.

Probabilistic and On-line Methods in Machine Learning

Adam Kalai
May 16, 2001
CMU-CS-01-132

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Thesis Committee:
Avrim Blum, chair
Manuel Blum
Danny Sleator
Santosh Vempala

This research was partially supported a National Science Foundation Graduate Fellowship and an IBM Distinguished Graduate Fellowship.
The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or IBM.

.

Keywords: Algorithms, On-line Algorithms, Machine Learning

Abstract

On the surface, the three on-line machine learning problems analyzed in this thesis may seem unrelated. The first is an on-line investment strategy introduced by Tom Cover. We begin with a simple analysis that extends to the case of fixed-percentage transaction costs. We then describe an efficient implementation that runs in time polynomial in the number of stocks. The second problem is k-fold cross validation, a popular technique in machine learning for estimating the error of a learned hypothesis. We show that this is a valid technique by comparing it to the hold-out estimate. Finally, we discuss work towards a dynamically-optimal adaptive binary search tree algorithm.

To my mother, Marilyn Kalai.
May her PBSCT be as easy on her as my committee was on me.

Acknowledgments

It should be no surprise that my biggest thanks go to my parents, who somehow created me and gave me a very happy childhood. For as long as I can remember, my father has been teaching me about problem solving and research through puzzles and questions. If I end up with a fraction of his creativity and accomplishments, I will feel very lucky. Since I was a baby, I couldn't have asked for a better role model than my mother. Even if I could have talked at that age, I still wouldn't have asked for one.

I came to CMU in large part because of Avrim Blum. After three advisors, I can say with full confidence that Avrim is the best advisor and teacher at CMU. I don't think I would have finished with anyone else. They often say that, by the time you're ready to graduate, you should know your area better than your advisor. If that was a requirement, I would never graduate.

I'm moving from one great advisor to another. Next year I'll be at MIT under the supervision of Santosh Vempala. Many thanks to Santosh for being such an amazing friend and also a patient teacher.

Also a big thanks to a several professors at CMU. If my committee was a circus, then Avrim would be the ringmaster, Manuel Blum would be the Lion Tamer, Santosh would be the clown, and Danny Sleator would be the bearded lady. Steven Rudich would steal the show with his magic tricks. I will never forget what I learned from my other two advisors, Mel Siegel and Randy Pausch. I am also grateful to the staff, especially Dorothy Zaborowski, Sharon Burks, Catherine Copetas, and Jennifer Roderick.

A ginormous thanks to my first set of friends, Dougs Beeferman and Rohde, Rosa Arriaga, Santosh Vempala, Cris and Neil Heffernan, Carl Burch, Stan Chen, Jeff Woodard, Hannah Niswonger, and Philip Wickline, and to my newer friends, mandY Holbrook, Sapna Puri, Dennis Cosgrove, Chris Sturgil, and the whole Stage 3 gang. At the risk of leaving people of the list, I will leave everyone else off the list. To be safe, I would also like to thank everyone who has read my acknowledgements.

Contents

Chapter 1

Introduction

The probabilistic method often makes an otherwise difficult problem seem easy. Its cornerstone is using a probabilistic analysis for a deterministic problem. We use the probabilistic method and other such simplifying tools for three problems in machine learning. These three problems can all be viewed as variations on predicting from expert advice, a problem well studied in machine learning theory and other areas [49, 12, 62].

In the first problem, the experts are an infinite family of investment strategies. Our goal is to do almost as well as the best of these strategies. Although the algorithm we analyze is completely deterministic, by observing that its performance is the expected performance of a random expert, the analysis becomes simple. Using rapidly-mixing Markov chains, we are able to efficiently implement this strategy with a large number of stocks.

In the second problem, we analyze k-fold cross-validation, a common technique in machine learning for estimating the error a hypothesis on data. The k-fold estimate is the average of k estimates, which can be thought of as expert suggestions. Despite the fact that these estimates may be very correlated, we show that their average is strictly better than an individual.

In the third problem, we are interested in the dynamic optimality of binary search trees. We propose an algorithm that also is a weighted average of all binary search tree algorithms. Again, while the algorithm is completely deterministic, the analysis is probabilistic. This is one of the classic problems in on-line algorithms, and the first problem is the most well-studied problem in on-line portfolio selection.

1.1 On-line Analysis

On-line analysis, introduced by Sleator and Tarjan [57], provides a metric for analyzing on-line algorithms. An on-line algorithm is an adaptive algorithm, in that it receives its input incrementally. In contrast, an off-line algorithm receives all of its input at the start. For example, in caching, we have a sequence of requested pages and a cache which can hold k pages. Proceeding through the sequence, every time a page is requested and is in the cache, we simply return that page. However, every time a page is accessed and is not in the cache, we must evict one of the pages in the cache and replace it with the requested page. For an off-line algorithm, the entire sequence of requests is known in advance, and page replacement decisions can be made based on knowledge of future requests. For an on-line algorithm, the requests are revealed and must be handled one at a time.

Given a cost metric, e.g. a cost of 1 for evicting a page, minimizing the cost of an off-line algorithm is an ordinary optimization problem. Sleator and Tarjan's method of on-line analysis provides a framework for comparing on-line algorithms. An on-line algorithm is r*-competitive*[1] if, for any sequence of requests, its cost is no more than r times the off-line optimal cost, plus some constant c. Say at the completion of the sequence, an on-line algorithm has a *regret* that is the ratio of its cost to the minimum cost of an off-line algorithm. This is the amount that the algorithm regrets its choices compared to the optimal choices made with hindsight. Then the competitive ratio r is the worst-case regret, modulo the constant additive term c. Perhaps this is best illustrated with an example.

List Update Example

In the classic list update problem, we have a linked list of items, which may be thought of as folders in a filing cabinet. Each time an item is requested, we have to search linearly through the list until we find it. If the item is at position i in the list, this search costs i. In the standard model, we are allowed to reinsert the item anywhere in positions $1, 2, \ldots i$ at no cost and perform arbitrary swaps of adjacent list items at a cost of 1 per swap.

Sleator and Tarjan have shown that the move-to-front algorithm, which moves each requested item to the front of the list, costs at most a factor of 2 more than any list algorithm, for any sequence. Thus their algorithm is 2-competitive, i.e. has a competitive ratio of 2, provided that both algorithms begin with the same list order [57].

A simple way to see this[2] is to change the problem so that it is mandatory to move the requested item to the front of the list. This never increases the cost of a sequence more than a factor of two. To see this, think of a new cost function where, if the requested item is at position i, the search still costs i, but the reinsertion at position $1 \leq j \leq i$ now costs $j-1$ (because we first move the item to the front at no cost and then perform $j-1$ swaps to move it to position j). The new cost is always at least as large as the old cost, but never more than twice as large, because the reinsertion cost is no more than the search cost. Thus a r-competitive algorithm under the new cost metric will be $2r$ competitive under the old.

Next, observe that an optimal thing to do is to never perform any swaps, even for an off-line algorithm. To see this, suppose the last thing done after a request was to swap x and y. Let us be lazy and delay this swap until after the next request. If the next request is neither x nor y, then the delay has had no additional cost. If x is requested, then we have actually decreased our search cost by 1, and we no longer have to do the (x, y) swap, since x is moved to the front in either case. Similarly, if y is requested, we have increased our search cost but reduced the number of swaps, thus not changing the total cost. Since we can always delay the last swap, we can actually delay all swaps indefinitely, and move-to-front is optimal. Since the new cost model differs by a factor of two from the old cost model, move-to-front is 2-competitive. ■

On-line algorithms are adaptive and thus can be viewed as learning algorithms. For example, there are many things one might learn from a sequence of accesses, such "as after x is requested, y will always be requested before z." However, on-line analysis show that the move-to-front algorithm uses, up to a factor of two, all the valuable information one might learn from the sequence, for the list update problem.

In addition, we can speak of the competitive ratio *relative* to a restricted class of off-line algorithms. In this case, the on-line algorithms are allowed to do more than the off-line algorithms, and we have a competitive ratio bounding the performance of an on-line algorithm to best of the restricted class of off-line

[1] In general, r is a function of the problem size and the sequence length.
[2] Sleator and Tarjan use a more sophisticated *potential function* analysis.

algorithms. For example, in the list update problem we might restrict the off-line algorithm to do swaps only in positions that preceded the accessed element, while the on-line algorithm could do arbitrary swaps.

1.2 Universal Portfolios

Consider the investment strategy of dividing one's money evenly among s stocks and letting it sit. This has the on-line guarantee that, at any point in time, regardless of the market, it will have at least the performance of $1/s$ times the performance of the best stock, where performance is defined as the value of the investment for an initial one-dollar investment. Since, over time stock prices grow exponentially, this popular strategy approaches the exponential growth rate of the best stock.

On the other hand, a Constant Rebalanced Portfolio (CRP) keeps the same distribution of wealth among a set of stocks each day. For example, the $(0.4, 0.5, 0.1)$ each day rebalances so that it has 40% of its money is stock 1, 50% in stock 2, and 10% in stock 3. Thus there are infinitely many possible CRPs.

These CRPs capture the notion of "buy low, sell high." As the price of one stock increases (relative to the others), it buys more, and as the relative price decreases, it sells. How low is low and how high is high? The CRP smoothes these decisions based on a single parameter for each stock. Typically, if you could choose the best parameters for your CRP in hindsight, you would be doing exponentially better than the best stock. CRPs capitalize on the relative volatility between stocks. It may be that two stocks, over a year, lost value, but a CRP may have gained. Furthermore, like the previous strategy, it has the nice property that if all the stock prices ever return to their initial values, then it cannot have lost money.

Thomas Cover has an algorithm called the UNIVERSAL portfolio, which performs almost as well as the best CRP in hindsight [16]. The way it works is by averaging over all CRPs[3]. We present a simple probabilistic analysis of the deterministic UNIVERSAL algorithm, which relies on the fact the performance of UNIVERSAL is the expected performance of a random CRP. We argue that any CRP that is "nearby" the optimal CRP will perform near-optimally. Then, we show that a random CRP has a high probability of being nearby the optimal CRP. This analysis extends without modification to the case of a fixed percentage transaction cost. This is joint with Avrim Blum [8].

Since UNIVERSAL achieves the expected performance of a random CRP, one might instead consider investing everything in a single, random CRP. This is analogous to investing all of one's money in a random stock rather than dividing it evenly and letting it sit. People seem to diversify their portfolio, perhaps because they are risk averse. For example, they would rather be worth one million dollars for certain than a 10% chance at ten million dollars and 90% chance of being bankrupt. Of course, for small amounts of money, this risk aversion is not as important. The UNIVERSAL's performance is guaranteed, which we would argue is better than a guarantee on its expected value.

Unfortunately, all previous implementations of UNIVERSAL are exponential in the number of stocks, with run times of $O(n^{s-1})$. In joint work with Santosh Vempala [42], we present an implementation which is polynomial in the number of stocks based on random walks. For this randomized algorithm, we prove that with arbitrarily high probability, it performs arbitrarily close to UNIVERSAL.

1.3 Bounds for K-Fold Cross Validation

K-fold cross validation is a common technique for measuring the error of a hypothesis in machine learning. First, we divide the given data into k equally sized folds. The standard hold-out estimate learns a hypothesis on the first $k - 1$ folds and tests it on the last fold, an independent test set. The k-fold estimate repeats

[3] Since the set of CRPs is infinite, this is an integral or, equivalently, the limit as we make a finer and finer grid on the set of CRPs.

this, learning on $k-1$ folds and testing on the other fold, and averages the k estimates. Thus, it is similar to the above scenario, where presumably more measurements are better. We show that indeed the absolute moments of our estimate's error are smaller for k-fold than a hold-out of size $1/k$ of the data, even though the k different estimates are by no means independent.

Previous analyses of k-fold cross-validation gave weaker "sanity-check" bounds. This is partly because they were viewing the k-fold estimate as an estimate of the error of the hypothesis that is trained on all the data. Unfortunately, nothing can be guaranteed for a black-box learning algorithm in this setting, because we have not a single test of this new hypothesis. Instead, we view the k-fold estimate as an estimate of the error of the average of the k hypotheses generated. While this is not what is used in practice, it clearly identifies the difficulty with bounding the k-fold estimate's error – the difference between the k constituent hypotheses and the hypothesis trained on all the data. This is joint work with Avrim Blum and John Langford [9].

1.4 Splay Trees

Splay trees [58], discussed in Chapter 3, are adaptive binary search trees that operate in a manner similar to move-to-front. Every time a node is accessed, it is rotated to the root through a clever series of rotations (the tree analog of swaps). The dynamic optimality conjecture of splay trees, a longstanding open problem, is simply the question of whether splay trees are constant-competitive in the same way that move-to-front is, i.e. when the cost is the depth of the accessed item plus the number of rotations performed.

The technique we used in the move-to-front analysis could potentially extend to splay trees. Suppose we make it mandatory to immediately move the requested node to the root exactly as splaying does. Then, by the same reason as in move-to-front, we will increase the cost of any algorithm by at most a constant factor. Next, suppose we increase the cost of each rotation by a constant factor c, further increasing the cost by at most a factor of c. Then, one might hope to show that the lazy approach works, i.e. delaying the last rotation to the next round will not cost anything extra. Unfortunately, this is not true for splay trees, but it might be true for some variation on them, such as randomized splay trees [31, 1].

In this thesis, though, we try a slightly different approach. We consider the set of all sequences of rotations as a set of experts. Every tree is viewed as a probability distribution predicting the next request, with nodes near the root having higher probability. We then take a suitable weighted average of these probability distributions and convert it back into a tree. The good news is that the search cost, i.e. the sum of the depths of accessed elements, is no more than a constant times the total cost of any sequence of rotations. Unfortunately, the rotation cost might be higher. However, we hope that this is a step towards finding a dynamically optimal binary search tree algorithm. This is joint work with Avrim Blum and Santosh Vempala.

Chapter 2

Universal Portfolios

Thomas Cover's UNIVERSAL portfolio [16] is an on-line investment strategy with assumption-free theoretical guarantees. In this chapter, we present a simple analysis of Cover's investment strategy and answer two open questions about his portfolios. First, our analysis shows that the UNIVERSAL properties hold in the case of fixed percentage transaction costs. This analysis was joint work with Avrim Blum [8]. Second, we present a polynomial-time implementation of UNIVERSAL. That is, each day UNIVERSAL specifies how much money to put in each stock. All previous methods of doing this calculation took time exponential in the number of stocks. We present an efficient implementation based on random walks. This is joint work with Santosh Vempala [42].

2.1 Constant Rebalanced Portfolios

The class of investment strategies we are interested in here is the set of constant rebalanced portfolios. A constant rebalanced portfolio (CRP) keeps the same distribution of wealth among a set of stocks from day to day. For example, the $\langle .5, .5\rangle$ CRP shown in Figure 2.1a keeps an equal amount of money in two stocks. At the beginning of each day, it rebalances its holdings to maintain this equality. In this fictitious market, the price of Stock 1 alternately doubles and halves, while the price of the Stock 2 stays the same. When the first stock doubles, the $\langle .5, .5\rangle$ CRP gains 50% because half of its money is in this stock. When the first stock halves, the $\langle .5, .5\rangle$ CRP only loses 25%. So, every two days its value increases by a factor of $1.5 * 0.75 = 1.125$. This is exponential growth at a rate of 12.5%, despite the fact that both stocks stay within a factor of 2 of their original price.

To some extent, a CRP captures the notion of "buy low, sell high." As the price of a stock drops relative to others, the CRP buys more of it. As the relative price of a stock increases, the CRP sells. Of course, the $\langle .5, .5\rangle$ CRP does not always do well. In Figure 2.1b, Stock 1 is constant, but the price of Stock 2 halves for twenty days and then doubles for twenty days. During the first twenty days, as Stock 2 drops, the CRP buys more. Had Stock 2's price never rebounded, this would have been the wrong thing to do. On this first half, the best thing would be to keep all of one's money in Stock 1, which is exactly what a $\langle 1, 0\rangle$ CRP would do. Over the whole time, though, a simple calculation shows that the best CRP to choose is the $\langle .5, .5\rangle$ CRP.

Another example may illustrate why it is valuable to know the best parameters for a CRP. Suppose that 75% of the time the first stock's price doubled and 25% it did very poorly, being reduced by a factor of

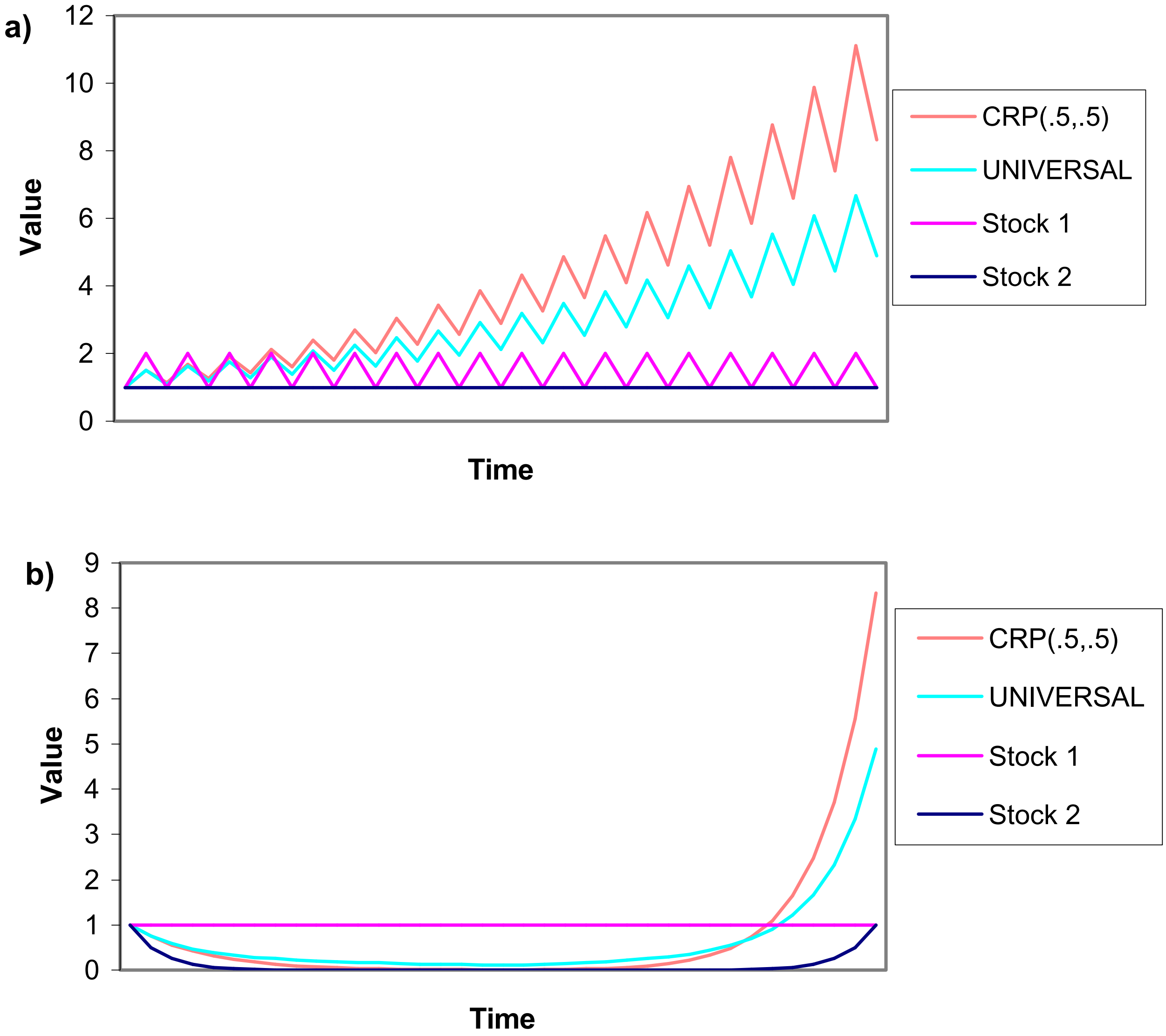

Figure 2.1: Fictitious markets: a) Stock 1 alternately halves and doubles, Stock 2 is constant b) Stock 1 is constant, Stock 2 drops exponentially but then returns

ϵ. On the days where Stock 1 did poorly, suppose Stock 2's price doubled, and on the other days its price dropped by ϵ. Then the wealth of a CRP that keeps an x fraction of its wealth in Stock 1 after n days is roughly $(2x)^{0.75n}(2(1-x))^{0.25n}$, for small ϵ. The $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$ CRP essentially loose all their money. The $\langle .5, .5\rangle$ CRP breaks even. However, the $\langle .75, .25\rangle$ CRP makes exponential money, and thus so does the UNIVERSAL. Moreover, the above function has a very sharp peak at $x = 0.75$. Thus it would be very valuable to know the best parameter to use in hindsight.

Of course, there may be more than just two stocks. In general a CRP has one parameter for each stock, reflecting the fraction of its wealth to keep in that stock. It has an additional parameter which is how often to rebalance. In this thesis, we will assume daily rebalancing, but due to various factors, such as volatility or transaction costs, one might want to rebalance more or less often.

Notation and Definitions

Let s be the number of stocks in the market. Let CRP$\langle v\rangle$ be the CRP which keeps a v_i fraction of its money invested in stock $1 \le i \le s$, for vector $\langle v\rangle \in \Delta$. Here Δ is the simplex,

$$\Delta = \{\langle v\rangle \in \Re^s | v_i \ge 0, \sum_1^s v_i = 1\}$$

Let x_i^d be the ratio of the closing price to opening price of stock i on day d. Then the *performance* of stock i over n days, the value of an initial one-dollar investment, is $P_n(i) = \prod_{d=1}^n x_i^d$. With slight abuse of notation, the performance of CRP$\langle v\rangle$ is,

$$P_n\langle v\rangle = \prod_{d=1}^{n}\sum_{i=1}^{s} v_i x_i^d = \prod_{d=1}^{n}\langle v\rangle \cdot \langle x^d\rangle$$

The performance of the UNIVERSAL algorithm will be written $P_n(\text{UNIVERSAL})$. Finally, the *log-performance* of an investment is simply the logarithm of its performance. The log-performance indicates how many digits are in the value of your holdings.

Log-concavity of CRP's

One of the striking things about our examples is that, even though neither stock is never significantly higher than its original price, we've made exponential amounts of money. In fact, one can say that if the prices of the stocks haven't changed, then any CRP can only have made money. This is an easy corollary of the log-concavity of CRP's.

Lemma 2.1 *The log-performance of CRP$\langle v\rangle$ is a concave function of $\langle v\rangle$, i.e., for CRPs $\langle v\rangle, \langle w\rangle$,*

$$\log P_n\left(\frac{\langle v\rangle + \langle w\rangle}{2}\right) \ge \frac{\log P_n\langle v\rangle + \log P_n\langle w\rangle}{2}$$

Proof. This follows directly from the concavity of the $\log$ function.

$$\begin{aligned}\log P_n\left(\frac{\langle v\rangle + \langle w\rangle}{2}\right) &= \sum_{d=1}^{n}\log\frac{\langle v\rangle + \langle w\rangle}{2}\cdot\langle x^d\rangle \\ &\ge \sum_{d=1}^{n}\frac{\log\langle v\rangle\cdot\langle x^d\rangle + \log\langle w\rangle\cdot\langle x^d\rangle}{2}\end{aligned}$$

And this last expression is the right-hand side of the lemma. ∎

We can then say that the log-performance of a CRP will be at least the weighted average of the log-performances of the constituent stocks.

Corollary 2.2 $\log P_n\langle v\rangle \geq \sum_{i=1}^{s} v_i \log P_n(i)$.

In particular, if the price of every stock goes up (or down) by a constant factor c over an arbitrary number of days, then any CRP's performance will change by a factor of at least c.

It is interesting to note that in the continuous case (the limit as we rebalance more and more often) the above becomes an equality. Thus more frequent rebalancing may not necessarily be good. For example, with continuous rebalancing, a CRP can never do better than the best stock.

2.2 The UNIVERSAL Portfolio

It would be very valuable to know which stock is going to perform the best over the next twenty years. An investor can almost achieve this growth by splitting their money among several stocks. If they divide their money evenly between s stocks, their performance will always be at least $1/s$ times the performance of the best of these stocks, and usually better. Of course, this is a common strategy.

It would be much more valuable to know which CRP is going to perform best, because the best CRP often performs exponentially better than the best stock, as we have seen from our examples. In particular, the best CRP performs at least as well as the best stock, because the strategy of investing everything in one stock is a CRP. Cover uses the same splitting idea, but among the set of CRPs, to asymptotically match the growth rate of the best CRP. To be precise, let $\langle\text{OPT}\rangle$ be an optimal CRP, chosen in hindsight. Then, as in [17],

Theorem 2.3 *For All markets with s stocks and n days,*

$$\begin{aligned}\frac{P_n(\mathit{UNIVERSAL})}{P_n\langle \mathit{OPT}\rangle} &\geq \binom{n+s-1}{s-1}^{-1}\\ &\geq \frac{1}{(n+1)^{s-1}}\end{aligned}$$

Equivalently,

$$\frac{\log P_n(\mathit{UNIVERSAL})}{n} \geq \frac{\log P_n\langle \mathit{OPT}\rangle}{n} - \frac{(s-1)\log n + 1}{n}$$

Looking at the daily log-performance above, we see that difference in daily log-performance goes to zero quickly. The term UNIVERSAL, like universal data compression, indicates this on-line asymptotic optimality relative to a class of investment strategies. In our first example, the best CRP is in fact the $\langle .5, .5\rangle$ CRP, which has performance of $(1.125)^{n/2}$. Thus, the above guarantees UNIVERSAL a performance at least $(1.125)^{n/2}/n$, but it performs even better. Alternatively, this may be seen as a competitive ratio of $\frac{1}{(n+1)^{s-1}}$ relative to the class of CRPs.

UNIVERSAL can be thought of as the simple split algorithm, but instead of dividing its money evenly among a set of stocks, it divides it evenly among all CRPs, and does not transfer between them. Since the the set of CRPs is infinite, this can be thought of as the limit, as we make a finer and finer grid on the simplex. Figure 2.2 illustrates this for three stocks. The set of possible CRPs is a simplex. Each sample is a lock box, invested according to the corresponding CRP, which starts with an even fraction of the total wealth. Money is never transferred between these lockboxes. Over time, some of the lockboxes grow and others shrink, and thus UNIVERSAL's distribution of wealth moves towards the optimal CRP. To be precise,

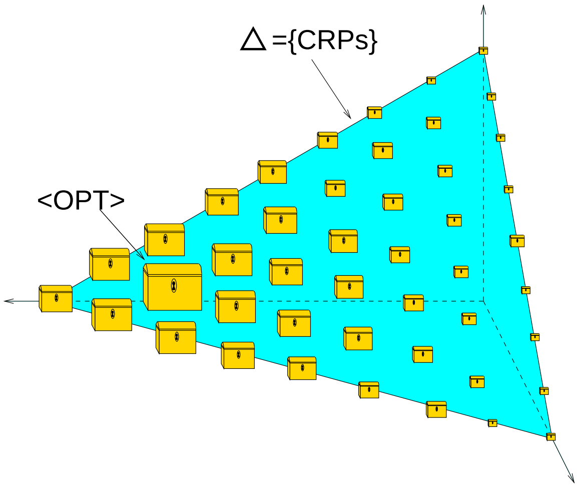

Figure 2.2: Initially, the money is divided equally among a set of evenly-spaced lockboxes, each representing a CRP. Over time, lockboxes change in size. The limit as the number of lockboxes increases without bound is the UNIVERSAL portfolio.

Definition 2.4 *(UNIVERSAL) On day n, UNIVERSAL has a u_i^n fraction of its wealth in stock i, where*

$$\langle u^n \rangle = \frac{\int_\Delta \langle v \rangle P_{n-1}\langle v \rangle d\mu\langle v \rangle}{\int_\Delta P_{n-1}\langle v \rangle d\mu\langle v \rangle}$$

Here, μ is the uniform distribution over Δ, the simplex of possible CRPs.

This is the form in which Cover defines the algorithm for the uniform distribution[1]. He also notes ([17]) that

$$P_n(\text{UNIVERSAL}) = \mathrm{E}_{\langle v \rangle \in \Delta}\left[P_n\langle v \rangle\right] \tag{2.1}$$

In other words, since UNIVERSAL is just an average over all CRPs, its performance is exactly the expected performance of a random CRP.

Simple Analysis

We will first show a very simple analysis that gives a slightly weaker bound than the theorem. The key to our analysis is to consider *nearby* CRPs. There is some CRP that is optimal in hindsight, say $\langle \text{OPT} \rangle$. There is a high probability that a random portfolio is nearby $\langle \text{OPT} \rangle$, and nearby portfolios perform nearly as well. To be precise, let us say $\langle v \rangle$ is nearby $\langle \text{OPT} \rangle$ if $v_i \geq (1-\alpha)\text{OPT}_i$, for all i. For simplicity, let us first say $\alpha = 1/(n+1)$.

Then the performance of CRP$\langle v \rangle$ on any given day will be at least $1-\alpha$ times the performance of CRP$\langle \text{OPT} \rangle$, because at least a $1-\alpha$ fraction of CRP$\langle v \rangle$'s wealth is distributed exactly like CRP$\langle \text{OPT} \rangle$.

[1] Cover has shown the better bound of $1/(n+1)^{\frac{s-1}{2}}$ for the Dirichelet$(1/2, \ldots, 1/2)$ ditribution, which has density proportional to $1/\sqrt{\prod v_j}$ on portfolio $\langle v \rangle$. Our analysis does not apply to this distribution precisely because it is not uniform, and our efficient algorithm does not work because it is not log-concave.

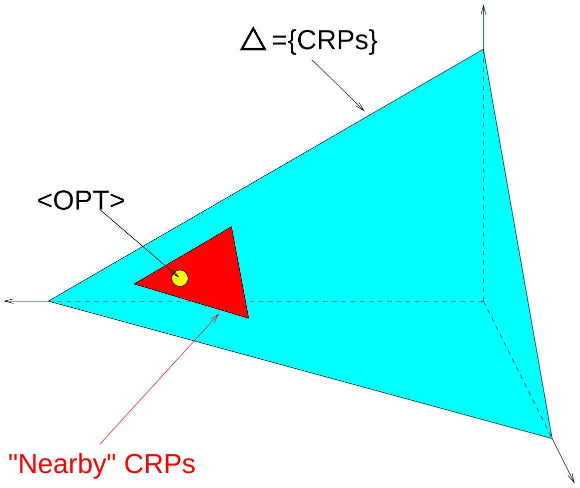

Figure 2.3: There are many CRPs "nearby" the optimal CRP, and these portfolios perform nearly as well.

Over n days, we get,

$$\begin{aligned} \frac{P_n\langle v\rangle}{P_n\langle \text{OPT}\rangle} &\geq (1-\alpha)^n \qquad (2.2) \\ &\geq \frac{1}{e} \left(\text{for } \alpha = \frac{1}{n+1}\right) \end{aligned}$$

This is the sense in which nearby portfolios do nearly as well.

How many such nearby portfolios are there? This is easy to compute because the set of nearby portfolios is a simplex, translated from the origin to $(1-\alpha)\langle\text{OPT}\rangle$ and shrunken by a factor of α. The simplex has dimension $s-1$.

$$\begin{aligned} \text{Vol}\{\langle v\rangle \in \Delta | \langle v\rangle \text{ is nearby } \langle\text{OPT}\rangle\} &= \text{Vol}\{(1-\alpha)\langle\text{OPT}\rangle + \alpha\langle w\rangle | \langle w\rangle \in \Delta\} \\ &= \text{Vol}\{\alpha\langle w\rangle | \langle w\rangle \in \Delta\} \\ &= \alpha^{s-1} Vol\Delta \qquad (2.3) \end{aligned}$$

Because UNIVERSAL does as well as a random CRP (2.1), nearby portfolios perform at least $1/e$ times as well (2.2), and there are $1/(n+1)^{s-1}$ portfolios near the best CRP (2.3),

$$\frac{P_n(\text{UNIVERSAL})}{P_n\langle\text{OPT}\rangle} \geq \frac{1}{e}\frac{1}{(n+1)^{s-1}}$$

Notice that this simple analysis is slightly worse than the guarantee of Theorem 2.3.

To get the better bound, we consider α-nearby portfolios. We say $\langle w\rangle$ is α-*nearby* $\langle v\rangle$ if $w_i \geq (1-\alpha)v_i$, for all i. We think of α as a function of $\langle v\rangle$, i.e. a random variable defined by $1-\alpha = \min_i v_i/\text{OPT}_i$. Then, from equations (2.1) and (2.2) we see that,

$$\begin{aligned} \frac{P_n(\text{UNIVERSAL})}{P_n\langle\text{OPT}\rangle} &\geq E_{\langle v\rangle\in\Delta}\left[(1-\alpha)^n\right] \\ &= \int_0^1 \text{Prob}_{\langle v\rangle\in\Delta}\left[(1-\alpha)^n \geq x\right] dx \qquad (2.4) \end{aligned}$$

The last equality is an identity for random variables in $[0,1]$.

By (2.3), the probability in the above integral is exactly $(1 - x^{1/n})^{s-1}$. Making the change of variable $y = x^{1/n}$, this yields,

$$\begin{aligned} \int_0^1 (1 - x^{1/n})^{s-1} dx &= \\ n \int_0^1 y^{n-1}(1-y)^{s-1} dy &= \\ \frac{1}{\left(\begin{array}{c} n+s-1 \\ s-1 \end{array} \right)} & \end{aligned}$$

The last equality can be found by integration by parts [2].■

2.3 Transaction Costs

In this section, we describe a simpler way to incorporate transaction costs than in [8], which also gives better guarantees. The model of transaction costs is still a fixed percentage commission $c < 1$ on transactions, as is common in financial modeling[3] [21].

The tricky part is defining a CRP in the presence of commission. We use the following *naive model* for how a CRP$\langle v \rangle$ rebalances. At the beginning of the day it has some distribution of wealth $\langle v' \rangle$. Without commission, it would simply sell $v_i' - v_i$ of every stock for which $v_i' > v_i$ and then buy $v_i - v_i'$ of every stock for which $v_i' < v_i$.

Definition 2.5 *(naive model of rebalancing) In the presence of commission c, the CRP$\langle v \rangle$ with a distribution of wealth of $\langle v' \rangle$ rebalances by selling $v_i' - v_i$ of every stock for which $v_i' > v_i$ and buying $(1-c)(v_i - v_i')$ of every stock for which $v_i' < v_i$.*

Thus, if an investor has M dollars distributed according to $\langle v' \rangle$, then after rebalancing she will have Mv_i dollars worth of every stock she sold and $(1-c)Mv_i + cMv_i'$ dollars worth of every stock she bought. We call this naive because it is as if the CRP investor is surprised every time to find out that there is commission (and that it is not perfectly rebalanced) but does nothing about it.

The previous *sophisticated model* of how a CRP$\langle v \rangle$ rebalances [8] is much harder to implement. In that model, the CRP chooses the minimal set of transactions to perform so that it is rebalanced exactly according $\langle v \rangle$. This is a logical, natural definition that, in some ways, better fits the term CRP. However, it requires solving an optimization problem each day just to rebalance. Furthermore, it lacks some properties of the naive model that we believe are nice. For example, with naive rebalancing, a $\langle .5, .5 \rangle$ CRP always does at least $(.5)^n$ times as well as the better of the two stocks, over n days. The sophisticated model would do

[2] Alternatively, the last equality can be verified by the following puzzle. Suppose we pick $n+s-1$ numbers in $[0,1]$ uniformly at random. What is the probability that each of the first n numbers will be smaller than the each of the last $s-1$ numbers? On the one hand, it is equally likely that any given subset of size $s-1$ will be the largest $s-1$ numbers selected, so the answer is $\left(\begin{array}{c} n+s-1 \\ s-1 \end{array} \right)^{-1}$. On the other hand, we can break it down into cases, based on the largest number among the first n. The probability that, say, the i^{th} number is greater than or equal to the first n and smaller than the last $s-1$ (for $i \le n$) is $\int_0^1 (1-y)^{n-1} y^{m-1} dy$. Thus the answer can also be written as n times this integral, and we have equality.

[3] This is as general as a cost $c_s < 1$ on sales and $c_b < 1$ on purchases. To see this, imagine transferring x dollars worth of one stock to another and paying for commissions with the sales. This would give $x(1-c_s)/(1+c_b)$ dollars worth of the new stock, which is equivalent to a commission of $1 - c = (1-c_x)/(1+c_b)$ or $c = (c_s + c_b)/(1+c_b) < 1$ charged on transactions.

worse than $(.5)^n$ if one stock's price stayed the same and the other dropped a lot every day. In essence, with the sophisticated model, the failure of one stock affects another stock by a factor larger than $.5$. One would be better off each day giving half of its stock to charity.

With the sophisticated model, it was shown that the bound in Theorem (2.3) still holds if we replace n with $(1+c)n$. In this section, we will show that with naive model, the bound and analysis work exactly as is. It is worth pointing out that the two models are actually very similar, differing in only a second order term. Even though the naive rebalancer may not have exactly the correct distribution of wealth each day, it will be off by at most a $(1-c)$ factor. Furthermore, if prices do not change for several days, it will approach the correct distribution exponentially quickly. Likewise, the sophisticated model does have the property that a $\langle .5, .5\rangle$ CRP will do at least $(.5(1-c))^n$ as well as the better of the two stocks.

Lemma 2.6 *If $\langle w\rangle$ is α-nearby $\langle v\rangle$, then in the naive model of rebalancing,*

$$P_n\langle w\rangle \geq (1-\alpha)^n P_n\langle v\rangle$$

Proof. This intuitive statement is a little messy to prove (and not true in the sophisticated model). We prove by induction on n that the amount of money in each stock for CRP$\langle w\rangle$ is at least $(1-\alpha)^n$ times the amount of money for CRP$\langle v\rangle$. Suppose it is true on day n. At the end of day n, say we have M_v and M_w dollars distributed according to $\langle v'\rangle$ and $\langle w'\rangle$, in our respective CRPs. Then, since stock i has changed by the same amount, x_i^d, in both portfolios, we will have,

$$\begin{aligned} M_w w_i' &\geq (1-\alpha)^n M_v v_i' && (2.5)\\ M_w &\geq (1-\alpha)^n M_v && (2.6)\\ M_w w_i &\geq (1-\alpha)^{n+1} M_v v_i && (2.7)\end{aligned}$$

After rebalancing, there are four cases depending on whether each CRP sells or buys. We need to show that the amount of money in every stock for $\langle w\rangle$ is at least $(1-\alpha)^{n+1}$ times the amount of money for $\langle v\rangle$. If both CRPs sell stock i ($v_i' > v_i$ and $w_i' > w_i$), then after rebalancing they have $M_v v_i$ and $M_w w_i$ worth of stock i, respectively. This satisfies the induction hypothesis by (2.7). If both CRPs buy stock i, then they will have $(1-c)M_v v_i + cM_v v_i'$ and $(1-c)M_w w_i + cM_w w_i'$ worth of stock i, which is good good by (2.7) and (2.5). If $\langle v\rangle$ buys and $\langle w\rangle$ sells, then we are okay because $v_i' < v_i$, and thus $(1-c)M_v v_i + cM_v v_i' < M_v v_i \leq (1-\alpha)^{n+1} M_w w_i$. Finally, if $\langle v\rangle$ sells and $\langle w\rangle$ buys, then we are okay because $(1-c)M_w w_i + cM_w w_i' \geq M_w w_i' \geq (1-\alpha)^n M_v v_i' \geq (1-\alpha)^n M_v v_i$.■

Finally, we define UNIVERSAL just as before as the average of all CRPs. We must explain how UNIVERSAL performs its transactions each day, in the presence of commission. Each constituent CRP is issuing orders such as "buy stock i" or "sell stock j." In accumulating all of these orders (an integral itself), we will most likely have some offsetting transactions, where one portfolio is buying stock i and another is selling it. Instead of performing the wasteful transactions, we donate the stock that was going to be used for transaction costs to charity. Thus (2.1) still holds. The above lemma is exactly (2.2), and so the analysis still holds.

Corollary 2.7 *With naive rebalancing of CRPs, the UNIVERSAL guarantees of Theorem 2.3 hold in the presence of arbitrary commission $c < 1$ charged on all transactions.*

2.4 Prediction From Expert Advice

The problem of prediction from expert advice is well-studied in machine learning theory and other areas [49, 12, 22, 27, 35, 46, 62, 63]. There are many variations on this problem. Here, we'll discuss two related to the stock market. First, suppose meteorologists were paid in a logical manner. For example, when the predict the chance of rain to be $0 \leq p \leq 1$, say their reward is $\log 2 + \ln p$ if it rains and $\log 2 + \log 1 - p$ if it doesn't. One benefit of this system is that if the meteorologist believes the true chance of rain is p, then to maximize her expected reward, she should predict rain with probability p[4].

Now, say there we have the opportunity to be a meteorologist, we know nothing about predicting rain, but we have access to s other meteorologists' predictions each day. Next, pretend that we have a stock for each meteorologist that decreases in price by a factor of p_i when it rains and $1 - p_i$ when it doesn't. Then the total reward of meteorologist i over n days will be $n \ln 2 + \log \text{perf}_n(i)$.

Imagine investing $1/s$ in each of the meteorologists' stocks, and letting it sit. A good way for us to predict rain is to take a weighted average of the experts' predictions each day, with weights equal to our distribution of wealth. If we do this and our investment goes down by a factor of $x(n)$ on day n, then our reward on that day will be $\log 2 + \log x(n)$. Since our investment value is the average performance of the stocks, we see that $n \log 2 + \sum \log x(n) \geq n \log 2 + \log(\sum \text{perf}_n(i)/s)$. In particular, our reward will be at least the reward of the best meteorologist minus $\log s$, regardless of the number of days. This is quite good because it allows us to combine the opinions of a very large number of meteorologists with little cost.

A second problem is that of predicting the average temperature. For simplicity, say predictions are in $[0, 1]$ and a meteorologist predicting $\hat{t}$ is rewarded by $1 - |\hat{t} - \bar{t}|$ if $\bar{t}$ was the average temperature. What we do here is in the style of the weighted majority algorithm [49] but differs slightly. For each expert, we again have a stock. If on day n, expert i gets a reward $0 \leq r_i^n \leq 1$, then stock i goes up by a factor of $1 + r_i^n$.

Now again, imagine investing $1/s$ in each stock and letting it sit. Suppose we make our prediction according to a weighted average of the s expert predictions, with weights distributed according to the fraction of wealth we have in each stock on that day. Now, say our investment went up by a factor of $1 + x(n)$ on day n. Then, the weighted average of the experts' rewards on that day would be x_n. By the convexity of $|x|$, our reward must be at least x_n. Since our investment's value is always at least $1/s$ times the value of any stock,

$$\prod_n 1 + x(n) \geq \frac{\prod_n 1 + r_i^n}{s}$$

Taking logs of both sides and using the fact that, for $0 \leq x \leq 1$, $x \log 2 \leq \log 1 + x \leq x$, we get,

$$\sum_n x(n) \geq \sum_n r_i^n \log 2 - \log s$$

Since our total reward is at least $\sum x(n)$, we are doing within a constant factor of the best expert minus a term logarithmic in the number of experts. If we repeat the above analysis but change the stock price by $1 + \epsilon r_i^n$, then we get,

$$\text{our reward} \geq (1 - \epsilon)\text{best expert's reward} - \frac{\log s}{\epsilon}$$

2.5 Efficient Implementation

In this section, we discuss implementing UNIVERSAL without transaction costs. For such an implementation, one must specify, on each day, how much money to keep in each stock. All previous implementations of Cover's algorithm are exponential in the number of stocks with run times of $O(n^{s-1})$. A previous suggestion was a randomized approximation based on sampling portfolios from the uniform distribution [8]. However, in the worst case, to have a high probability of performing almost as well as UNIVERSAL, they require

[4] If the reward were say p in the case of rain and $1 - p$ otherwise, then she would have incentive to predict rain with probability $p = 0$ or $p = 1$.

$O(n^{s-1})$ samples. We show that by sampling portfolios from a non-uniform distribution, only polynomially many samples are required to have a high probability of performing nearly as well as UNIVERSAL. This non-uniform sampling can be achieved by random walks on the simplex of portfolios.

Since UNIVERSAL is really just an average of CRP's, it is natural to approximate the portfolio by sampling [8]. However, with uniform sampling, one needs $O(n^{s-1})$ samples in order to have a high probability of performing as well as UNIVERSAL, which is still exponential in the number of stocks. Intuitively, this is because one needs $O(n^{s-1})$ samples to have a high probability of having one nearby $\langle$OPT$\rangle$. Here we show that, with non-uniform sampling, we can approximate the portfolio efficiently. With high probability $(1-\delta)$, we can achieve performance of at least $(1-\epsilon)$ times the performance of UNIVERSAL. The algorithm is polynomial in $1/\epsilon$, $\log(1/\delta)$, s, and n.

The key to our algorithm is sampling according to a biased distribution. Instead of sampling according to μ, the uniform distribution on Δ, we sample according to ν_n, which weights portfolios in proportion to their performance, i.e.,

$$d\nu_n\langle v\rangle = \frac{P_n\langle v\rangle}{\int_\Delta P_n\langle w\rangle d\mu\langle w\rangle}$$

We will later show how to efficiently sample from this biased distribution.

Based on the above and (2.1) UNIVERSAL can be thought of as computing each component of the portfolio by taking the expectation of draws from ν_n, i.e.,

$$u_i^n = \int_\Delta v_i d\nu_n\langle v\rangle = \mathrm{E}_{\langle v\rangle\in\nu_n}[v_i] \tag{2.8}$$

Thus our sampling implementation of UNIVERSAL averages draws from ν_n:

Definition 2.8 *(***Universal biased sampler***) The Universal biased sampler, with m samples, on the end of day n chooses a portfolio $\langle a^n\rangle$ as the average of m portfolios drawn independently from ν_n.*

Now, we apply Chernoff bounds to show that with high probability, for each i, a_i^n closely approximates u_i^n. In order to ensure that this biased sampling will get us a_i^n/u_i^n close to 1, we need to ensure that u_i^n isn't too small:

Lemma 2.9 *For all $1 \le i \le s$ and $n \ge 1$, $u_i^n \ge 1/(n+s)$.*

Proof. WLOG $i = 1$. Then, u_1^n is a random variable between 0 and 1, so by (2.8) and identity (2.4),

$$u_1^n = \mathrm{E}_{\langle v\rangle\in\nu_n}[v_1] = \int_0^1 \nu_n\left(\{\langle v\rangle \in \Delta | v_1 \ge z\}\right) dz.$$

Now we know from (2.3) that the volume of $\{\langle v\rangle \in \Delta | v_1 \ge z\}$ is $(1-z)^{s-1}$ times the volume of Δ. Furthermore, the average performance of portfolios in this set is at least $(1-z)^n$ times the average over Δ, because for each of n days, a portfolio in this set $\langle v\rangle = (z, 0, \ldots, 0) + (1-z)\langle w\rangle$ performs at least $(1-z)$ as well as the corresponding portfolio $\langle w\rangle \in \Delta$. So the probability of this set under ν_n is at least $(1-z)^{s-1}(1-z)^n$ and,

$$u_1^n \ge \int_0^1 (1-z)^{s-1}(1-z)^n dz = 1/(n+s).$$

∎

Combining this lemma with Chernoff bounds, we get:

Theorem 2.10 *With $m = 2n^2(s+n)\log(sn/\delta)/\epsilon^2$ samples, the Universal biased sampler performs at least $(1-\epsilon)$ as well as Universal, with probability at least $1-\delta$.*

Proof. Say each u_i^d is approximated by a_i^d. Furthermore, suppose each $a_i^d \geq u_i^d(1-\epsilon/n)$. Then, on any individual day, the performance of the $\langle a^d \rangle$ is at least $(1-\epsilon/n)$ times as good as the performance of $\langle u^d \rangle$. Thus, over n days, our approximation's performance must be at least $(1-\epsilon/n)^n \geq 1-\epsilon$ times the performance of UNIVERSAL.

The multiplicative Chernoff bound for approximating a random variable $0 \leq X \leq 1$, with mean $\bar{X}$, by the sum S of m independent draws is,

$$\Pr\left[S < (1-\alpha)\bar{X}m\right] \leq e^{-m\bar{X}\alpha^2/2}.$$

In our case, we are approximating each u_i^d by m samples, our lemma shows that the expectation of $u_i^d = X$ is $\bar{X} \geq 1/(d+s) \geq 1/(n+s)$, and we want to be within $\alpha = \epsilon/n$. Since this must hold for ns different u_i^d's, it suffices for,

$$e^{-m\epsilon^2/(2n^2(s+n))} \leq \frac{\delta}{ns},$$

which holds for the number of samples m chosen in the theorem. ■

The biased sampler will actually sample from a distribution that is close to ν_n, call it p_n, with the property that

$$\int_\Delta |\nu_n\langle v\rangle - p_n\langle v\rangle| d\mu\langle v\rangle \leq \epsilon_0$$

for any desired $\epsilon_0 > 0$ in time proportional to $\log\frac{1}{\epsilon_0}$. Since $u_i^n \geq 1/(s+n)$, we see that the mean of the p_n distribution must be $(n+s)\epsilon_0$-nearby the mean of ν_n, which is u_i^n. Thus, if we choose $\epsilon_0 < \frac{1}{kn(n+s)}$, then we will incur a performance loss of at most $(1-\frac{1}{kn})^n \approx e^{-k}$ over n days, and our run-time is only logarithmic in $\frac{1}{\epsilon_0}$.

The biased sampler

In this section we describe a random walk for sampling from the simplex with probability density proportional to

$$f\langle v\rangle = P_n\langle v\rangle$$

Before we do this, note that sampling from the uniform distribution over the simplex is easy: pick $s-1$ reals $x_1,\ldots,x_{s-1}$ uniformly at random between 0 and 1 and sort them into $y_1 \leq \ldots \leq y_{s-1} \leq 1$; then the vector $(y_1, y_2-y_1, \ldots, y_{s-1}-y_{s-2}, 1-y_{s-1})$ is uniformly distributed on the simplex.

There is another (less efficient) way. Start at some point x in the simplex. Pick a random point y within a small distance δ of x. If y is also in the simplex, then move to y; if it is not, then try again. The *stationary* distribution of a random walk is the distribution on the points attained as the number of steps tends to infinity. Since this random walk is *symmetric*, i.e. the probability of going from x to y is equal to the probability of going from y to x, the distribution of the point reached after t steps tends to the uniform distribution. In fact, in a polynomial number of steps, one will reach a point whose distribution is nearly uniform on the simplex.

The symmetric random walk described above can be modified to have any desired target distribution. This is called the Metropolis filter [52], and can be viewed as a combination of the walk with rejection sampling: If the walk is at x and chooses the point y as its next step, then move to y with probability $\min(1,\frac{f(y)}{f(x)})$ and do nothing with the remaining probability (i.e. try again). The difficult question, though, is whether or not the random walk is rapidly mixing, i.e. it attains a distribution close to the stationary one in polynomial time. The answer is that it depends on the function f. Lovasz and Simonovits [50] have shown that for a large class of log-concave f, the walk is rapidly mixing. The important parameters are how quickly f changes and how much of the probability of f is on the border of the set.

The simplest form of the random walk would be to discretize the simplex by considering all points on the simplex whose coordinates are of the form m/N, where m is any integer between 0 and N, and N is a large fixed integer. Then a neighbor would be chosen by picking two coordinates and adding to one $1/N$ while subtracting from the other $1/N$. However, we would like to use previous analysis to show that the walk is rapidly mixing, rather than starting from scratch.

For the purposes of analysis, we consider the following random walk. First rotate Δ so that it is on the plane $x = 0$ and scale it by a factor of $1/\sqrt{2}$ so that it has unit diameter. We will only walk on the set of points in Δ whose coordinates are multiples of a fixed parameter $\delta > 0$ (to be chosen below), i.e. points on an axis parallel grid whose "unit" length is δ. Any point on this grid has $2n$ neighbors, 2 along each axis.

1. Start at a (uniformly) random grid point in the simplex.
2. Suppose $X(\tau)$ is the location of the walk at time τ.
3. Let y be a random neighbor of $X(\tau)$.
4. If y is in Δ, then move to it, i.e. set $X(\tau+1) = y$ with probability $p = \min(1, \frac{f(y)}{f(x)})$, and stay put with probability $1-p$ (i.e. $X(\tau+1) = X(\tau)$).

Let the set of grid points be denoted by D. We will actually only sample from the set of grid points in Δ that are not too close to the boundary, namely, each coordinate x_i is at least $\frac{\epsilon}{s+n}$ for a small enough ϵ. For convenience we will assume that each coordinate is at least $\frac{1}{2(s+n)}$. Let this set of grid points be denoted by D. Each grid point x can be associated with a unique axis-parallel cube of length δ centered at x. Call this cube $C(x)$. The step length δ is chosen so that for any grid point x, $f(x)$ is close to $f(y)$ for any $y \in C(x)$.

Lemma 2.11 *If we choose $\delta < \frac{\log 1+\alpha}{2(s+n)n}$ then for any grid point z in D, and any point $y \in C(z)$, we have*

$$(1+\alpha)^{-1} f(z) \le f(y) \le (1+\alpha) f(z).$$

Proof. Since $y \in C(z)$, $\max_j |y_j - z_j| \le \delta$. For any price relative x^*, the ratio $\frac{y \cdot x^*}{z \cdot x^*}$ is at most $\max_j \frac{y_j}{z_j}$. This can be written as

$$\max_j \frac{z_j + (y_j - z_j)}{z_j} = \max_j (1 + \frac{\delta}{z_j})$$

Since each coordinate is at least $\frac{1}{2(s+n)}$ we have that the ratio is at most $(1 + 2\delta(s+n))$. Thus the ratio $\frac{f(y)}{f(z)}$ is at most $(1+2\delta(s+n))^n$ and the lemma follows. ∎

The stationary distribution π of the random walk will be proportional to $f(z)$ for each grid point z. Thus when viewed as a distribution on the simplex, for any point y in the simplex,

$$\pi(y)(1+\alpha)^{-1} \le d\nu_n(y) \le \pi(y)(1+\alpha)$$

The main issue is how fast the random walk approaches π. We return to the discrete distribution on the grid points. Let the distribution attained by the random walk after τ steps be p_τ, i.e. $p_\tau(x)$ is the probability that the walk is at the grid point x after τ steps. The progress of the random walk can be measured as the distance between its current distribution p_τ and the stationary distribution as follows:

$$||p_\tau - \pi|| = \sum_{x \in D} |p_\tau(x) - \pi(x)|$$

In [29], Frieze and Kannan derive a bound on the convergence of this random walk which can be used to derive the following bound for our situation.

Theorem 2.12 *After τ steps of the random walk,*

$$||p_\tau - \pi||^2 \leq e^{-\frac{\gamma\tau}{sn^2(s+n)^2}}(s+n)^2$$

where $\gamma > 0$ is an absolute constant.

Corollary 2.13 *For any $\epsilon_0 > 0$, after $O(sn^2(s+n)^2 \log \frac{s+n}{\epsilon_0})$ steps,*

$$||p_\tau - \pi||^2 \leq \epsilon_0.$$

Proof (of theorem). Frieze and Kannan prove that

$$2||p_\tau - \pi||^2 \leq e^{-\frac{\gamma\theta\tau\delta^2}{sd^2}} \log \frac{1}{\pi_*} + \frac{M\pi_\theta sd^2}{\gamma\delta^2}$$

where $\gamma > 0$ is a constant, d is the diameter of the convex body in which we are running the random walk, π_* is $\min_{x \in D} \pi(x)$, θ is a parameter between 0 and 1, and

$$\pi_\theta = \sum_{x \in D : \frac{vol(C(x) \cap \Delta)}{vol(C(x))} \leq \theta} \pi(x).$$

In words, π_θ is the probability of the grid points whose cubes intersect the simplex in less than θ fraction of their volume. The parameter M is defined as $\max_x p_0(x) \log \frac{p_0(x)}{\pi(x)}$, where p_0 is the initial distribution on the states.

For us the diameter d is 1. We will set $\theta = \frac{1}{4}$ and choose δ small enough so that π_θ is a constant. This can be done for example with any $\delta \leq \frac{1}{2(s+n)}$. To see this, consider the simplex blown up by a factor of $\frac{1}{\delta}$ i.e. the set $\frac{1}{\delta}\Delta = \{y | y \geq 0, \sum_i y_i = \frac{1}{\delta}\}$. Now the set of points with integer coordinates correspond to the original grid points. Let B be the set of cubes on the border of this set, i.e. the volume of each cube in B that is in $\frac{1}{\delta}\Delta$ is less than $\frac{1}{4}$. Then by blowing up further by 1 unit, we get a set that contains all these cubes. But the ratio of the volumes is

$$\frac{(\frac{1}{\delta}+1)^s}{(\frac{1}{\delta})^s} = (1+\delta)^s.$$

Also, the performance of these border grid points can only be $(1+\delta)^t$ better than the corresponding (non-blown up) points in the corresponding points. Thus $\pi_\theta \leq (1+\delta)^{n+t} < 2$ for $\delta \leq \frac{1}{2(s+n)}$.

Thus the bound above on the distance to stationary becomes

$$2||p_\tau - \pi||^2 \leq e^{-\frac{\gamma\tau\delta^2}{s}} \log \frac{1}{\pi_*} + \frac{2Ms}{\gamma\delta^2}$$

Next we observe that by our choice of starting point (uniform over the simplex) M is exponentially small. Thus we can ignore the second term in the right hand side. Finally we note that π_* is at least $\delta^s(\frac{\epsilon}{s+n})^n$, which simplifies the inequality to

$$||p_\tau - \pi||^2 \leq e^{-\frac{\gamma\tau\delta^2}{s}}(s+n)^2$$

Our choice of δ (= $O(\frac{1}{(s+n)n})$ implies the theorem (with a different γ)). ∎

Unfortunately, in the presence of transaction costs, the performance function is not necessarily log-concave, and the above random walk does not necessarily rapidly mix.

2.6 Related Work

Cover introduced the notion of UNIVERSAL portfolios [16], proving guarantees for bounded markets with $c_1 \leq x_i^d \leq c_2$. He and Ordentlich then proved the UNIVERSAL guarantees of Theorem 2.3 [17] for arbitrary markets. They also showed there that using a weighted average of CRPs according to a Dirichelet$(1/2, \ldots, 1/2)$ distribution, as opposed to a uniform average, the competitive ratio in Theorem 2.3 is improved to $2(n+1)^{\frac{s-1}{2}}$. Further, they later showed that this is the best possible by proving a lower bound of $O\left(\left(\frac{n}{2}\right)^{\frac{s-1}{2}}\right)$ [18].

In fact, in a bounded market, other algorithms have been shown to have the universal property, i.e. that they asymptotically approach the daily log-performance of the optimal CRP. Recently, Gaivorinski et. al. showed that, with certain assumptions, simply using the CRP which has done best until the current is universal [32]. Helmbold et al. show that the so-called $EG(\eta)$ algorithm is universal for bounded markets. This algorithm maintains a single weight for each stock and has a simple update rule. With the correct choice of the parameter η, they achieved better performance than UNIVERSAL on their experiments. Previous experiments involving UNIVERSAL have generally been limited to two or three stocks. This includes a paper from last year by Cover and Julian [20]. Hopefully, the biased sampling approach will enable larger experiments.

Cover and Ordentlich also consider *side information* [17], a single discrete value known each day, such as the high temperature in New York. They then run a separate, independent UNIVERSAL for each value of side information and get the natural guarantees, i.e. $(n+1)^{k(s-1)}$ with k different values of side information. Singer extends the model by competing against portfolios which switch between different CRPs from time to time [56].

Cover shows that UNIVERSAL can be applied to data compression [19], and Kalai et. al. apply it to combining language models [41]. Foster and Vohra [28] look at UNIVERSAL as making a grid on the simplex and also relate UNIVERSAL to Blackwell's Approachability Theorem. Helmbold et. al. [37] compare different efficient methods of finding the optimal CRP in hindsight.

The questions which we address, those of transaction costs and efficient implementations, were raised in [16, 37, 17, 18, 54, 19]. A nice survey of on-line investment algorithms, including our simple analysis, appears in a book by Alan Borodin and Ran El-Yaniv [10].

Chapter 3

Bounds for K-Fold Cross-Validation

For many machine learning tasks, we are given a fixed-size data set and would like to produce a hypothesis generalizing the data. In addition, we often would like an estimate of the accuracy of that hypothesis on unseen data. This is especially useful for model selection, where we have many hypotheses and we would like to choose the best.

The scientific method proscribes that the data we use to generate our hypothesis should be independent from the data we use to estimate its accuracy. Thus a natural idea is to divide the data into two parts: *training data* for generating a hypothesis and *testing data* (also called the hold-out) for testing its error [53]. The tradeoff is clear. Using more *training data* should presumably lead to a better hypothesis while using more *testing data* should lead to a more accurate estimate of its error. The latter can be made precise by Hoeffding bounds. Assuming that the given data comes from the same distribution as future data, with high probability the estimate will be close to the true error.

Is it possible to beat this hold-out estimate: for a data set of size n and training set of size m, can we get a more accurate estimate than that of testing on $n - m$ data? We show that k-fold cross-validation, one of the most popular forms of cross-validation used in machine learning, does in fact achieve this goal. In k-fold cross-validation, depicted in Figure 3.3b, the data is divided into k equal size *folds*. Learning is performed on all data not in one fold and testing is performed on that held-out fold. The results of these tests are then averaged. So, k-fold cross-validation is exactly an average of k hold-out estimates of size $1/k$ of the data. An extended abstract of this work, joint with Avrim Blum and John Langford, appeared in COLT '99 [9].

There are three difficulties in showing that the k-fold error estimate is better than a hold-out estimate. First, the errors measured on each fold are not necessarily independent. For example, consider *leave-one-out* cross-validation, corresponding to $k = |\text{data}|$, and suppose we are trying to learn the bias of an unfair coin. Our naive learning algorithm believes that the coin has $p(\text{heads}) = 1$ if it has more examples of heads than tails in the data, and $p(\text{heads}) = 0$, otherwise. Taking 100 examples with a fair coin, about 8% of the time we will have 50 heads and 50 tails. In this case, when we leave out a heads, we will predict tails, and vice versa. Thus the leave-one-out estimator will incorrectly estimate that each hypothesis has error 1, when in fact they all have error 0.5. Since the n estimates are identical, the use of all of them was no better than just one. We show that for $2 < k < n$, while these estimates may not be completely independent, there will still be some independence between different folds.

A second difficulty is that, with k different hypotheses, it is not clear what hypothesis to use for future predictions. Practitioners use a completely new hypothesis which is trained on all the data. However,

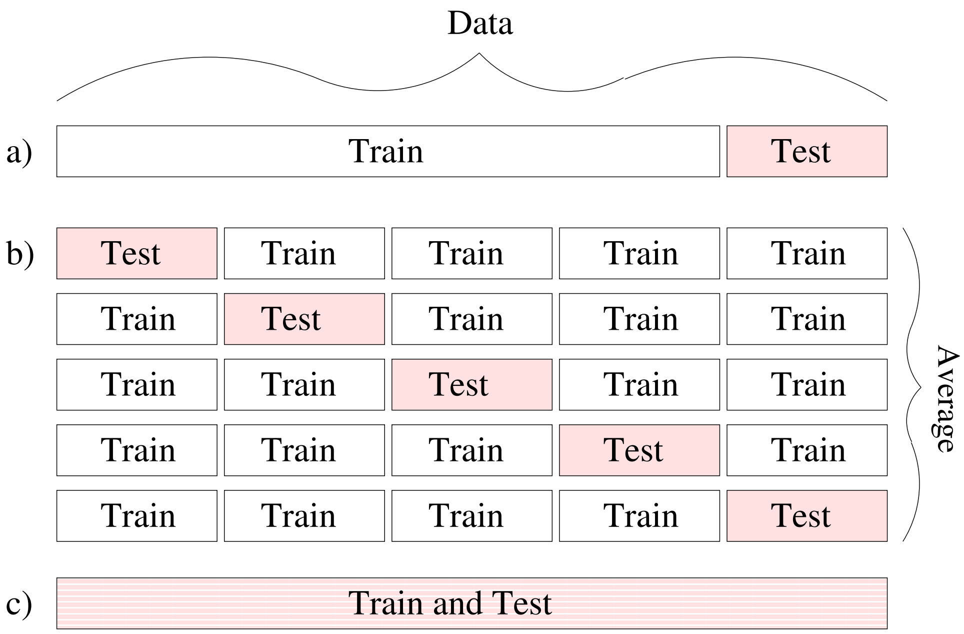

Figure 3.1: a) The (20%) hold-out estimate b) The 5-fold estimate c) The resubstitution estimate

nothing can be said about this new hypothesis without assuming something about the learning algorithm. Previous k-fold bounds, which attempted to do this, were termed sanity-check[1] because they restricted the VC dimension of the learning algorithm, required a notion of hypothesis stability, and only compared the k-fold estimate to the resubstitution estimate [4]. We get around these difficulties by using a *k-fold hypothesis* which is a randomized combination of the k constituent hypotheses. That is, for a new object, we make a prediction according to a random one of the k hypotheses.

The last difficulty comes in comparing the k-fold estimate to the hold-out. We would like to say the k-fold estimate is better than a hold-out of size n/k. Imagine that we will flip an unfair coin ten times, and we want to estimate the probability of heads, p. The full estimator "$\hat{p}_{10} = (\text{total number of heads})/10$" seems better than the one-flip estimator "$\hat{p}_1 = 1$ if the first flip is a head and $\hat{p}_1 = 0$ otherwise", but in what sense? For $p = 1/100$, the chance that $|\hat{p}_1 - p| > 0.05$ is 1/100, while the chance that $|\hat{p}_{10} - p| > 0.05$ is nearly 10/100, namely the chance that any of the flips were heads. Thus, $\hat{p}_{10}$ doesn't completely dominate $\hat{p}_1$ under every conceivable notion of "better". Instead, what *can* be said is that $E\left[|\hat{p}_{10} - p|^m\right] < E\left[|\hat{p}_1 - p|^m\right]$, for all $m \geq 1$.

The same absolute moments are what we use to compare the k-fold estimate to the hold-out. As a corollary, we will show that Hoeffding bounds apply to the k-fold estimate, just as they apply to the hold-out estimate.

3.1 The Learning Model

In order to make theoretical guarantees about the quality of error estimates, we must in some way assume that past data is correlated with future data. We choose a simple theoretical model based on a classification problem[2]. Say there is a finite set of objects X, e.g. pictures of food, and labels L, e.g. "pizza," "donuts,"

[1] Our bounds must be *in*sanity check bounds.

[2] The analysis works for other types of learning problems as well.

etc., such that each object $x \in X$ has a unique label $l(x) \in L$. Furthermore, there is a distribution $\mathcal{D}$ over these objects. We are given labelled data, $(x_1, l_1), (x_2, l_2), \ldots, (x_n, l_n)$, where $l_i = l(x_i)$.

We think of a learning algorithm as a black box that takes as input *training data* in the form of labeled objects, and outputs a *hypothesis* h, a (possibly randomized) function from objects to labels. Ideally we would like to learn a hypothesis h which is not only accurate on the data we've seen ($h(x_i) = l(x_i)$), but generalizes to unseen examples. The generalization error or *true error* of a hypothesis h is the probability, under $\mathcal{D}$, that it mislabels an object, $Pr_{\mathcal{D}}[h(x) \neq l(x)]$. For randomized h, this probability includes the random choices h makes. We would also like to have a high-accuracy estimate of this error, such that $|(\text{estimated error}) - (\text{true error})|$ is small.

3.2 Error Estimation

Our goal is to find a procedure that, given a black-box learning algorithm and a fixed amount of labeled data drawn from $\mathcal{D}$, produces both learning a low-error hypothesis and accurately estimating the true error of this hypothesis under $\mathcal{D}$. In this section, we describe the error estimation part of the procedure.

The *hold-out procedure*, shown in Figure 3.3a, is a natural method for learning a hypothesis and estimating its error on unseen objects. This procedure takes its input, labeled data, and divides into two parts. The first is used as training data, i.e. input to the learning algorithm. The hypothesis output by the learning algorithm is then applied to the remaining data, and the *hold-out estimate* of the error is the fraction of data correctly classified by the hypothesis. In as much as we expect the given data to be similar to future data, this is a good estimate of the error of the learned hypothesis.

For a good learning algorithm, a larger training set would hopefully give a better hypothesis. A larger test set would hopefully give a better estimate of its error. The *k-fold procedure*, shown in Figure 3.3b, is designed to effectively increase the size of the test set without decreasing the size of the training set. This procedure divides the data into k equal *folds*. For each fold, it trains the learning algorithm on the data in the $k - 1$ remaining fold, and tests the learned hypothesis on the held-out fold. It then report the average of these k tests, each of which is really a hold-out estimate, as the *k-fold estimate*.

Of course there are many other possible procedures. For example, the *resubstitution procedure* generates a hypothesis by training on all the data, and the *resubstitution estimate* is the fraction of these same data that are incorrectly classified by the hypothesis. For many learning algorithms, the resubstitution estimate is a very poor estimate of the error of the learned hypothesis. For example, rote memorization and nearest neighbor learning algorithms always produce a zero estimate of error, even when the hypothesis may be far from correct.

3.3 Hypothesis Selection

In each of the three cases, we have described how to estimate the error of a hypothesis, but we have not described which hypothesis to use, corresponding to the error. In the case of the holdout, the *hold-out hypothesis* is trained on the training set only. One is tempted to select the *complete hypothesis*, i.e. the one trained on all the data, as practitioners often do. However, there are several reasons for us not to choose this hypothesis. First, the hold-out estimate is exactly an estimate of the hold-out hypothesis. If the complete hypothesis has much lower true error, then our estimate may be very inaccurate. Second, since our learning algorithm is a black box, we have no guarantee whatsoever of the error of the complete hypothesis. The learning algorithm may, for example, have run out of memory trying to handle the extra data, and the complete hypothesis could be awful.

In practice, the hypothesis chosen for k-fold cross-validation is the complete hypothesis. However, for the same reasons given above, it would be impossible to guarantee the quality of the complete hypothesis for a black-box learning algorithm. Instead, our k-fold procedure outputs the *k-fold hypothesis*, which is a meta-hypothesis that, given an object x, randomly chooses one of the k generated hypotheses h_i and outputs the prediction of that hypothesis, $h_i(x)$. We believe that, from a theoretical point of view, this is the correct choice for the k-fold hypothesis because the k-fold estimate is an unbiased estimate of its error. This means that the expected value of (true error) $-$ (estimated error) is zero, over data sets drawn from $\mathcal{D}$.

One may be tempted to take a majority vote of the constituent hypothesis, but again this may do better or worse than the k-fold estimate. If hypotheses were allowed to predict distributions over labels, rather than just a single label, we might be tempted to use the *average hypothesis*, which is an average of these distributions. The validity of the k-fold estimate then would depend on how we measure errors. Say errors are measured using a loss function L, for which a probability p assigned to the correct label counts as an $L(p)$ fraction of an error. If $L(p) = 1 - p$, then the k-fold estimate would be an unbiased estimate of the average hypothesis, because the average hypothesis has the same error as the k-fold hypothesis. For other loss functions, the k-fold estimate may be biased. Interestingly, one can say that the average hypothesis will have no larger error than the k-fold hypothesis as long as L is convex.

3.4 Notation

Recall that X is the set of objects and $\mathcal{D}$ is a fixed distribution over X. We also have a fixed target function $l : X \longrightarrow L$, the set of labels. A learning algorithm produces a hypothesis h, which is a (possibly randomized) function from X to L. The error of this hypothesis on a particular example $x \in X$ is $e_h(x) = Pr[h(x) \neq l(x)]$, where the probability is taken over randomization in h. The *true error* of this hypothesis is $\bar{e}_h = E_{x \in \mathcal{D}}[e_h(x)]$.

3.5 K-Fold Analysis

Say we have a labelled data set of size n, and $1 < k \leq n$. We divide the data into k equally sized *folds*. Then we generate k hypotheses, $h_1, \ldots, h_k$, where h_i is trained on all the data except the ith fold. We let $\bar{e}_i = \bar{e}_{h_i}$ be the true error of h_i, and $\hat{e}_i$ be the measured error frequency of h_i on the ith fold. As discussed earlier, the *k-fold hypothesis*, h_{K}, makes a prediction on an example x by randomly choosing $1 \leq i \leq k$ and outputting $h_i(x)$. The true error of the k-fold hypothesis is the average of the true errors of its k hypotheses,

$$\bar{e}_{\text{K}} = \frac{\bar{e}_1(x) + \bar{e}_2(x) + \cdots + \bar{e}_k(x)}{k}.$$

Finally, we let the k-fold error estimate be the average of the fold estimates, $\hat{e}_{\text{K}} = (\hat{e}_1 + \hat{e}_2 + \cdots + \hat{e}_k)/k$.

Notice that the estimated and true errors of the k hypotheses and k-fold hypothesis, $\hat{e}_i, \bar{e}_i, \hat{e}_{\text{K}}, \bar{e}_{\text{K}}$, are random variables that are functions of the data set and possibly the randomization parameters of the learning algorithm. We would like the *error discrepancy* $|\hat{e}_{\text{K}} - \bar{e}_{\text{K}}|$ to be small in absolute value.

The mth moment of the error discrepancy is $E\left[|\hat{e}_{\text{K}} - \bar{e}_{\text{K}}|^m\right]$. We begin by showing that, for all $m \geq 1$, the moments of the error discrepancy are *no larger than* those of a single hold-out of size n/k. Notice that the error discrepancy of a single hold-out is $|\hat{e}_1 - \bar{e}_1|$. The following theorem takes the trivial observation that the k-fold error is an unbiased estimate of the true error a step further. Expectations, unless otherwise noted, are over complete data sets drawn i.i.d. from $\mathcal{D}$.

Theorem 3.1 *For all $m \geq 1$, $E\left[(\text{error discrepancy})^m\right]$ is no larger for the k-fold procedure than for a hold-out of a $1/k$ fraction of the data, i.e.,*

$$E[|\hat{e}_K - \bar{e}_K|^m] \leq E[|\hat{e}_1 - \bar{e}_1|^m].$$

Proof. Jensen's inequality for any convex function f and reals x_i is,

$$f\left(\frac{x_1 + x_2 + \cdots + x_n}{n}\right) \leq \frac{f(x_1) + f(x_2) + \cdots + f(x_n)}{n}.$$

Because $|x|^m$ is convex for all $m \geq 1$,

$$\begin{aligned} |\hat{e}_\mathrm{K} - \bar{e}_\mathrm{K}|^m &= \left|\frac{\hat{e}_1 - \bar{e}_1 + \cdots + \hat{e}_k - \bar{e}_k}{k}\right|^m \\ &\leq \frac{|\hat{e}_1 - \bar{e}_1|^m + \cdots + |\hat{e}_k - \bar{e}_k|^m}{k}. \end{aligned}$$

Using linearity of expectation and that, for $1 \leq i \leq k$, $E[|\hat{e}_1 - \bar{e}_1|^m] = E[|\hat{e}_i - \bar{e}_i|^m]$ the expected value of the right-hand side is $E[|\hat{e}_1 - \bar{e}_1|^m]$, whereas the expected value of the left-hand side is $E[|\hat{e}_\mathrm{K} - \bar{e}_\mathrm{K}|^m]$. This completes the proof. ∎

Now we wish to show that the k-fold error is a better estimate. However, it is possible that the hold-out error is a perfect estimate of the true error, if, for example, the learned hypothesis has true error equal to 0 or 1. To say something meaningful, we need to assume the learning algorithm has the property that $Pr[\hat{e}_1 \neq \bar{e}_1] > 0$ (all probabilities are taken over the draw of the full data set). In addition, our proof will need to assume that the instance space X is finite, and that the learning algorithm is insensitive to example ordering. This insensitivity can be enforced in our k-fold procedure simply by shuffling the training examples randomly before giving them to the learning algorithm, on each of the k runs. Thus we are not violating the black-box assumption of our learning algorithm.

It is interesting to note that the k-fold estimate can be identical to the single hold-out estimate if $k = n$ or $k = 2$. In the case where $k = n$ (leave-one-out), Kearns and Ron [45] give several nice examples of poor performance. For instance, a learning algorithm that uses the rule "if I have seen an even number of positive examples then predict positive, else predict negative" will have the property that no matter what the data is, $\hat{e}_1 = \hat{e}_2 \ldots = \hat{e}_n$; thus the leave-one-out estimate will be exactly the same as a hold-out of size 1. Furthermore, if the underlying distribution has 50% positive examples, then the true errors will be the same as well. In the case where $k = 2$, an example is as follows. Suppose that we are to predict the label of integers drawn uniformly in some range $[1, \ldots, 2t]$, and the truth is that all labels are 0. Our hypotheses have a single parameter p. On even integers it will predict 0 with probability p and 1 with probability $1 - p$. On odd integers it will do the opposite, predicting 0 with probability p and 1 with probability $1 - p$. Thus the true error is 50% regardless of p. Furthermore, our "learning" algorithm chooses p to be the fraction of even examples seen in the input. Now, if $k = 2$, we will have two hypotheses with p_1 and p_2, and $\hat{e}_1 = p_1 p_2 + (1 - p_1)(1 - p_2) = \hat{e}_2$. So the two-fold estimate, which is identical to the hold-out estimate, is no better an estimate of the 50% true error.

Theorem 3.2 *Suppose the example space is finite, our learning algorithm is insensitive to example ordering, and the hold-out estimate is not always perfect, i.e. $Pr[\hat{e}_1 \neq \bar{e}_1] > 0$. Then, for $2 < k < n$ and $m \geq 2$,*

$$E[|\hat{e}_K - \bar{e}_K|^m] < E[|\hat{e}_1 - \bar{e}_1|^m],$$

where, unlike the previous theorem, we now have strict inequality.

Proof. Without loss of generality, we assume that all examples in our finite example space have positive probability so that every dataset has positive probability. Now, for a strictly convex function, such as $|x|^m$, $m \geq 2$, Jensen's inequality holds with equality if and only if all the terms x_i are equal. Substituting $x_i = \hat{e}_i - \bar{e}_i$, we see that if $\hat{e}_i - \bar{e}_i \neq \hat{e}_j - \bar{e}_j$ for some dataset, then we are done. Otherwise, for contradiction, assume that

$$\hat{e}_i - \bar{e}_i = \hat{e}_j - \bar{e}_j, \text{ for all data sets, and } 1 \leq i, j \leq n. \tag{3.1}$$

Now, we consider several possible data sets. To describe these, let S_1 be a set of $\frac{n}{k} - 2$ examples, let S_2 be a set of $\frac{n}{k} - 1$ examples, and let $S_3, S_4, \ldots, S_k$ be sets of $\frac{n}{k}$ examples each. The basic idea is that we will be swapping the first element of the first fold with first element of the second fold. Specifically, the data sets we consider (using semicolons to separate the folds) are:

A. $z, x, S_1;\ z', S_2;\ S_3;\ S_4; \cdots$

B. $z', x, S_1;\ z, S_2;\ S_3;\ S_4; \cdots$

C. $z, y, S_1;\ z', S_2;\ S_3;\ S_4; \cdots$

D. $z', y, S_1;\ z, S_2;\ S_3;\ S_4; \cdots$

To distinguish between the hypotheses of different data sets, we'll refer to the errors by their letters, e.g. $\bar{e}_{Bi}$ refers to the true error of the hypothesis h_{Bi} trained on everything but the ith fold in dataset B.

By the assumption of insensitivity to example order, we see that $\hat{e}_{A3} - \bar{e}_{A3} = \hat{e}_{B3} - \bar{e}_{B3}$. By (3.1), we see that $\hat{e}_{A1} - \bar{e}_{A1} = \hat{e}_{B1} - \bar{e}_{B1}$. Similarly, insensitivity to example ordering implies that $\hat{e}_{C3} - \bar{e}_{C3} = \hat{e}_{D3} - \bar{e}_{D3}$ so we have $\hat{e}_{C1} - \bar{e}_{C1} = \hat{e}_{D1} - \bar{e}_{D1}$. Noting that $h_{A1} = h_{C1}$ and $h_{B1} = h_{D1}$, we subtract equations to get,

$$\begin{aligned} \hat{e}_{A1} - \bar{e}_{A1} - (\hat{e}_{C1} - \bar{e}_{C1}) &= \hat{e}_{B1} - \bar{e}_{B1} - (\hat{e}_{D1} - \bar{e}_{D1}) \\ \hat{e}_{A1} - \hat{e}_{C1} &= \hat{e}_{B1} - \hat{e}_{D1}. \end{aligned}$$

Now, again using the fact that $h_{A1} = h_{C1}$ and $h_{B1} = h_{D1}$ we have:

$$e_{A1}(x) - e_{A1}(y) = e_{B1}(x) - e_{B1}(y),$$

where $e_{A1}(x)$ denotes the error of h_{A1} on example x. Since this last equation holds for arbitrary z, z', and S_i, it means that changing a single training example (z to z') does not change the quantity $e(x) - e(y)$. Therefore, $e_h(x) - e_h(y)$ must be the same for any training set, because one training set can be changed to any other by a sequence of individual changes. Since this is also true for arbitrary y, this means the the function $f(x, y) = e_h(x) - e_h(y)$ is well-defined (i.e., it doesn't depend on the training data). In particular, we see that $e_h(x) - \bar{e}_h = E_{y \in D}[e_h(x) - e_h(y)]$ is a constant quantity across training sets for h.

This strict requirement that $e_h(x) - \bar{e}_h$ is constant leads us to conclude that $e_h(x) = e_h(y)$ always. To see this, consider the following data set:

E. $x, x, \ldots, x;\ y, y, \ldots, y;\ S_3;\ S_4;\ \cdots$

By applying (3.1) to data set E, we see that

$$\hat{e}_{E1} - \bar{e}_{E1} = e_{E1}(x) - \bar{e}_{E1} = e_{E2}(y) - \bar{e}_{E2}.$$

But, from the previous paragraph, we know these differences do not depend on the specific training data. Thus, $e_{E1}(x) - \bar{e}_{E1} = e_{E1}(y) - \bar{e}_{E1}$, $e_{E1}(x) = e_{E1}(y)$, and $e_h(x) = e_h(y)$ for any h learned from training data. This implies all individual fold error estimates are perfectly accurate, violating $Pr[\hat{e}_1 \neq \bar{e}_1] > 0$. ∎

It is interesting to consider when the k-fold estimate will be much better than the hold-out. It is sufficient that $\hat{e}_i - \bar{e}_i$ have a significant chance of being different than $\hat{e}_j - \bar{e}_j$, i.e. that these variables are not completely correlated. One scenario in which this is the case is when you have a form of hypothesis stability, which could guarantee that $\bar{e}_j$ is close to $\bar{e}_i$.

Finally, we show a worst-case type of result, that Hoeffding bounds can still be used for the k-fold estimate, as if we had just a hold-out of size n/k:

Theorem 3.3 *Hoeffding bounds hold as if we used n/k testing examples. In particular,*

$$Pr[\hat{e}_K > \bar{e}_K + a] \leq e^{-2a^2 n/k} \text{ and } Pr[\hat{e}_K < \bar{e}_K - a] \leq e^{-2a^2 n/k}.$$

Proof. The proof of Hoeffding bounds for the standard hold-out case of $\hat{e}_1$ and $\bar{e}_1$ with a hold-out set of size $s = n/k$, e.g. [5], begins by bounding $E[e^{\lambda s(\hat{e}_1 - \bar{e}_1)}]$. Then they use Markov's inequality with this bound,

$$Pr[\hat{e}_1 > \bar{e}_1 + a] = Pr[e^{\lambda s(\hat{e}_1 - \bar{e}_1)} > e^{\lambda a}] \leq \frac{E[e^{\lambda s(\hat{e}_1 - \bar{e}_1)}]}{e^{\lambda sa}}.$$

However, since $e^{\lambda sx}$ is a convex function of x, Jensen's inequality implies that,

$$\begin{aligned} e^{\lambda s(\hat{e}\mathrm{K} - \bar{e}\mathrm{K})} &= e^{\frac{\lambda s}{k}(\hat{e}_1 - \bar{e}_1 + \cdots + \hat{e}_k - \bar{e}_k)} \\ &\leq \frac{e^{\lambda s(\hat{e}_1 - \bar{e}_1)} + \cdots + e^{\lambda s(\hat{e}_k - \bar{e}_k)}}{k}. \end{aligned}$$

Thus $E[e^{\lambda(\hat{e}\mathrm{K} - \bar{e}\mathrm{K})}] \leq E[e^{\lambda(\hat{e}_1 - \bar{e}_1)}]$, and the proof goes through. ■

3.6 Related Work

Leave-one-out cross-validation, which is also common in practice, corresponds to $k = n$. There is more prior work on this type of cross-validation [45, 55, 24, 61, 39, 43, 44], as referenced by Kearns and Ron [45]. Their bounds depend on the VC dimension [60] and hypothesis stability [45]. Restrictions of some kind seem unavoidable, as there are interesting examples of situations where the leave-one-out estimate is always off by 50% [45]. These terrible-case examples do not exist for k-fold cross-validation with small k, because it is better than a hold-out set of corresponding size, which is a good estimator. In addition, certain algorithms, such as nearest neighbor, have been shown to have good performance with leave-one-out [23]. Our bounds, however, are not very informative in the leave-one-out case, because we would be comparing it to a hold-out of a single element.

As far as we know, the only other theoretical guarantees of the k-fold estimate are given by Anthony and Holden [4], who extend the analysis of Kearns and Ron [45] to the k-fold setting. They judge the k-fold error as an estimate of the true error of the hypothesis trained on all the data. This is a natural formulation of the problem, because in practice the hypothesis often chosen is this untested hypothesis. However, because the new hypothesis is untested, their performance guarantees depend on VC dimension, and their results are sanity-check bounds which relate the k-fold error to the resubstitution error. For large k, leaving a small number out, the resubstitution error may be a better estimate than the corresponding hold-out, and their bounds may bridge the gap between leave-one-out ($k = n$) and typical k-fold (k is a small constant).

In very nice theoretical work, Kearns has analyzed the right amount of data to hold out for the purposes of model selection [44]. In this setting, he would like to select the best hypothesis from a set of hypotheses generated by learning algorithms. He shows that there is, in some sense, an optimal amount of data to hold out for the purposes of testing these hypotheses. Since cross validation is very useful for model selection [43], it would be useful to do the same for k-fold cross-validation.

Other forms of cross-validation that are used in practice include the bootstrap [26] and jackknife estimates [25]. Analyses of these are average case or asymptotic, so it would also be nice to provide meaningful worst-case guarantees.

On another note, if the k-fold hypothesis is chosen as an average of the k generated hypotheses rather than the randomizing hypothesis, it is similar to bagging[11]. In that situation, the goal is to reduce the generalization error, which Breiman claims can be achieved by reducing the variance in the hypothesis. On the other hand, we are concerned more with the variance in our error discrepancy. Thus decreasing the generalization error of the final hypothesis would make the k-fold error a worse estimate. It would also be interesting to explore the connection between hypothesis instability, which Breiman discusses for the purposes of reducing generalization error, to hypothesis stability, which Kearns and Ron [45] trace back to Devroye and Wagner [24] for the purposes of accurate error estimation.

Finally, *progressive validation*, introduced in [9] is another method of beating the hold-out. We still have a hold-out of size m, but when testing the ith element of the hold-out, we train on the training set *and* the first $i-1$ elements of the hold-out.

Chapter 4

Splay Trees

In this chapter, we first describe splay trees and then discuss the major open question about splay trees – the so called dynamic optimality conjecture [58]. We do not prove the dynamic optimality of splay trees. Instead, we describe a binary search tree algorithm with what we call dynamic search optimality. We hope that this algorithm may lead to a dynamically optimal algorithm.

4.1 Binary Search Trees

A binary search tree (BST) is a data structure for storing a totally ordered set of (key,object) pairs, such as (name,social security number) with the standard lexicographic ordering on names. Each node in the tree represents one object, and everything in the left subtree must have a key less than the node, while everything in the right subtree must have a key greater than the node. For our purposes, we are not interested in the actual objects or even the specific values of the keys (just their relative ordering), so we will assume that the trees only contain integer keys $1, 2, \ldots, n$. An example of such a tree containing integers is shown in Figure 4.1.

In general, the operations on such trees include insertion, deletion, access(finding a key), and various other operations. In fact, we will specifically be interested in just the access operations. For the access operation, we simply have to start at the root of the tree and move down the edges until we find the accessed node. The search cost of an access is simply the depth of that node.

You can see that if a tree is very deep, then the search cost of an access may be very high. A BST of n nodes has depth between $\log n$ and n. One traditional way to ensure low-cost accesses was to *balance* the tree, i.e. make sure that it doesn't have more than $O(\log n)$ depth. This can be achieved by several methods such as AVL trees [2] and red-black trees [34]. The standard way to restructure a tree is by rotations. A rotation, shown in Figure 4.1, is a way of swapping a child and parent node in a BST, such that the order properties of a BST are preserved.

4.2 Static and Dynamic Optimality

As mentioned, the search cost of a node is simply the depth of that node. In addition, we would like to allow algorithms to perform arbitrary rotations, at a cost of one per rotation. An on-line algorithm is said to have

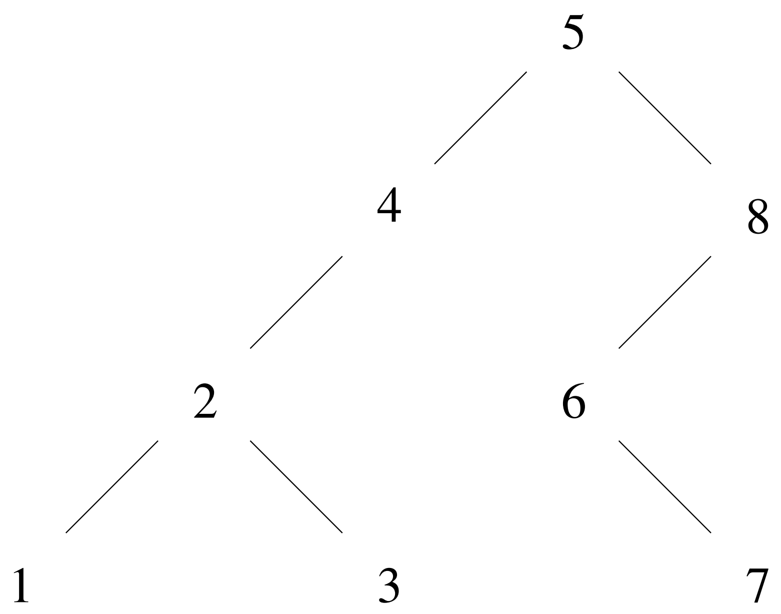

Figure 4.1: An example of a binary search tree.

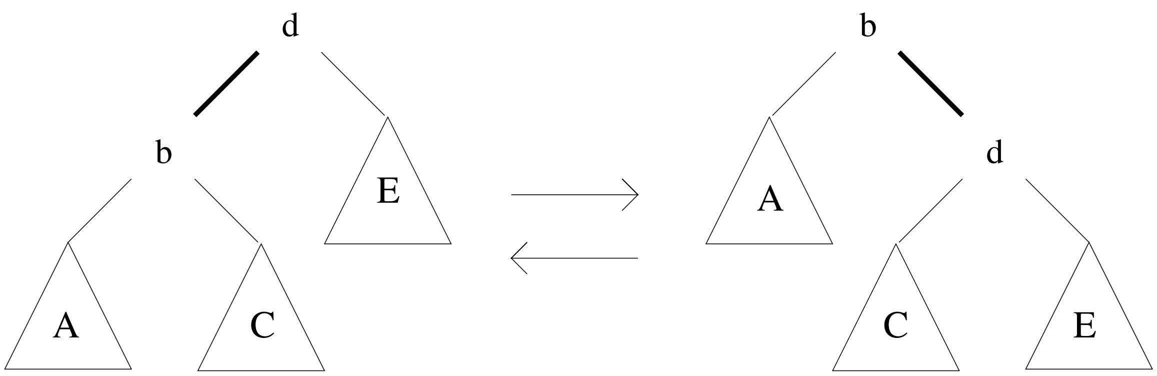

Figure 4.2: Rotating the edge (b, d).

dynamic optimality if it has a constant competitive ratio, i.e.,

Definition 4.1 *(Dynamic optimality) An algorithm is dynamically optimal if its cost on an arbitrary sequence of accesses is $O(n + \text{DYNAMIC-OPT})$, where DYNAMIC-OPT is the minimum cost of this sequence of accesses for a dynamic BST algorithm that pays 1 per rotation.*

Sleator and Tarjan have conjectured that splay trees, defined in the next section, are dynamically optimal [58]. While they were not able to prove this, they did show *static optimality* for splay trees, a weaker condition stating that the algorithm is constant competitive relative to the class of static BSTs, i.e.,

Definition 4.2 *(Static optimality) An algorithm is statically optimal if its cost on an arbitrary sequence of accesses is $O(n+\text{STATIC-OPT})$, where STATIC-OPT is the minimum cost of this sequence of accesses for a static BST that does no rotations.*

Various other properties of BST algorithms have been proposed and shown for splay trees, but they are almost all corollaries of dynamic optimality. In particular, dynamic optimality implies of course static optimality and the much more difficult dynamic finger conjecture, proved by Cole [13, 14, 15]. Tarjan has proven the sequential access theorem, stating that the cost of accessing nodes 1 through n by splays is $O(n)$, regardless of the starting tree [59]. This is also necessary condition for dynamic optimality to be true.

We believe that there is some dynamically optimal BST algorithm, which would be progress towards the dynamic optimality of splay trees.

Conjecture 4.3 *There is some on-line BST algorithm that is dynamically optimal.*

In trying to prove this conjecture, we ignore the computational costs associated with deciding which nodes to rotate. Of course, this allows us to propose some ridiculously inefficient algorithms, but our main goal is to find out if the above conjecture is true or false.

Dynamic optimality, for any BST algorithm, may be difficult to achieve for two reasons. First of all, the algorithm has to decide which nodes to keep near the root. Secondly, it has to be able to get these nodes near the root without using too many rotations. We show that the first difficulty is not insurmountable. That is, we break the cost of an access into two parts, the *search cost* and the *rotation cost*. Then, we have an algorithm with *dynamic search optimality*, meaning that its search cost is $O(n+\text{DYNAMIC-OPT})$, where DYNAMIC-OPT is still the minimum total cost (search cost + rotation cost) BST algorithm, for any sequence.[1] This type of unfair competitive analysis, where the cost function for the on-line algorithm is different than the cost function of the off-line algorithm, is not uncommon [7]. Furthermore, static optimality can be viewed in the same way, if we consider the cost of rotations to be infinite cost for off-line algorithms.

Definition 4.4 *(Dynamic search optimality) An algorithm has dynamic search optimality if its* search *cost on any sequence of accesses is $O(n+\text{DYNAMIC-OPT})$, where DYNAMIC-OPT is the minimum cost of this sequence of accesses for a dynamic BST algorithm that pays 1 per rotation.*

There is no especially strong evidence suggesting that any BST algorithm is dynamically optimal. A natural approach to disproving this would be to present a set of sequences, and argue that no on-line algorithm can handle all these sequences in a dynamically optimal manner. The simplest form of this argument would be that the on-line algorithm cannot "guess" which node comes next and therefore has too many nodes to keep near the root. However, the existence of an algorithm with dynamic search optimality implies that this type of argument is not possible.

[1] If rotations are free, a natural idea is at each step, to choose the tree of the optimal off-line algorithm so far. This is not as simple as it sounds, because off-line optimality is ambiguous. For example, suppose you start with a 7 node "line" tree of depth 7. After several accesses to node 1, the deepest node, any optimal off-line algorithm must have brought it to the root. There are 132 equally optimal ways to do this, all using 6 rotations.

Wilber [65] made some progress by proving lower bounds for off-line algorithms. In particular, he has shown that a random sequence of accesses in $\{1, 2, \ldots, n\}$ costs an expected $\Omega(\log n)$ per access for off-line algorithms. This is another necessary condition for dynamic optimality to be possible. For, any on-line algorithm costs an expected $\theta(\log n)$ just in search cost (not counting rotations) since the average depth of a node is $\theta(\log n)$.

We go slightly farther in analyzing the cost of random sequences. We show that the number of sequences with off-line cost k is less than 2^{12k} for any k. For information-theoretic reasons, dynamic optimality of any BST algorithm would imply that there are less than 3^k such sequences.[2] Essentially, we give a way to describe off-line rotations in 12 bits per rotation, even though there are, in general, $n-1$ possible rotations one can perform.

4.3 Splay Trees

Splay trees are an adaptive BST algorithm. When an element is accessed, several rotations are performed in the tree to move that node to the root. There are three cases based on the position of the accessed element x in the tree:

A. If x is a child of the root, rotate x up in the tree.

B. If x is a left child of a right child (or a right child of a left child), then rotate x up twice.

C. If x is a left child of a left child (or a right child of a right child), then first rotate x's parent up and then x up.

It is not difficult to see that every rotation described above decreases x's depth in the tree by one. When a node is accessed, the above is applied until x is at the root. For a good survey of splay trees, see [3].

We view the *cost* of an access to be the depth of the element (with the root having depth 1) plus the number of rotations performed. For splay trees, this cost is $2d-1$ if a node at depth d is accessed. For a static tree, i.e. no rotations are performed, it is simply d. Sleator and Tarjan proved the *static optimality* of splay trees. This says that splay trees are as good, to within a constant factor, as any static tree.

Theorem 4.5 *(Static optimality of splay trees [58]) Starting from any tree on n nodes, the cost of an arbitrary sequence of accesses by splaying is $O(n + \text{STATIC-OPT})$, where STATIC-OPT is the minimum cost of this sequence of accesses for a static tree.*

From this theorem, we see immediately that the average cost per access in splaying is $O(\log n)$ as long as $m \geq n$, because the static complete binary search tree has depth $\log n$.

4.4 Related Work

On-line analysis was invented by Sleator and Tarjan [57]. The state of affairs with regards to splay tree conjectures and theorems is described well in [3]. In terms of the off-line optimal algorithm, little is known. Wilber [65] has shown a particular sequence that has a $O(n \log n)$ lower bound. The sequence involves writing down the numbers between 0 and $2^k - 1$ in binary in order, using k bits each. Then simply reverse the bits in each number, i.e. $1010 \to 0101$.

[2] Given an on-line algorithm, such as splaying, any access sequence can be described by the location of each node in its corresponding tree. This can be described using 3 symbols (left, right, and stop) and has length proportional to the search cost. There is no need to describe the rotations performed by the on-line algorithm, since those can be determined from the access sequence. So, for any on-line algorithm, there are at most 3^k sequences costing less than k.

Lucas shows that a constant competitive optimal off-line algorithm exists that rotates edges which form a connected subtree containing the root and the next element to be accessed [51]. She also shows that if x and its descendents are never accessed, then the optimal algorithm need not rotate them.

Sleator and Tarjan prove most of their results by assigning general weights $w_1, \ldots, w_n$ to the different nodes. They define a potential which assigns each node a rank that is the log of the sum of weights on it and all of its children. The total potential is the sum of these ranks, and they show that the amortized cost of splaying x is at most $3(r(t) - r(x)) + 1$, where $r(t)$ is the rank of the root and $r(x)$ is the rank of x. From this theorem, they are able to establish the balance theorem, stating that the amortized cost per access is $O(\log n)$, the static optimality theorem, and the static finger theorem, which says that for a fixed element $1 \leq y \leq n$ the amortized cost of splaying x is $O(\log |x - y| + 1)$.

The dynamic finger theorem, which would follow from dynamic optimality, was proven by Cole [13, 14, 15]. This states that the amortized cost of splaying x is $O(\log |x - y| + 1)$, where y was the node splayed prior to x.

Several variants on splaying have been suggested including top-down splaying, semi-splaying (where the node being splayed isn't moved all the way to the root) [58], and randomized splaying [31, 1]. Splay trees have also been applied to data compression [33].

4.5 Lower bound

We assume that the nodes in the tree are simply the numbers $1, 2, \ldots, n$, and m is the length of the access sequence. We further assume that all algorithms, on- and off-line, begin with the same fixed tree, say, rooted at n and having depth n. In this section, we prove the following. The constant 12 is not really important, and we are mostly interested in the fact that it is $2^{O(k)}$.

Theorem 4.6 *The number of access sequences having optimal off-line cost k is at most 2^{12k}, for all $k \geq 0$, regardless of n or m.*

Proof. We use an information theoretic argument based on the the fact that one can concisely describe any sequence having optimal off-line cost k. To summarize, we will argue that it is possible to describe any access sequence via the trees used in the optimal off-line algorithm and get a description that is of size $O(k)$. First, loosing a factor of two, we assume that the off-line algorithm moves the next node accessed to the root prior to its access. Then, it suffices to describe the sequence of trees, because the accesses will just be their roots. To do this, we describe the set of rotations performed from each tree to the next, which we do in at most $6r_i$ bits if there are r_i rotations. This implies that there are at most 2^{6k} possible sequences of cost k because there are at most 2^{6k} descriptions of length $6k$. However, we lose an overall factor of two due to the following assumption.

As several people have observed, we may assume that the off-line algorithm rotates the next node to be accessed to the root before each access. This adds at most a factor of two to the optimal cost. Given any off-line algorithm, we can modify it by making it rotate a node to the root immediately before accessing it. If the item was at depth d, then we have paid an additional $d - 1$, but we have decreased the search cost by $d - 1$. Immediately after the access, we can reverse the rotations (all rotations are reversible), and move the node back to where it was at a cost of $d - 1$. Thus, we have paid $2(d - 1)$ when the former algorithm paid only $d - 1$.

Like Lucas [51], we think of a rotation as an edge rotation which changes a single edge from either left to right or right to left. Of course, the nodes adjacent to an edge may change. But based on our assumption that the next access is rotated to the root, it is not difficult to see that all the edges on its path to the root must be rotated at least once.

Lucas argues that without further loss of generality one may assume that the set of edges rotated by an optimal off-line algorithm form a single connected component that includes the root and the next node to be accessed. Briefly, this is because any rotations of edges not in this connected component could easily be delayed (using lazy programming) until they are in such a connected component. Their delay will not affect the search cost, since these rotations cannot affect the depth of the next access, nor does their delay affect the rotation costs.

Next, observe that regardless of the order of the rotations, one can completely describe the result of the rotations in $6w$ bits if there are w edges. First, describe the subset of edges that were rotated one or more times. Since this is a rooted subtree, one can describe this using the symbols left (00), right (01) and up (1), to form a cycle that traverses each edge twice, using a total of $3w$ bits. Next describe their position in the resulting tree. In the resulting tree, these edges will still be a rooted subtree, so one can describe them also with $3w$ bits. First note that the set of nodes in this subtree doesn't change even as the positions of the edges do. Secondly, notice that the shape of this subtree completely determines the positions of all the nodes, because this is a BST. Finally, note that off-line algorithm has to perform at least w rotations.

Thus, one can describe the optimal sequence of trees (and thus the access sequence) in $6r$ bits if it performs r rotations. Since we lose a factor of two due to our first assumption, this proves the theorem. ∎

4.6 Dynamic Search Optimality

In this section, we will consider probability distributions. A tree can be thought of predicting the next access, where it predicts nodes closer to the root with higher probability. From our lower bound, we see

Corollary 4.7 *There is a probability distribution over arbitrary sequences of accesses, that assigns probability at least 2^{-13k} to an access sequence of optimal off-line cost k.*

Proof. Choose a cost k according to the distribution $1/2^k$. By Theorem 1, there are at most 2^{12k} sequences of that cost. ∎

It is easy to convert a BST into a probability distribution p such that $p(j) \geq 3^{-\text{depth}(j)}$. Simply choose j by beginning at the root, going left, right, or stopping, each with probability 1/3 (when possible). It is also possible to convert a probability distribution into a tree.

Observation 4.8 *For a probability distribution p over individual accesses, we can create a BST such that $depth(a) \leq 1 - \log p(a)$ for any node a.*

Proof: For the root, choose the first i such that $\sum_1^{i-1} p(i) \leq 1/2$ and $\sum_{i+1}^{n} p(i) \leq 1/2$. Recurse on the numbers less than i (normalizing p) to create the left subtree, and the numbers greater than i for the right subtree. It is easy to see that the total probability of any subtree rooted at depth d is at most $1/2^{d-1}$ so a node of probability $p(i)$ cannot be deeper than $-\log p(i)$. ∎

We can combine these two ideas to make an on-line algorithm.

Theorem 4.9 *For any probability distribution p over access sequences, we can create an on-line algorithm with search cost at most $m - \log p(a_1 a_2 \ldots a_m)$ for every access sequence $a_1 a_2 \ldots a_m$.*

Proof. The distribution p can be thought of as predicting the next access from the previous accesses. In particular, the conditional probability of the next access given the previous accesses is ,

$$p_i(a_i) = p(a_i | a_1 a_2 \ldots a_{i-1}) = \frac{\sum_{b_{i+1},\ldots,b_m} p(a_1 \ldots a_{i-1} a_i b_{i+1} \ldots b_m)}{\sum_{b_i,\ldots,b_m} p(a_1 \ldots a_{i-1} b_i b_{i+1} \ldots b_m)}.$$

We can write p as a product of the conditional distributions of access i, i.e.,

$$p(a_1 a_2 \ldots a_m) = \prod_1^m p_i(a_i).$$

Our on-line algorithm works as follows. For the ith access, we have enough information to compute p_i. We then convert p_i into a tree by the method of Observation 1. Thus, the depth of access a_i in this tree will be no more than $1 - \log p_i(a_i)$. Our total search over m accesses is at most

$$\sum_1^m 1 - \log p_i(a_i) = m - \log p(a_1 a_2 \ldots a_m).$$

Corollary 4.10 *There is an on-line algorithm that has dynamic search optimality. In particular, on any access sequence, its search cost is at most 14 times the optimal off-line total cost.*

Proof. We use the probability distribution of Corollary 1 in combination with Theorem 2 to get a total cost of m plus 13 times the optimal off-line cost. But m is no larger than the optimal off-line cost. ∎

4.7 Future Work

In trying to go from dynamic search optimality to dynamic optimality, we must find a way to ensure that the number of rotations is small, i.e. not much more than the total cost of the optimal algorithm. It would suffice to find a dynamlically search optimal algorithm whose rotation cost was bounded by a constant times the search cost. This is not true for the algorithm we have proposed. Our algorithm makes no attempt to save on rotations, and goes from a probability distribution to a tree after each access, using in no way the previous tree.

One possibility would be to carefully analyze how it is that the probability distribution over next accesses changes. When node i is accessed, how exactly does it affect the probability of node j being accessed next? If one understood this well, then perhaps one could move between consecutive distributions with few rotations.

It would be nice to extend the technique we used for the move-to-front analysis in the introduction to splay trees. Suppose we make it mandatory to immediately move the requested node to the root exactly as splaying does. Then, by the same reason as in move-to-front, we will increase the cost of any algorithm by at most a constant factor. Next, suppose we increase the cost of each rotation by a constant factor c, further increasing the cost by at most a factor of c. Then, one might hope to show that the lazy approach works, i.e. delaying a rotation to the next round will not cost anything extra. Unfortunately, this is not true for splay trees, but it might be true for some variation on them, such as randomized splay trees [31, 1].

Bibliography

[1] S. Albers and M. Karpinski. Randomized splay trees: theoretical and experimental results. Technical Report CS-85212, Universitat Bonn, 2000.

[2] G. Adelson-Velskii and E. Landis. An algorithm for the organization of information. *Sov. Math. Dokl. 3*, 1259-1262, 1962.

[3] S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms: The State of the Art*, edited by Amos Fiat and Gerhard Woeginger. Springer LNCS 1442, pages 31-51, 1998.

[4] M. Anthony and S. Holden. Cross-Validation for Binary Classification by Real-Valued Functions: Theoretical Analysis In *Proc. Eleventh Annual Conference on Computational Learning Theory*, 1998.

[5] N. Alon and J. Spencer. *The Probabilistic Method.* Wiley, 1991.

[6] D. Applegate and R. Kannan. Sampling and integration of near log-concave functions. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, 1991.

[7] A. Blum, C. Burch, and A. Kalai. Finely competitive paging. In *Proceedings of the 40th Annual Symposium on the Foundations of Computer Science* (FOCS '99), 1999.

[8] A. Blum and A. Kalai. Universal portfolios with and without transaction costs. *Machine Learning*, 35:3, 1999.

[9] A. Blum, A. Kalai, and J. Langford. Beating the holdout: bounds for k-fold and progressive cross-validation. In *Proceedings of the 10th Annual Conference on Computational Learning Theory (COLT '99)*, 1999.

[10] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge, 1998.

[11] L. Brieman. Bagging predictors. *Machine Learning*, 24(2):123-140, 1996.

[12] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. In *Annual ACM Symposium on Theory of Computing*, pages 382–391, 1993.

[13] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part 1: Splay sorting log n-block sequences. Technical Report 471, Courant Institute, NYU, 1989.

[14] R. Cole. On the dynamic finger conjecture for splay trees. Part 2: Finger searching. Technical report 472, Courant Institute, NYU, 1989.

[15] R. Cole. On the dynamic finger conjecture for splay trees. In *Proc. Symp. on Theory of Computing* (STOC '90), pages 8-17, 1990.

[16] T. Cover. Universal portfolios. *Math. Finance*, 1(1):1-29, January 1991.

[17] T. Cover and E. Ordentlich. Universal portfolios with side information. *IEEE Transactions on Information Theory*, 42(2), March 1996.

[18] T. Cover and E. Ordentlich. The Cost of Achieving the Best Portfolio in Hindsight. Department of Statistics Technical Report NSF-90, Stanford University, 1996.

[19] T. Cover. Universal data compression and portfolio selection. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 534-538, Oct 1996.

[20] T. Cover and D. Julian. Performance of Universal Portfolios in the Stock Market. In *Proceedings of IEEE International Symposium on Information Theory*, (ISIT 2000), 2000.

[21] M. Davis and A. Norman. Portfolio Selection with Transaction Costs. *Mathematics of Operations Research*, 15(4), November 1990.

[22] A. DeSantis, G. Markowsky, and M. Wegman. Learning probabilistic prediction functions. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 110–119, Oct 1988.

[23] L. Devroye, L. Gyrofi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer-Verlag, 1996.

[24] L. Devroye and T. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IT-25(5):601-604, 1979.

[25] B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics,* 7, 1979.

[26] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* Chapman-Hall, London, 1993.

[27] D. Foster and R. Vohra. A randomization rule for selecting forecasts. *Operations Research*, 41:704–709, 1993.

[28] D. Foster and R. Vohra. Regret in the On-line Decision Problem. *Games and Economic Behavior*, Vol. 29, No. 1/2, pp. 7-35, Nov 1999.

[29] A. Frieze and R. Kannan. Log-Sobolev inequalities and sampling from log-concave distributions. Annals of Applied Probability 9, 14-26.

[30] Y. Freund and R. Shapire. Discussion of the paper "Arcing classifiers" by Leo Breiman. *Annals of Statistics*, 26(3): 824-832, 1998.

[31] M. Furer. Randomized splay trees. In *Proc. of the Tenth Annual ACM-SIAM Symp. on Discrete Algorithms* (SODA '99), 1999.

[32] A. Gaivoronski and F. Stella. Stochastic nonstationary optimization for finding universal portfolios. To appear in *Annals of Operations Research.*

[33] D. Grinberg, S. Rajagopalan, and K. Wei. Splay trees for data compression. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA '95), 1995.

[34] L. Guibas and R. Sedgewich. A dichromatic framework for balanced trees. In *Proc. 19th IEEE Symp. on Foundations of Computer Science*, 8-21, 1978.

[35] D. Haussler, J. Kivinen, and M. Warmuth. Tight worst-case loss bounds for predicting with expert advice. Technical Report UCSC-CRL-94-36, University of California, Santa Cruz, November 1994.

[36] D. Helmbold, R. Schapire, Y. Singer, and M. Warmuth. A comparison of new and old algorithms for a mixture estimation problem. *Proceedings of the Eighth Annual Workshop on Computational Learning Theory*, pages 69-78, 1995.

[37] D. Helmbold, R. Schapire, Y. Singer, and M. Warmuth. On-line portfolio selection using multiplicative updates. *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

[38] D. Helmbold and M. Warmuth. On Weak Learning. *JCSS*, 50(3): 551-573, 1995.

[39] S. Holden. PAC-like upper bounds for the sample complexity of leave-one-out cross validation. In *Proceedings of the Ninth Anual ACM Workshop on Computational Learning Theory*, pages 41-56, 1990.

[40] R. Kannan, L. Lovasz and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. Random Structures and Algorithms 11, 1-50.

[41] A. Kalai, S. Chen, A. Blum, and R. Rosenfeld. On-line Algorithms for Combining Language Models. In *Proceedings of the International Conference on Accoustics, Speech, and Signal Processing* (ICASSP '99), 1999.

[42] A. Kalai and S. Vempala. Efficient Algorithms for Universal Portfolios. In *Proceedings of the 41st Annual Symposium on the Foundations of Computer Science* (FOCS '00), 2000.

[43] M. Kearns, Y. Mansour, A. Ng, and D. Ron. An experimental and theoretical comparison of model selection. In *The International Joint Conference on Aritificial Intelligence*, 1985.

[44] M. Kearns. A Bound on the Error of Cross Validation Using the Approximation and Estimation Rates, with Consequences for the Training-Test Split. In *Advances in Neural Information Processing Systems 8*, pp. 183-189. MIT Press, 1996.

[45] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. In *Proc. Tenth Annual Conference on Computational Learning Theory*, 1997.

[46] J. Kivinen and M. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, University of California, Santa Cruz, June 1994.

[47] N. Littlestone. Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms. Technical Report UCSC-CRL-89-11, University of California, Santa Cruz 1989.

[48] N. Littlestone. From on-line to batch learning. In *Proceedings of the 2nd Annual Workshop on Computational Learning Theory*, pp. 269–284, 1989.

[49] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Inform. Computation*, 108:212-261, 1994.

[50] L. Lovasz and M. Simonovits. On the randomized complexity of volume and diameter. In *Proceedings of the IEEE Symp. on Foundation of Computer Science,*, 1992.

[51] J. Lucas. Canonical forms for competitive binary search tree algorithms. Technical Report No. DCS-TR-250, Computer Science Department, Rutgers University, 1988.

[52] N. Metropolis, A. Rosenberg, M. Rosenbluth, A. Teller and E. Teller. Equation of state calculation by fast computing machines. In *Journal of Chemical Physics*, 21 (1953), pp 1087-1092.

[53] C. Mosier. Problems and Designs of Cross-Validation. *Educational and Psychological Measurement,* 11, 1951.

[54] E. Ordentlich. and T. Cover. Online portfolio selection. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, Desenzano del Garda, Italy, 1996.

[55] W. Rogers and T. Wagner. A fine sample distribution-free bound for local discrimination rules. *The Annals of Statistics,* 6(3):506-514, 1978.

[56] Y. Singer. Switching portfolios. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence* (UAI-98), 1998.

[57] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *CACM* 28:202-208, 1985.

[58] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652-686, 1985.

[59] R. Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5(4), 1985, 367–378.

[60] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264-280, 1971.

[61] V. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer Verlag, New York, 1982.

[62] V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.

[63] V. Vovk. A game of prediction with expert advice. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 51–60. ACM Press, New York, NY,

[64] V. Vovk and C. Watkins. Universal Portfolio Selection. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, 12-23, 1998.

[65] R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18:56-67, 1989.