficiency curves. Points indicate the average over all the experiments of the minimum number of labels to reach attain test error lower than a given threshold on test error (i.e. one minus the value plotted on the x-axis), only if all experiments reached that threshold. It is important to note that the algorithms were only tuned to minimize labels to reach one of these test error thresholds; an algorithm that was minimal at the chosen threshold need not be

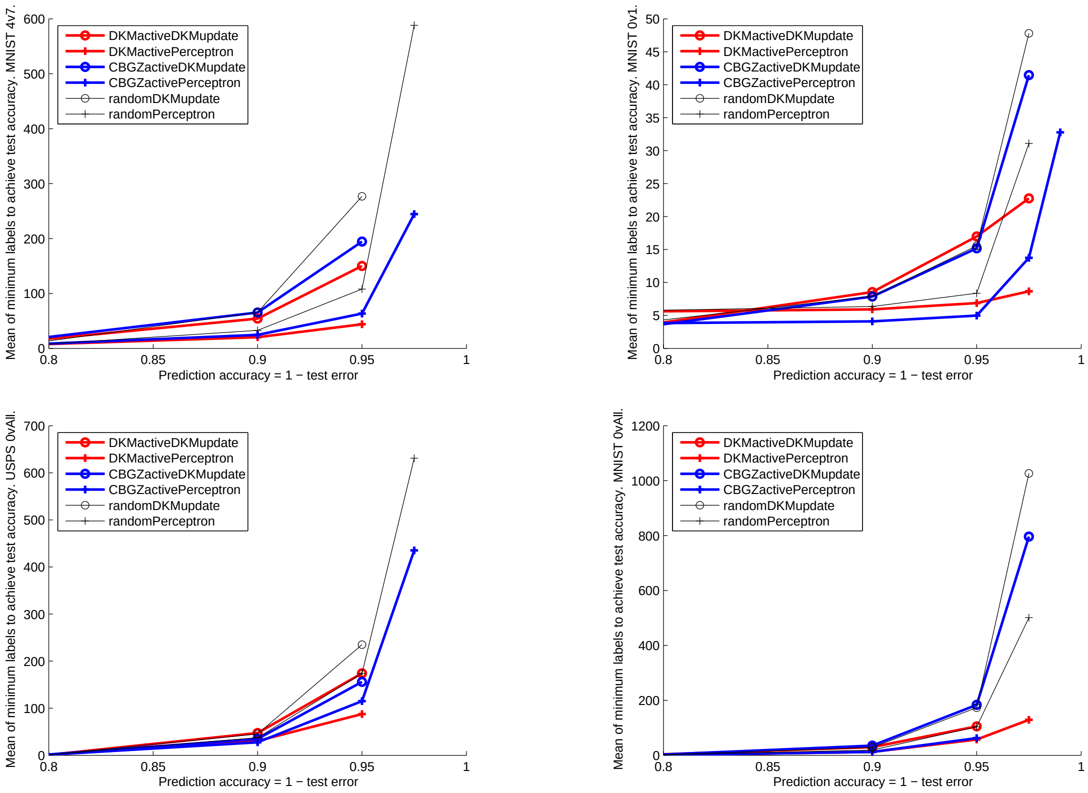


Figure 4-5: Statistical efficiency. Mean minimum labels to attain test accuracy (i.e. $1 -$ test error) above each threshold is over 5 folds 10 runs if all folds/runs reached that test accuracy. a). MNIST 4v7. b) MNIST 0v1. c) USPS 0vAll. d) MNIST 0vAll.

minimal at all thresholds plotted. Some of the plots illustrate a slower rate of label increase for algorithms using DKM as their active learning rule. Not all the plots were as conclusive, but an interesting example is Figure 4-5 b) in which the DKM active algorithms have higher label usage, at most of the thresholds measured, than their CBGZactive counterparts, however the rate of label increase, as the test error decreases (x-axis increases), appears to be much slower.

To provide a qualitative perspective we present some representive learning curves, with respect to labeled examples, in Figure 4-6. We show a) a problem that was particularly easy and b) a problem that we did not find to be linearly separable. Figure 4-6 c) and d) compare the MNIST and USPS versions of the problem of 0 vs. all other characters, which is separable but has a large label imbalance with very few positive examples. In these problems, while DKMactivePerceptron reduces test error at a faster rate than all the other algorithms, DKM^2 and CBGZ continue querying for more labels, eventually reaching lower error.

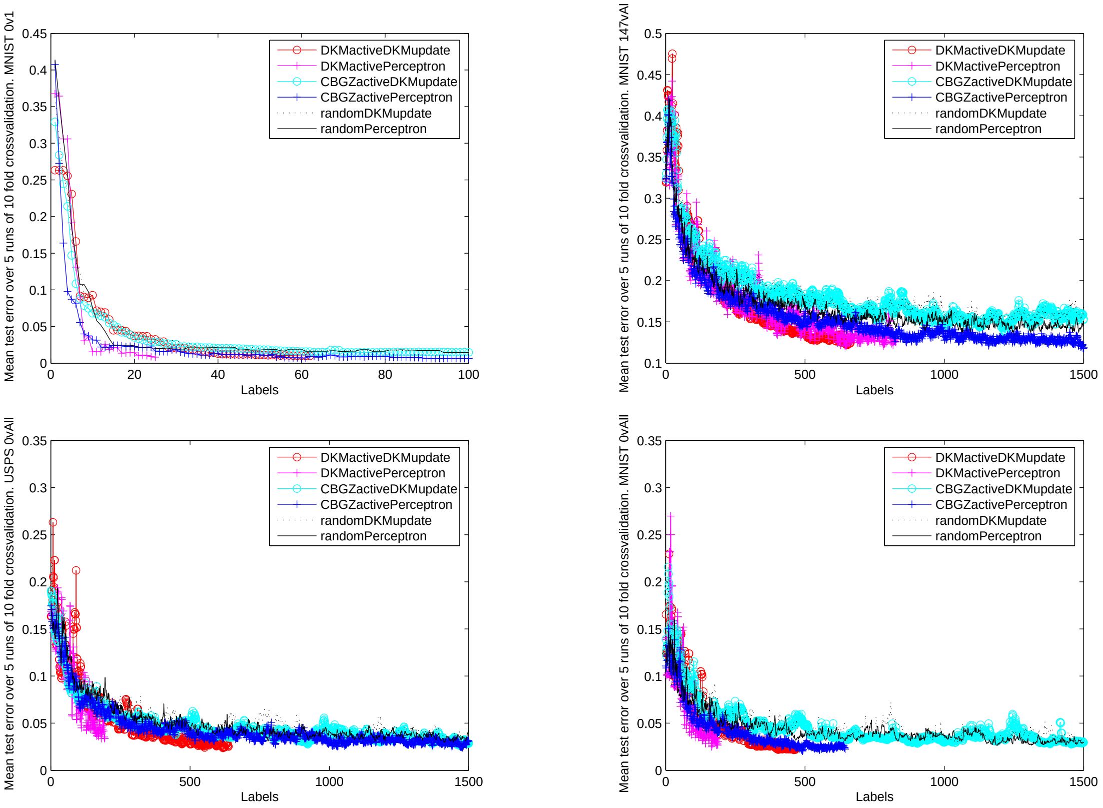


Figure 4-6: Learning curves. a) An extremely separable problem, MNIST 0v1. b) An unseparable problem, MNIST 147vAll. c) USPS 0vAll. d) MNIST 0vAll.

4.6.3 Discussion and conclusions

One potentially detrimental artifact of our experimental framework may have been overfitting of the parameter settings to the holdout tuning set, per problem, which may have prevented some of the algorithms from generalizing well to the actual problem data. Supervised DKM has no parameters, but for all other algorithms, tuning was involved.

Another tuning issue relates to the behavior observed above in Figure 4-6 c) and d), of DKMactivePerceptron attaining an initial test error threshold faster than all other algorithms, but then not querying for any more labels in the rest of the fold. This issue is related to how one should set the waiting time threshold R for the DKM active learning algorithm. With a very small value of R, the algorithm has very little tolerance for labeled examples that do not yield a mistake and thus an update, and so will quickly halve its active learning threshold. Labeling an example with a smaller margin with respect to the current hypothesis, is more likely to yield a mistake. Although this can cause a steep descent of error with respect to the number of labels, once the active learning threshold becomes too small the algorithm will hardly ever make label queries. Since we are experimenting on a finite set of data as opposed to an endless stream, this means the algorithm may not query for any more labels on the fold. Ideally, we would like to optimize the constants in DKM so that the parameter R need not be tuned, but this is left for future work. Additional future work would entail modeling other domains using online active learning, and testing the performance of these algorithms therein.

At a general level, we conclude that online active learning provides significant performance gains over random sampling the same number of labels, when the random sampler must obey online constraints on memory and computation. In particular, we provide an application of DKM, and to our knowledge this algorithm had not yet been applied in practice. We study the performance of DKM when the input distribution is non-uniform, a question left open in Chapter 3, as the performance guarantees were shown under the assumptions of realizability and a uniform input distribution. When these assumptions are violated, we observe that, in this application, DKM active learning has better performance when paired with standard Perceptron as the supervised sub-algorithm, as opposed to the update proposed in Chapter 3. This is an outcome we did not predict, due to the strikingly better performance guarantees of the update proposed in Chapter 3, with respect to Perceptron, under the uniform assumption.

Chapter 5

Conclusion

This thesis presented several theoretical advances for learning with online constraints, and demonstrated them in practical applications. When online constraints are considered for their efficacy in modeling temporal forecasting problems, in which the observations may change with time, we advanced the understanding of a class of shifting algorithms, by providing a negative result: a lower bound on their regret. We applied an algorithm we provided in previous work, that avoids this lower bound, to an energy-management problem in wireless networks, yielding encouraging results in simulation.

In the context of supervised online learning algorithms with mistake bounds under the iid assumption, we gave a negative result for standard Perceptron in the half-space learning framework we analyzed, and we introduce a modification to Perceptron that avoids this lower bound. We showed that our modified algorithm attains the optimal mistake-complexity for this setting.

We motivated online active learning, a combined framework of active learning with online constraints, useful whenever resource-constrained agents perform learning, and labels are hard to come by. Our lower bound applies to the active learning setting as a lower bound on labels for Perceptron, using any active learning rule. We presented an online active learning algorithm for this framework, and showed that it avoids this lower bound. We upper bounded its label-complexity by the optimal for this setting, and showed that this also bounds the algorithm's total errors, labeled and unlabeled. We analyzed the algorithm in several other scenarios, yielding a label-complexity bound that relaxes the distributional assumption. We applied our algorithm, as well as various other online active learning variants, to the problem of optical character recognition, demonstrating encouraging empirical performance even though the analysis assumptions were unmet by the real data.

Along the way, we introduced several analysis techniques, raised some interesting open problems, and made progress towards bridging theory and practice for machine learning in general, and for learning with online constraints, and active learning, in particular. Interestingly our advances come full circle in the study of learning with online constraints, in that our results build on algorithms that are descended from the two canonical forms of online learning algorithms: Winnow [Lit88] and Perceptron [Ros58].

Appendix A

Proof of Claim 1

We start by taking the first derivative of $L_T(\alpha)$, by taking the derivative of $L_T(\alpha) - L_T(\alpha^*)$ with respect to α, since $L_T(\alpha^*)$ is constant with respect to α. We use the form for $L_T(\alpha) - L_T(\alpha^*)$ shown in the proof of Theorem 1 [Mon03], and let $T' = T - 1$, and $f(\hat{\alpha}) = \exp\{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*} + (1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)\}$. Thus

$$\begin{aligned}
\frac{d}{d\alpha}L_T(\alpha) &= -\frac{d}{d\alpha}\log E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right] \\
&= -\frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}\frac{d}{d\alpha}E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right] \\
&= -\frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}E_{\hat{\alpha}\sim Q}\left[\frac{d}{d\alpha}\,f(\hat{\alpha})\right] \\
&= -\frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}E_{\hat{\alpha}\sim Q}\left[e^{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)}T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right]
\end{aligned}$$

We now proceed to take the derivative with respect to α again, yielding

$$\begin{aligned}
\frac{d^2}{d\alpha^2}L_T(\alpha) &= \frac{1}{(E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right])^2}\left(\frac{d}{d\alpha}E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]\right)\times \\
&\qquad \times E_{\hat{\alpha}\sim Q}\left[e^{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)}T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right] - \\
&\quad \frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}\frac{d}{d\alpha}E_{\hat{\alpha}\sim Q}\left[e^{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)}T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right] \\
&= \frac{1}{(E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right])^2}(E_{\hat{\alpha}\sim Q}\left[e^{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)}T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right])^2 \\
&\quad - \frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}E_{\hat{\alpha}\sim Q}\left[\frac{d}{d\alpha}e^{T'\left(\hat{\alpha}\log\frac{\alpha}{\alpha^*}+(1-\hat{\alpha})\log\frac{1-\alpha}{1-\alpha^*}\right)}T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right]
\end{aligned}$$

$$
\begin{aligned}
= \;\; & \frac{1}{(E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right])^2}(E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})T'(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})\right])^2 \;\; - \\
& \frac{1}{E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})\right]}E_{\hat{\alpha}\sim Q}\left[f(\hat{\alpha})(T'^2(\frac{\hat{\alpha}}{\alpha} - \frac{1-\hat{\alpha}}{1-\alpha})^2 + T'(-\frac{\hat{\alpha}}{\alpha^2} - \frac{1-\hat{\alpha}}{(1-\alpha)^2}))\right]
\end{aligned}
$$

We continue by evaluating this expression at $\alpha = \alpha^*$:

$$
\begin{aligned}
\frac{d^2}{d\alpha^2}L_T(\alpha))_{|\alpha=\alpha^*} \;\; = \;\; & (E_{\hat{\alpha}\sim Q}\left[T'(\frac{\hat{\alpha}}{\alpha^*} - \frac{1-\hat{\alpha}}{1-\alpha^*})\right])^2 - \\
& E_{\hat{\alpha}\sim Q}\left[T'^2(\frac{\hat{\alpha}}{\alpha^*} - \frac{1-\hat{\alpha}}{1-\alpha^*})^2 + T'(-\frac{\hat{\alpha}}{\alpha^{*2}} - \frac{1-\hat{\alpha}}{(1-\alpha^*)^2})\right]
\end{aligned}
$$

As shown in the full proof of Theorem 1 (see [Mon03]), the optimality constraint on α^* is equivalent to the following constraint on Q: $E_{\hat{\alpha}\sim Q}[\hat{\alpha}] = \alpha^*$. Thus we can simplify as follows:

$$
\begin{aligned}
= \;\; & 0 \; - \; E_{\hat{\alpha}\sim Q}\left[T'^2(\frac{\hat{\alpha}^2}{\alpha^{*2}} - 2\frac{\hat{\alpha}(1-\hat{\alpha})}{\alpha^*(1-\alpha^*)} + \frac{(1-\hat{\alpha})^2}{(1-\alpha^*)^2})\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & - \; T'^2\left[\frac{1}{\alpha^{*2}}E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}^2\right] - \frac{2}{\alpha^*(1-\alpha^*)}(E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}\right] - E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}^2\right]) + \right. \\
& \qquad \left. \frac{1}{(1-\alpha^*)^2}(1 - 2E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}\right] + E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}^2\right])\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & - \; T'^2\left[E_{\hat{\alpha}\sim Q}\left[\hat{\alpha}^2\right](\frac{1}{\alpha^{*2}} + \frac{2}{\alpha^*(1-\alpha^*)} + \frac{1}{(1-\alpha^*)^2}) - \right. \\
& \qquad \left. \frac{2}{(1-\alpha^*)} + \frac{1}{(1-\alpha^*)^2} - \frac{2\alpha^*}{(1-\alpha^*)^2}\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & - \; T'^2\left[E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2 + 2\hat{\alpha}\alpha^* - \alpha^{*2}\right]\frac{(1-\alpha^*)^2 + 2\alpha^*(1-\alpha^*) + \alpha^{*2}}{\alpha^{*2}(1-\alpha^*)^2} \; - \right. \\
& \qquad \left. \frac{1}{(1-\alpha^*)^2}\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & - \; T'^2\left[(E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right] + \alpha^{*2})\frac{1}{\alpha^{*2}(1-\alpha^*)^2} - \frac{1}{(1-\alpha^*)^2}\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & - \; \frac{T'^2}{\alpha^{*2}(1-\alpha^*)^2}E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right] + \frac{T'}{\alpha^*(1-\alpha^*)} \\
= \;\; & \frac{T'^2}{\alpha^{*2}(1-\alpha^*)^2}\left[\frac{\alpha^*(1-\alpha^*)}{T'} - E_{\hat{\alpha}\sim Q}\left[(\hat{\alpha}-\alpha^*)^2\right]\right]
\end{aligned}
$$

□

Appendix B

Proof of Lemma 2

Let $r = \gamma/\sqrt{d}$ and let A_d be the area of a d-dimensional unit sphere, i.e. the surface of a $(d+1)$-dimensional unit ball.

$$P_x\left[|a \cdot x| \leq r\right] = \frac{\int_{-r}^{r} A_{d-2}(1-z^2)^{\frac{d-2}{2}}dz}{A_{d-1}} = \frac{2A_{d-2}}{A_{d-1}}\int_0^r (1-z^2)^{d/2-1}dz$$

First observe,

$$r(1-r^2)^{d/2-1} \leq \int_0^r (1-z^2)^{d/2-1}dz \leq r \tag{B.1}$$

For $x \in [0, 0.5]$, $1 - x \geq 4^{-x}$. Hence, for $0 \leq r \leq 2^{-1/2}$,

$$(1-r^2)^{d/2-1} \geq 4^{-r^2(d/2-1)} \geq 2^{-r^2 d}.$$

So we can conclude that the integral of (B.1) is in $[r/2, r]$ for $r \in [0, 1/\sqrt{d}]$. The ratio $2A_{d-2}/A_{d-1}$ can be shown to be in the range $[\sqrt{d/3}, \sqrt{d}]$ by straightforward induction on d, using the definition of the Γ function, and the fact that $A_{d-1} = 2\pi^{d/2}/\Gamma(d/2)$.

□

Bibliography

[ACBFS02] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[Ang01] Dana Angluin. Queries revisited. *In Proc. 12th International Conference on Algorithmic Learning Theory*, LNAI,2225:12–31, 2001.

[Bau97] Eric B. Baum. The perceptron algorithm is fast for nonmalicious distributions. *Neural Computation*, 2:248–260, 1997.

[BBK99] Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science*, page 450, New York, New York, October 1999.

[BBL06] Maria-Florina Balcan, Alina Beygelzimer, and John Langford. Agnostic active learning. In *Proc. International Conference on Machine Learning*, 2006.

[Ber96] Berkeley. UC Berkeley home IP web traces. In *http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html*, 1996.

[BFKV96] Avrim Blum, Alan Frieze, Ravi Kannan, and Santosh Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. In *Proc. 37th Annual IEEE Symposium on the Foundations of Computer Science*, 1996.

[CAL94] David A. Cohn, Les Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[CBCG03] Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. Learning probabilistic linear-threshold classifiers via selective sampling. In *Proc. 16th Annual Conference on Learning Theory*, 2003.

[CBCG05] Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. A second-order perceptron algorithm. *SIAM Journal on Computing*, 34(3):640–66, 2005.

[CBGZ04] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis of selective sampling for linear-threshold algorithms. In *Advances in Neural Information Processing Systems 17*, 2004.

[CBM99] Eui-Young Chung, Luca Benini, and Giovanni De Micheli. Dynamic power management for non-stationary service requests. In *Proc. DATE*, pages 77–81, 1999.

[CJBM02] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks Journal*, 8(5), September 2002.

[Das04] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17*, 2004.

[Das05] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In *Advances in Neural Information Processing Systems 18*, 2005.

[DKM05] Sanjoy Dasgupta, Adam Tauman Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. In *Proc. 18th Annual Conference on Learning Theory*, 2005.

[FN01] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. INFOCOM 2001*, Anchorage, Alaska, April 2001.

[FS99] Yoav Freund and Robert Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[FSST97] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[FV99] Dean P. Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29:7–35, 1999.

[GBNT05] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Query by committee made real. In *Advances in Neural Information Processing Systems 18*, 2005.

[HK99] Steven Hampson and Dennis Kibler. Minimum generalization via reflection: a fast linear threshold learner. *Machine Learning*, 37:51–73, 1999.

[HKW98] David Haussler, Jyrki Kivinen, and Manfred K. Warmuth. Sequential prediction of individual sequences under general loss functions. *IEEE Trans. on Information Theory*, 44(5):1906–1925, 1998.

[HSSW96] David P. Helmbold, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. On-line portfolio selection using multiplicative updates. In *Proc. International Conference on Machine Learning*, pages 243–251, 1996.

[HW98] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

[IEE99] IEEE. Computer society LAN MAN standards committee. In *IEEE Std 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[Kää05] Matti Kääriäinen. Generalization error bounds using unlabeled data. In *Proc. 18th Annual Conference on Learning Theory*, 2005.

[Kää06] Matti Kääriäinen. Active learning in the non-realizable case. In *Proc. 17th International Conference on Algorithmic Learning Theory*, 2006.

[Kal01] Adam Kalai. Probabilistic and on-line methods in machine learning. PhD Thesis. In *Carnegie Mellon Computer Science Technical Report CMU-CS-01-132*, 2001.

[KB02] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proc. MobiCom 2002*, Atlanta, GA, September 2002.

[Kra02] Ronny Krashinsky. Traces of bsd simulation runs, in ns http traffic simulators, generated in personal communication with authors. MIT, 2002.

[KT81] Raphail E. Krichevsky and Victor K. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27(2):199–207, 1981.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[LeC98] Yann LeCun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/. 1998.

[LG94] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proc. of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, 1994.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

[Lon95] Philip M. Long. On the sample complexity of PAC learning halfspaces against the uniform distribution. *IEEE Transactions on Neural Networks*, 6(6):1556–1559, 1995.

[Lon03] Philip M. Long. An upper bound on the sample complexity of PAC learning halfspaces with respect to the uniform distribution. *Information Processing Letters*, 87(5):229–23, 2003.

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[MBFJ04] Claire Monteleoni, Hari Balakrishnan, Nick Feamster, and Tommi Jaakkola. Managing the 802.11 energy/performance tradeoff with machine learning. In *MIT-LCS-TR-971 Technical Report*, 2004.

[MJ03] Claire Monteleoni and Tommi Jaakkola. Online learning of non-stationary sequences. In *Advances in Neural Information Processing Systems 16*, 2003.

[MK06] Claire Monteleoni and Matti Kääriäinen. Online active learning in practice. In *submission.*, 2006.

[Mon03] Claire E. Monteleoni. Online learning of non-stationary sequences. SM Thesis. In *MIT Artificial Intelligence Technical Report 2003-011*, 2003.

[Mon06] Claire Monteleoni. Efficient algorithms for general active learning (Open problem). In *Proc. 19th Annual Conference on Learning Theory*, 2006.

[Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

[SBGM00] Tajana Simunic, Luca Benini, Peter W. Glynn, and Giovanni De Micheli. Dynamic power management for portable systems. In *Proc. ACM MOBICOM*, pages 11–19, Boston, MA, 2000.

[Ser99] Rocco A. Servedio. On PAC learning using winnow, perceptron, and a perceptron-like algorithm. In *Computational Learning Theory*, pages 296 – 307, 1999.

[SOS92] H. S. Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proc. Fifth Annual ACM Conference on Computational Learning Theory*, 1992.

[Ste02] Carl Steinbach. A reinforcement-learning approach to power management. In *AI Technical Report, M.Eng Thesis*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 2002.

[TK01] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

[Val84] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[Vov99] Vladimir Vovk. Derandomizing stochastic prediction strategies. *Machine Learning*, 35:247–282, 1999.