Computational Learning Theory Lecture Notes for CS 582 Spring Semester, 1991

Sally A. Goldman

This manuscript is a compilation of lecture notes from the graduate level course CS 582, "Computational Learning Theory," I taught at Washington University in the spring of 1991. Students taking the course were assumed to have background in the design and analysis of algorithms as well as good mathematical background. Given that there is no text available on this subject, the course material was drawn from recent research papers. I selected the first twelve topics and the remainder were selected by the students from a list of provided topics. This list of topics is given at the end of... Read complete abstract on page 2.
Complete Abstract:

This manuscript is a compilation of lecture notes from the graduate level course CS 582, "Computational Learning Theory," I taught at Washington University in the spring of 1991. Students taking the course were assumed to have background in the design and analysis of algorithms as well as good mathematical background. Given that there is no text available on this subject, the course material was drawn from recent research papers. I selected the first twelve topics and the remainder were selected by the students from a list of provided topics. This list of topics is given at the end of these notes. These notes were mostly written by the students in the class and then reviewed by me. However, there are likely to be errors and omissions, particularly with regard to the references. Readers finding errors in the notes are encouraged to notify me by electronic mail (sg@cs.wustl.edu) so that later versions may be corrected.
Computational Learning Theory
Lecture Notes for CS 582
Spring Semester, 1991

Sally A. Goldman

WUCS-91-36

June 1991

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899
Computational Learning Theory

Lecture Notes for CS 582
Spring Semester, 1991

Sally A. Goldman
Department of Computer Science
Washington University
St. Louis, Missouri 63130
Preface

This manuscript is a compilation of lecture notes from the graduate level course CS 582, "Computational Learning Theory," I taught at Washington University in the spring of 1991. Students taking the course were assumed to have background in the design and analysis of algorithms as well as good mathematical background. Given that there is no text available on this subject, the course material was drawn from recent research papers. I selected the first twelve topics and the remainder were selected by the students from a list of provided topics. This list of topics is given at the end of these notes.

These notes were mostly written by the students in the class and then reviewed by me. However, there are likely to be errors and omissions, particularly with regard to the references. Readers finding errors in the notes are encouraged to notify me by electronic mail (sg@cs.wustl.edu) so that later versions may be corrected.
Acknowledgements

This compilation of notes would not have been possible without the hard work of the following students: Andy Fingerhut, Nilesh Jain, Gadi Pinkas, Kevin Ruland, Marc Wallace, Ellen Witte, and Weilan Wu.

I thank Mike Kearns and Umesh Vazirani for providing me with a draft of the scribe notes from their Computational Learning Theory course taught at University of California at Berkeley in the Fall of 1990. Some of my lectures were prepared using their notes. Also most of the homework problems which I gave came from the problems used by Ron Rivest for his Machine Learning course at MIT taught during the Falls of 1989 and 1990. I thank Ron for allowing me to include these problems here.
# Contents

**Introduction**  
1.1 Course Overview ........................................... 9  
1.2 Introduction ............................................ 10  
1.2.1 A Research Methodology ............................... 10  
1.2.2 Definitions .......................................... 11  
1.3 The Distribution-Free (PAC) Model ....................... 12  
1.4 Learning Monomials in the PAC Model .................... 13  
1.5 Learning $k$-CNF and $k$-DNF ............................. 15  

**Two-Button PAC Model**  
2.1 Model Definition ....................................... 17  
2.2 Equivalence of One-Button and Two-Button Models ....... 17  
    Aside: Chernoff Bounds ................................ 19  

**Learning $k$-term-DNF**  
3.1 Introduction ........................................ 23  
3.2 Representation-dependent Hardness Results ............. 23  
3.3 Learning Algorithm for $k$-term-DNF .................... 26  
3.4 Relations Between Concept Classes ..................... 27  

**Handling an Unknown Size Parameter**  
4.1 Introduction ........................................ 29  
4.2 The Learning Algorithm ................................ 29  
    Aside: Hypothesis Testing ................................. 29  

**Learning With Noise**  
5.1 Introduction ........................................ 33  
5.2 Learning Despite Classification Noise ................. 35  
5.2.1 The Method of Minimizing Disagreements ............ 35  
5.2.2 Handling Unknown $\eta_b$ .......................... 37  
5.2.3 How Hard is Minimizing Disagreements? .............. 37  
5.3 Learning $k$-CNF Under Random Misclassification Noise  
5.3.1 Basic Approach .................................... 38
### CONTENTS

5.3.2 Details of the Algorithm ........................................ 41

**Occam’s Razor** ........................................ 43

6.1 Introduction ........................................ 43
6.2 Using the Razor ........................................ 43

**The Vapnik-Chervonenkis Dimension** ........................................ 47

7.1 Introduction ........................................ 47
7.2 VC Dimension Defined ........................................ 47
7.3 Example Computations of VC Dimension ........................................ 48
7.4 VC Dimension and Sample Complexity ........................................ 51
7.5 Some Relations on the VC Dimension ........................................ 54

**Representation Independent Hardness Results** ........................................ 57

8.1 Introduction ........................................ 57
8.2 Previous Work ........................................ 57
8.3 Intuition ........................................ 58
8.4 Prediction Preserving Reductions ........................................ 59

**The Strength of Weak Learnability** ........................................ 61

9.1 Introduction ........................................ 61
9.2 Preliminaries ........................................ 61
9.3 The Equivalence of Strong and Weak Learning ........................................ 62
9.3.1 Hypothesis Boosting ........................................ 62
9.3.2 The Learning Algorithm ........................................ 64
9.3.3 Correctness ........................................ 64
9.3.4 Analysis of Time and Sample Complexity ........................................ 67
9.4 Consequences of Equivalence ........................................ 68

**Learning With Queries** ........................................ 69

10.1 Introduction ........................................ 69
10.2 General Learning Algorithms ........................................ 70
10.2.1 Exhaustive Search ........................................ 70
10.2.2 The Haicing Algorithm ........................................ 70
10.3 Relationship Between Exact Identification and PAC Learnability ........................................ 71
10.4 Examples of Exactly Identifiable Classes ........................................ 72
10.4.1 $k$-CNF and $k$-DNF Formulas ........................................ 73
10.4.2 Monotone DNF Formulas ........................................ 73
10.4.3 Other Efficiently Learnable Classes ........................................ 75

**Learning Horn Sentences** ........................................ 77

11.1 Introduction ........................................ 77
11.2 Preliminaries ........................................ 77
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.3 The Algorithm</td>
<td>78</td>
</tr>
<tr>
<td>11.4 Example Run of Algorithm</td>
<td>80</td>
</tr>
<tr>
<td>11.5 Analysis of Algorithm</td>
<td>81</td>
</tr>
<tr>
<td><strong>Learning with Abundant Irrelevant Attributes</strong></td>
<td>83</td>
</tr>
<tr>
<td>12.1 Introduction</td>
<td>83</td>
</tr>
<tr>
<td>12.2 On-line Learning Model</td>
<td>83</td>
</tr>
<tr>
<td>12.3 Definitions and Notation</td>
<td>85</td>
</tr>
<tr>
<td>12.4 Halving Algorithm</td>
<td>85</td>
</tr>
<tr>
<td>12.5 Standard Optimal Algorithm</td>
<td>86</td>
</tr>
<tr>
<td>12.6 The Linear Threshold Algorithm: WINNOW1</td>
<td>88</td>
</tr>
<tr>
<td>12.7 Extensions: WINNOW2</td>
<td>91</td>
</tr>
<tr>
<td>12.8 Transformations to Other Concept Classes</td>
<td>92</td>
</tr>
<tr>
<td><strong>Learning Regular Sets</strong></td>
<td>95</td>
</tr>
<tr>
<td>13.1 Introduction</td>
<td>95</td>
</tr>
<tr>
<td>13.2 The Learning Algorithm</td>
<td>95</td>
</tr>
<tr>
<td>13.3 An Example</td>
<td>97</td>
</tr>
<tr>
<td>13.4 Algorithm Analysis</td>
<td>102</td>
</tr>
<tr>
<td>13.4.1 Correctness of $L^*$</td>
<td>102</td>
</tr>
<tr>
<td>13.4.2 Termination of $L^*$</td>
<td>104</td>
</tr>
<tr>
<td>13.4.3 Runtime complexity of $L^*$</td>
<td>105</td>
</tr>
<tr>
<td><strong>Results on Learnability and the VC-Dimension</strong></td>
<td>107</td>
</tr>
<tr>
<td>14.1 Introduction</td>
<td>107</td>
</tr>
<tr>
<td>14.2 Dynamic Sampling</td>
<td>108</td>
</tr>
<tr>
<td>14.3 Learning Enumerable Concept Classes</td>
<td>108</td>
</tr>
<tr>
<td>14.4 Learning Decomposable Concept Classes</td>
<td>109</td>
</tr>
<tr>
<td>14.5 Reducing the Number of Stages</td>
<td>111</td>
</tr>
<tr>
<td><strong>Learning in the Presence of Malicious Noise</strong></td>
<td>113</td>
</tr>
<tr>
<td>15.1 Introduction</td>
<td>113</td>
</tr>
<tr>
<td>15.2 Definitions and Notation</td>
<td>113</td>
</tr>
<tr>
<td>15.3 Upper Bounds on $E_M(C)$</td>
<td>114</td>
</tr>
<tr>
<td>15.4 Generalization of Occam’s Razor</td>
<td>116</td>
</tr>
<tr>
<td>15.5 Using Positive and Negative Examples to Improve Learning Algorithms</td>
<td>117</td>
</tr>
<tr>
<td>15.6 Relationship between Learning Monomials with Errors and Set Cover</td>
<td>119</td>
</tr>
<tr>
<td><strong>Inferring Graphs from Walks</strong></td>
<td>123</td>
</tr>
<tr>
<td>16.1 Introduction</td>
<td>123</td>
</tr>
<tr>
<td>16.2 Intuition</td>
<td>124</td>
</tr>
<tr>
<td>16.3 Preliminaries</td>
<td>125</td>
</tr>
<tr>
<td>16.4 Inferring Linear Chains from End-to-end Walks</td>
<td>127</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.5 Inferring Linear Chains from General Walks</td>
<td>130</td>
</tr>
<tr>
<td>16.6 Inferring Cycles from Walks</td>
<td>132</td>
</tr>
<tr>
<td><strong>Learning in an Infinite Attribute Space</strong></td>
<td>137</td>
</tr>
<tr>
<td>17.1 Introduction and Description of Model</td>
<td>137</td>
</tr>
<tr>
<td>17.2 Learning Monotone Disjunctions</td>
<td>138</td>
</tr>
<tr>
<td>17.3 Learning Monotone k-CNF</td>
<td>138</td>
</tr>
<tr>
<td>17.4 Learning Non-Monotone k-CNF</td>
<td>139</td>
</tr>
<tr>
<td>17.5 Generalized Halving Algorithm</td>
<td>140</td>
</tr>
<tr>
<td>17.6 Other Topics</td>
<td>141</td>
</tr>
<tr>
<td><strong>The Weighted Majority Algorithm</strong></td>
<td>143</td>
</tr>
<tr>
<td>18.1 Introduction</td>
<td>143</td>
</tr>
<tr>
<td>18.2 Weighted Majority Algorithm</td>
<td>144</td>
</tr>
<tr>
<td>18.3 Analysis of Weighted Majority Algorithm</td>
<td>145</td>
</tr>
<tr>
<td>18.4 Variations on Weighted Majority</td>
<td>147</td>
</tr>
<tr>
<td>18.4.1 Shifting Target</td>
<td>147</td>
</tr>
<tr>
<td>18.4.2 Selection from an Infinite Pool</td>
<td>148</td>
</tr>
<tr>
<td>18.4.3 Pool of Functions</td>
<td>148</td>
</tr>
<tr>
<td>18.4.4 Randomized Responses</td>
<td>149</td>
</tr>
<tr>
<td><strong>Implementing the Halving Algorithm</strong></td>
<td>151</td>
</tr>
<tr>
<td>19.1 The Halving Algorithm and Approximate Counting</td>
<td>151</td>
</tr>
<tr>
<td>19.2 Learning a Total Order</td>
<td>153</td>
</tr>
<tr>
<td><strong>Choice of Topics</strong></td>
<td>157</td>
</tr>
<tr>
<td><strong>Homework Assignment 1</strong></td>
<td>161</td>
</tr>
<tr>
<td><strong>Homework Assignment 2</strong></td>
<td>163</td>
</tr>
<tr>
<td><strong>Homework Assignment 3</strong></td>
<td>165</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>167</td>
</tr>
</tbody>
</table>
1.1 Course Overview

Building machines that learn from experience is an important research goal of artificial intelligence, and has therefore been an active area of research. Most of the work in machine learning is empirical research. In such research, learning algorithms typically are judged by their performance on sample data sets. Although these ad hoc comparisons may provide some insight, it is difficult to compare two learning algorithms carefully and rigorously, or to understand in what situations a given algorithm might perform well, without a formally specified learning model with which the algorithms may be evaluated.

Recently, considerable research attention has been devoted to the theoretical study of machine learning. In computational learning theory one defines formal mathematical models of learning that enable rigorous analysis of both the predictive power and the computational efficiency of learning algorithms. The analysis made possible by these models provides a framework in which to design algorithms that are provably more efficient in both their use of time and data.

During the first half of this course we will cover the basic results in computational learning theory. This portion will include a discussion of the distribution-free (or PAC) learning model, the model of learning with queries, and the mistake-bound (or on-line) learning model. The primary goal is to understand how these models relate to one another and what classes of concepts are efficiently learnable in the various models. Thus we will present efficient algorithms for learning various concept classes under each model. (And in some cases we will consider what can be done if the computation time is not restricted to be polynomial.) In contrast to these positive results we present hardness results for some concept classes indicating that no efficient learning algorithm exists. In addition to studying the basic noise-free versions of these learning models, we will also discuss various models of noise and techniques for designing algorithms that are robust against noise. Finally, during the second half of this course we will study a selection of topics that follow up on the material presented during the first half of the course. These topics were selected by the students, and are just a sample of the types of other results that have been obtained. We warn the reader that this course only covers a small portion of the models, learning techniques, and methods for proving hardness results that are currently available in the literature.
1.2 Introduction

In this section we give a very basic overview of the area of computational learning theory. Portions of this introduction are taken from Chapter 2 of Goldman's thesis [18]. Also see Chapter 2 of Kearns' thesis [27] for additional definitions and background material.

1.2.1 A Research Methodology

Before describing formal models of learning, it is useful to outline a research methodology for applying the formalism of computational learning theory to "real-life" learning problems. There are four steps to the methodology.

1. Precisely define the problem, preserving key features while simplifying as much as possible.

2. Select an appropriate formal learning model.

3. Design a learning algorithm.

4. Analyze the performance of the algorithm using the formal model.

In Step 2, selecting an appropriate formal learning model, there are a number of questions to consider. These include:

- What is being learned?
- How does the learner interact with the environment? (e.g., Is there a helpful teacher? An adversary?)
- What is the prior knowledge of the learner?
- How is the learner's hypothesis represented?
- What are the criteria for successful learning?
- How efficient is the learner in time, data and space?

It is critical that the model chosen accurately reflect the real-life learning problem.
1.2.2 Definitions

In this course, we consider a restricted type of learning problem called concept learning. In a concept learning problem there are a set of instances and a single target concept that classifies each instance as a positive or a negative instance. The instance space denotes the set of all instances that the learner may see. The concept space denotes the set of all concepts from which the target concept can be chosen. The learner's goal is to devise a hypothesis of the target concept that accurately classifies each instance as positive or negative.

For example, one might wish to teach a child how to distinguish chairs from other furniture in a room. Each item of furniture is an instance; the chairs are positive instances and all other items are negative instances. The goal of the learner (in this case the child) is to develop a rule for the concept of a chair. In our models of learning, one possibility is that the rules are Boolean functions of features of the items presented, such as has-four-legs or is-wooden.

Since we want the complexity of the learning problem to depend on the "size" of the target concept, often we assign to it some natural size measure $n$. If we let $X_n$ denote the set of instances to be classified for each problem of size $n$, we say that $X = \bigcup_{n \geq 1} X_n$ is the instance space, and each $x \in X$ is an instance. For each $n \geq 1$, we define each $C_n \subseteq 2^{X_n}$ to be a family of concepts over $X_n$, and $C = \bigcup_{n \geq 1} C_n$ to be a concept class over $X$. For $c \in C_n$ and $x \in X_n$, $c(x)$ denotes the classification of $c$ on instance $x$. That is, $c(x) = 1$ if and only if $x \in c$. We say that $x$ is a positive instance of $c$ if $c(x) = 1$ and $x$ is a negative instance of $c$ if $c(x) = 0$. Finally, a hypothesis $h$ for $C_n$ is a rule that given any $x \in X_n$ outputs in polynomial time a prediction for $c(x)$. The hypothesis space is the set of all hypotheses $h$ that the learning algorithm may output. The hypothesis must make a prediction for each $x \in X_n$. For the learning models which we will study here, it is not acceptable for the hypothesis to answer "I don't know" for some instances.

To illustrate these definitions, consider the concept class of monomials. (A monomial is a conjunction of literals, where each literal is either some boolean variable or its negation.) For this concept class, $n$ is the number of variables. Thus $|X_n| = 2^n$ where each $x \in X_n$ represents an assignment of 0 or 1 to each variable. Observe that each variable can be placed in the target concept unnegated, placed in the target concept negated, or not put in the target concept at all. Thus, $|C_n| = 3^n$. One possible target concept for the class of monomials over five variables is $x_1 \overline{x_4} x_5$. For this target concept, the instance "10001" is a positive instance and "00001" is a negative instance.

We will model the learning process as consisting of two phases, a training phase and a performance phase. In the training phase the learner is presented with labeled examples (i.e., an example is chosen from the instance space and labeled according to the target concept). Based on these examples the learner must devise a hypothesis of the target concept. In the performance phase the hypothesis is used to classify examples from the instance space, and the accuracy of the hypothesis is evaluated. The various models of learning will differ primarily in the way they allow the learner to interact with the environment during the training phase and how the hypothesis is evaluated.
1.3 The Distribution-Free (PAC) Model

The first formal model of machine learning we shall consider is the distribution-free or PAC model introduced by Valiant [43] in 1984. (This work initiated the field of computational learning theory.) In this model, an adversary chooses a concept class $C$, a target concept $c \in C$ and an arbitrary distribution $D$ on the instance space. (We note that absolutely no restrictions are placed on $D$.) The learner is presented with the concept class $C$, an accuracy bound $\epsilon$ and a confidence bound $\delta$. The learner is required to formulate a hypothesis $h$ of the target concept based on labeled examples drawn randomly from the distribution $D$ (which is unknown to the learner). See Figure 1.1.\footnote{Compliments of Andy Fingerhut.}

The PAC model requires the learner to produce a hypothesis which meets a certain error criteria. We can define the error of the hypothesis $h$ on target concept $c$ under distribution $D$ to be:

$$error_D(h) = \Pr[c \oplus h] = \sum_{c(x) \neq h(x)} \Pr[x]$$

where $c \oplus h$ is the symmetric difference between $c$ and $h$ (i.e., the instances for which $c$ and $h$ differ). The error is the sum of the weight under distribution $D$ placed on the examples for which $c$ and $h$ differ.

The goals of the learner are as follows:

1. With high probability ($\geq 1 - \delta$) the hypothesis must give a good approximation ($error_D(h) \leq \epsilon$) of the target concept.

2. The time and sample complexity of the learning algorithm must be polynomial in the size of the target concept, $1/\epsilon$ and $1/\delta$. (The sample complexity is the number of labeled examples needed by the algorithm.) Observe that as $\epsilon$ and $\delta$ go down, the algorithm is allowed more time and samples to produce a hypothesis.
This model is called distribution-free because the distribution on the examples is unknown to the learner. Because the hypothesis must have high probability of being approximately correct, it is also called the PAC model.

1.4 Learning Monomials in the PAC Model

We would like to investigate various concept classes that can be learned efficiently in the PAC model. The concept class of monomials is one of the simplest to learn and analyze in this model. Furthermore, we will see that the algorithm for learning monomials can be used to learn more complicated concept classes such as k-CNF.

A monomial is a conjunction of literals, where each literal is a variable or its negation. In describing the algorithm for learning monomials we will assume that \( n \), the number of variables, is known. (If not, then it can be determined simply by counting the number of bits in the first example.)

We now describe the algorithm given by Valiant [43] for learning monomials. The algorithm is based on the idea that a positive example gives significant information about the monomial being learned. For example, if \( n = 5 \), and we see a positive example “10001”, then we know that the monomial does not contain \( \overline{x_1}, x_2, x_3, x_4 \) or \( \overline{x_5} \). A negative example does not give us as much information since we do not know which of the bits caused the example to violate the target monomial.

**Algorithm Learn-Monomials\((n, \epsilon, \delta)\)**

1. Initialize the hypothesis to the conjunction of all \( 2n \) literals.
   
   \[ h = x_1 \overline{x_1} x_2 \overline{x_2} \cdots x_n \overline{x_n} \]

2. Make \( m = 1/\epsilon(n \ln 3 + \ln 1/\delta) \) calls to EX.
   
   - For each positive instance, remove \( x_i \) from \( h \) if \( x_i = 0 \) and remove \( \overline{x_i} \) from \( h \) if \( x_i = 1 \).
   - For each negative instance, do nothing.

3. Output the remaining hypothesis.

In analyzing the algorithm, there are three measures of concern. First, is the number of examples used by the algorithm polynomial in \( n, 1/\epsilon \) and 1/\( \delta \)? The algorithm uses \( m = 1/\epsilon(n \ln 3 + \ln 1/\delta) \) examples; clearly this is polynomial in \( n, 1/\epsilon \) and 1/\( \delta \). Second, does the algorithm take time polynomial in these three parameters? The time taken per example is constant, so the answer is yes. Third is the hypothesis sufficiently accurate by the criteria of the PAC model? In other words, is \( \Pr[\text{error}_D(h) \leq \epsilon] \geq (1 - \delta) \)?

To answer the third question we first show that the final hypothesis output by the algorithm is consistent with all \( m \) examples seen in training. That is, the hypothesis correctly
classifies all of these examples. We will also show that the hypothesis logically implies the target concept. This means that the hypothesis does not classify any negative example as positive. We can prove this by induction on the number of examples seen so far. We initialize the hypothesis to the conjunction of all $2n$ literals, which is logically equivalent to classifying every example as false. This is trivially consistent with all examples seen, since initially no examples have been seen. Also, it trivially implies the target concept since false implies anything. Let $h_i$ be the hypothesis after $i$ examples. Assuming the hypothesis $h_k$ is consistent with the first $k$ examples and implies the target concept, we show the hypothesis $h_{k+1}$ is consistent with the first $k+1$ examples and still implies the target concept. If example $k + 1$ is negative, $h_{k+1} = h_k$. Since $h_k$ implies the target concept, it does not classify any negative example as positive. Therefore $h_{k+1}$ correctly classifies the first $k + 1$ examples and implies the target concept. If example $k + 1$ is positive, we alter $h_k$ to include this example within those classified as positive. Clearly $h_{k+1}$ correctly classifies the first $k + 1$ examples. In addition, it is not possible for some negative example to satisfy $h_{k+1}$, so this new hypothesis logically implies the target concept.

To analyze the error of the hypothesis, we define an $\varepsilon$-bad hypothesis $h'$ as one with $\text{error}(h') > \varepsilon$. We satisfy the PAC error criteria if the final hypothesis (which we know is consistent with all $m$ examples seen in training) is not $\varepsilon$-bad. By the definition of $\varepsilon$-bad,

$$\Pr[\text{an } \varepsilon\text{-bad hyp is consistent with } 1 \text{ ex}] \leq 1 - \varepsilon$$

and since each example is taken independently,

$$\Pr[\text{an } \varepsilon\text{-bad hyp is consistent with } m \text{ exs}] \leq (1 - \varepsilon)^m.$$  

Since the hypothesis comes from $C$, the maximum number of hypotheses is $|C|$. Thus,

$$\Pr[\exists \text{ an } \varepsilon\text{-bad hyp consistent with } m \text{ exs}] \leq |C|(1 - \varepsilon)^m.$$  

Now, we require that $\Pr[h \text{ is } \varepsilon\text{-bad}] \leq \delta$, so we must choose $m$ to satisfy

$$|C|(1 - \varepsilon)^m \leq \delta.$$  

Solving for $m$,

$$m \geq \frac{\ln |C| + \ln 1/\delta}{-\ln(1 - \varepsilon)}.$$  

Using the Taylor series expansion for $e^x$

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots > 1 + x,$$

letting $x = -\varepsilon$ and taking $\ln$ of both sides we can infer that $\varepsilon < -\ln(1 - \varepsilon)$. Also, $|C| = 3^n$ since each of the variables may appear in the monomial negated, unnegated, or not appear at all. So if $m \geq 1/\varepsilon(n \ln 3 + \ln 1/\delta)$ then $\Pr[\text{error}(h) > \varepsilon] \leq \delta$.

It is interesting to note that only $O(n \ln 3)$ examples are required by the algorithm (ignoring dependence on $\varepsilon$ and $\delta$) even though there are $2^n$ examples to classify. Also, the analysis of the number of examples required can be applied to any algorithm which finds a hypothesis consistent with all the examples seen during training. Observe, this is only an upper bound. In some cases a tighter bound on the number of examples may be achievable.
1.5 Learning $k$-CNF and $k$-DNF

We now describe how to extend this result for monomials to the more expressive classes of $k$-CNF and $k$-DNF. The class $k$-Conjunctive Normal Form, denoted $k$-CNF$_n$, consists of all Boolean formulas of the form $C_1 \land C_2 \land \ldots \land C_l$ where each clause $C_i$ is the disjunction of at most $k$ literals over $x_1, \ldots, x_n$. We assume $k$ is some constant. We will often omit the subscript $n$. If $k = n$ then the class $k$-CNF consists of all CNF formulas. The class of monomials is equivalent to 1-CNF.

The class $k$-Disjunctive Normal Form, denoted $k$-DNF$_n$ consists of all Boolean formulas of the form $T_1 \lor T_2 \lor \ldots \lor T_l$ where each term $T_i$ is the conjunction of at most $k$ literals over $x_1, \ldots, x_n$.

There is a close relationship between $k$-CNF and $k$-DNF. By DeMorgan’s law

$$f = C_1 \land C_2 \land \ldots \land C_l \Rightarrow \overline{f} = T_1 \lor T_2 \lor \ldots \lor T_l$$

where $T_i$ is the conjunction of the negation of the literals in $C_i$. For example if

$$f = (a \lor b \lor c) \land (d \lor e)$$

then by DeMorgan’s law

$$\overline{f} = (\overline{a} \land b \land c) \lor (\overline{d} \lor e).$$

Finally, applying DeMorgan’s law once more we get that

$$\overline{\overline{f}} = (\overline{a} \land b \land c) \lor (d \land e).$$

Thus the classes $k$-CNF and $k$-DNF are duals of each other in the sense that exchanging $\land$ for $\lor$ and complementing each variable gives a transformation between the two representations. So an algorithm for $k$-CNF can be used for $k$-DNF (and vice versa) by swapping the use of positive and negative examples, and negating all the attributes.

We now describe a general technique for modifying the set of variables in the formula and apply this technique to generalize the monomial algorithm given in the previous section to learn $k$-CNF. The idea is that we define a new set of variables, one for each possible clause. Next we apply our monomial algorithm using this new variable set. To do this we must compute the value for each new variable given the value for the original variables. However, this is easily done by just evaluating the corresponding clause on the given input. Finally, it is straightforward to translate the hypothesis found by the monomial algorithm into a $k$-CNF formula using the correspondence between the variables and the clauses.

We now analyze the complexity of the above algorithm for learning $k$-CNF. Observe that the number of possible clauses is upper bounded by:

$$\sum_{i=1}^{k} \binom{2n}{k} = O(n^k).$$

This overcounts the number of clauses since it allows clauses containing both $x_i$ and $\overline{x}_i$. The high order terms of the summation dominate. Since $\binom{2n}{k} < (2n)^k$, the number of clauses is $O(n^k)$ which is polynomial in $n$ for any constant $k$. Thus the time and sample complexity of our $k$-CNF algorithm are polynomial.

15
2.1 Model Definition

Here we consider a variant of the single button PAC model described in the previous notes. The material presented in this lecture is just one portion of the paper, “Equivalence of Models for Polynomial Learnability,” by David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth [21]. Rather than pushing a single button to receive an example drawn randomly from the distribution \( D \), a variation of this model has two buttons, one for positive examples and one for negative examples. (In fact, the this is the model originally introduced by Valiant.) There are two distributions, one on the positive examples and one on the negative examples. When the positive button is pushed a positive example is drawn randomly from the distribution \( D^+ \). When the negative button is pushed a negative example is drawn randomly from the distribution \( D^- \). A learning algorithm in the two-button model is required to be accurate within the confidence bound for both the positive and negative distributions. Formally, we require

\[
\Pr[\text{error}^+(h) \equiv \Pr_{D^+}(c \oplus h) \leq \varepsilon] \geq 1 - \delta
\]

\[
\Pr[\text{error}^-(h) \equiv \Pr_{D^-}(c \oplus h) \leq \varepsilon] \geq 1 - \delta.
\]

Offering two buttons allows the learning algorithm to choose whether a positive or negative example will be seen next. It should be obvious that this gives the learner at least as much power as in the one button model. In fact, we will show that the one-button and two-button models are equivalent in the concept classes they can learn efficiently.

2.2 Equivalence of One-Button and Two-Button Models

In this section we show that one-button PAC-learnability is equivalent to two-button PAC-learnability.

**Theorem 2.1** One-button PAC-learnability (1BL) is equivalent to two-button PAC-learnability (2BL).
Proof:
Case 1: 1BL \Rightarrow 2BL.

Let \( c \in C \) be a target concept over an instance space \( X \) that we can learn in the one-button model using algorithm \( A_1 \). We show that there is an algorithm \( A_2 \) for learning \( c \) in the two-button model. Let \( D^+ \) and \( D^- \) be the distributions over the positive and negative examples respectively. The algorithm \( A_2 \) is as follows:

1. Run \( A_1 \) with inputs \( n, \varepsilon/2, \delta \).
2. When \( A_1 \) requires an example, flip a fair coin. If the outcome is heads, push the positive example button and give the resulting example to \( A_1 \). If the outcome is tails, push the negative example button and give the resulting example to \( A_1 \).
3. When \( A_1 \) terminates, return the hypothesis \( h \) found by \( A_1 \).

The use of the fair coin to determine whether to give \( A_1 \) a positive or negative example results in a distribution \( D \) seen by \( A_1 \) defined by,

\[
D(x) = \frac{1}{2}D^+(x) + \frac{1}{2}D^-(x).
\]

Let \( e \) be the error of \( h \) on \( D \), \( e^+ \) be the error of \( h \) on \( D^+ \) and \( e^- \) be the error of \( h \) on \( D^- \). Then

\[
e = e^+/2 + e^-/2.
\]

Since \( A_1 \) is an algorithm for PAC learning, it satisfies the error criteria. That is,

\[
\Pr[e \leq \varepsilon/2] \geq 1 - \delta.
\]

Since \( e \geq e^+/2 \) and \( e \geq e^-/2 \) we can conclude that

\[
\Pr[e^+/2 \leq \varepsilon/2] \geq 1 - \delta \quad \text{and} \quad \Pr[e^-/2 \leq \varepsilon/2] \geq 1 - \delta.
\]

Therefore, \( e^+ \leq \varepsilon \) and \( e^- \leq \varepsilon \) each with probability \( \geq 1 - \delta \), and so algorithm \( A_2 \) satisfies the accuracy criteria for the two-button model. Notice that we had to run \( A_1 \) with an error bound of \( \varepsilon/2 \) in order to achieve an error bound of \( \varepsilon \) in \( A_2 \).

Case 2: 2BL \Rightarrow 1BL.

Let \( c \in C \) be a target concept over instance space \( X \) that we can learn in the two-button model using algorithm \( A_2 \). We show that there is an algorithm \( A_1 \) for learning \( c \) in the one-button model.

The idea behind the algorithm is that \( A_1 \) will draw some number of examples from \( D \) initially and store them in two bins, one bin for positive examples and one bin for negative examples. Then \( A_1 \) will run \( A_2 \). When \( A_2 \) requests an example, \( A_1 \) will supply one from the appropriate bin. Care will need to be exercised because there may not be enough of one type of example to run \( A_2 \) to completion. Let \( m \) be the total number of examples (of both types) needed to run \( A_2 \) to completion with parameters \( n, \varepsilon, \delta/3 \). Algorithm \( A_1 \) is as follows:
1. Make \( q \) calls to EX and store the positive examples in one bin and the negative examples in another bin, where
\[
q = \max \left\{ \frac{2}{\epsilon} m, \frac{8}{\epsilon} \ln \frac{3}{\delta} \right\}.
\]

2. If the number of positive examples is \( < m \) then output the hypothesis false. (This hypothesis classifies all instances as negative.)

3. Else if the number of negative examples is \( < m \) then output the hypothesis true. (This hypothesis classifies all instances as positive.)

4. Else there are enough positive and negative examples, so run \( A_2 \) to completion with parameters \( (n, \min(\epsilon/2, \delta/3)) \). Output the hypothesis returned by \( A_2 \).

For now this choice of \( q \) seems to have appeared out of thin air. The rationale for the choice will become clear in the analysis of the algorithm. Before continuing the analysis, we need an aside to discuss Chernoff Bounds, which will be needed in the proof.

**Aside: Chernoff Bounds**

Chernoff Bounds are formulas which bound the area under the tails of the binomial distribution. We now describe some of the bounds cited in the literature. Much of this discussion is taken directly from Sloan [40]. Normally we are trying to say that if we run \( m \) Bernoulli trials each with probability of success \( p \), then the chance of getting a number of successes very much different from \( pm \) is exponentially vanishing.

Formally, let \( X_1, X_2, \ldots X_m \) be independent Boolean random variables each with probability of \( p \) \((0 \leq p \leq 1)\) of being 1. We now define a random variable \( S = \sum_{i=1}^{m} X_i \). Clearly the expectation of \( S \) is \( pm \).

Define \( LE(p, m, r) = \Pr[S \leq r] \) (i.e. the probability of at most \( r \) successes in \( m \) independent trials of a Bernoulli random variable with probability of success \( p \)). Let \( GE(p, m, r) = \Pr[S \geq r] \) (i.e. the probability of at least \( r \) successes in \( m \) independent trials of a Bernoulli random variable with probability of success \( p \)). So \( LE(p, m, r) \) bounds the area in the tail at the low end of the binomial distribution, while \( GE(p, m, r) \) bounds the area in the tail at the high end of the binomial distribution.

Hoeffding's Inequality [22] states that:

\[
\Pr[S \geq pm + t] \leq e^{-2mt^2} \quad (2.1)
\]
\[
\Pr[S \geq \alpha m], \Pr[S \leq \alpha m] \leq e^{-2m(\alpha - p)^2} \quad (2.2)
\]

where it is understood that in Equation (2.2) the first \( \alpha \) must be at least \( p \) and the second \( \alpha \) must be at most \( p \).
The above bound is as good or better than any of the others in the literature except for the case when \( p < 1/4 \). In this case the following bounds given by Angluin and Valiant [8] are better:

\[
LE(p, m, (1 - \alpha)pm) \leq e^{-\alpha^2mp/2} \tag{2.3}
\]
\[
GE(p, m, (1 + \alpha)pm) \leq e^{-\alpha^2mp/3}. \tag{2.4}
\]

where \( 0 \leq \alpha \leq 1 \). Note that in Equation (2.3), \( r = (1 - \alpha)pm \leq pm \). And in Equation (2.4), \( r = (1 + \alpha)pm \geq pm \).

We now return to the analysis of algorithm \( A_1 \). Let \( p^+ \) denote the probability that we draw a positive example from \( D \), and \( p^- \) denote the probability that we draw a negative example from \( D \). To analyze the accuracy of algorithm \( A_1 \) we consider four cases based on the probabilities \( p^+ \) and \( p^- \). For each case we will show that \( \Pr[error(h) \leq \epsilon] \geq 1 - \delta \).

Subcase A: \( p^+ \geq \epsilon, p^- \geq \epsilon \).

There are three things which may happen in algorithm \( A_1 \). First, we may not have enough of one type of example and thus will output a hypothesis of true or false. Notice that if this occurs then \( \Pr[error_D(h) \leq \epsilon] = 0 \) since \( p^+, p^- \geq \epsilon \). Second, we may have enough examples to run \( A_2 \) to completion and receive an \( \epsilon \)-bad hypothesis from \( A_2 \). Third, we may have enough examples to run \( A_2 \) to completion receive an \( \epsilon \)-good hypothesis from \( A_2 \). The first two outcomes are undesirable. We must make sure they occur with probability at most \( \delta \).

Let us determine the probability that we do not have enough of one type of example. Consider the positive examples. The probability of drawing a positive example is \( p^+ \geq \epsilon \). We are interested in the probability that we draw fewer than \( m \) positive examples in \( q \) calls to \( EX \). This probability can be bounded by the Chernoff bound of Equation (2.3).

\[
\Pr[< m \text{ positive exs in } q \text{ calls to } EX] \leq LE(\epsilon, q, m)
\]

From Equation (2.3) we know that

\[
LE(p, m', (1 - \alpha)m'p) \leq e^{-\alpha^2m'p/2}.
\]

So, with \( p = \epsilon, m' = q = \frac{2}{\epsilon}m \) and \( \alpha = 1/2 \) we obtain

\[
LE(\epsilon, q, m) \leq e^{-m/4}.
\]

Finally, we require that this bad event occurs with probability at most \( \delta/3 \). That is, we must have

\[
e^{-m/4} \leq \delta/3.
\]

Solving for \( m \) yields

\[
m \geq 4 \ln \frac{3}{\delta} \Rightarrow q \geq \frac{8}{\epsilon} \ln \frac{3}{\delta}
\]

20
which is satisfied by the choice of $q$ in the algorithm.

The same analysis holds for bounding the probability that there are fewer than $m$ negative examples in $q$ calls to EX. That is, we know

$$\Pr[< m \text{ negative exs in } q \text{ calls to EX}] \leq \delta/3$$

$$\Pr[< m \text{ positive exs in } q \text{ calls to EX}] \leq \delta/3.$$ 

This implies that

$$\Pr[\geq m \text{ positive exs and } \geq m \text{ negative exs in } q \text{ calls to EX}] \geq 1 - 2\delta/3.$$ 

If we have enough positive and negative examples then we will run algorithm $A_2$ to completion. With probability $\geq 1 - \delta/3$ algorithm $A_2$ will return a hypothesis with $e^+ \leq \epsilon$ and $e^- \leq \epsilon$. This implies that the hypothesis $h$ returned by $A_2$ satisfies:

$$error_D(h) = p^+e^+ + p^-e^-$$

$$= p^+e^+ + (1 - p^+)e^-$$

$$\leq p^+\epsilon + (1 - p^+)\epsilon$$

$$= \epsilon$$

We have determined the following probabilities of each of the two bad outcomes that can occur when running $A_1$.

- With probability $\leq 2\delta/3$ we do not have enough of one type of example to run $A_2$.

- The probability that we run $A_2$ and it returns a bad hypothesis is given by the product of the probability that we have enough examples and the probability $A_2$ returns a bad hypothesis. This probability is at most $(1 - \frac{2\delta}{3})\frac{\delta}{3} \leq \delta/3$.

In all other cases, the hypothesis output will be $\epsilon$-good. Combining these probabilities yields the following expression for the probability that the good outcome occurs.

$$\Pr[error_D(h) \leq \epsilon] \geq 1 - \left(\frac{2\delta}{3} + \frac{\delta}{3}\right) = 1 - \delta$$

Subcase B: $p^+ < \epsilon$, $p^- \geq \epsilon$.

In this case, we have a good chance of getting more than $m$ negative examples and less than $m$ positive examples. If this occurs algorithm $A_1$ will output a hypothesis of false. Since $p^+ < \epsilon$, a hypothesis $h$ of false satisfies $Pr[error_D(h) \leq \epsilon] = 1$. This is good. It is also possible (although less likely) that we will not get enough negative examples, or that we will get enough of each kind of example to run $A_2$. In running $A_2$ we may get a hypothesis which has error $> \epsilon$. We must ensure that these bad things occur with probability at most $\delta$. 

21
More formally, we consider two possible bad situations. First, we may draw less than $m$ negative examples. In Subcase A we showed that this will occur with probability $\leq \delta/3$. In this situation we will draw enough positive examples and thus output a hypothesis $h$ of true. For this hypothesis, $\Pr[\text{error}_D(h) > \epsilon] = 1$. In the second situation, we draw at least $m$ negative examples. This occurs with probability $\geq 1 - \delta/3$. In the worst case we also draw at least $m$ positive examples and thus run $A_2$. We know that the hypothesis $h$ returned by $A_2$ will satisfy $\Pr[\text{error}_D(h) \leq \epsilon] \geq 1 - \delta/3$. This implies that $\Pr[\text{error}_D(h) > \epsilon] \leq \delta/3$. Combining these two results we have

$$\Pr[\text{error}_D(h) > \epsilon] \leq \delta/3 + \delta/3$$

which implies that

$$\Pr[\text{error}_D(h) \leq \epsilon] \geq 1 - 2\delta/3.$$

Subcase C: $p^- < \epsilon$, $p^+ \geq \epsilon$.

This case is the same as Subcase B but with the roles of positive and negative examples interchanged. By a similar analysis,

$$\Pr[\text{error}_D(h) \leq \epsilon] \geq 1 - 2\delta/3.$$

Subcase D: $p^+ < \epsilon$, $p^- < \epsilon$.

Since $\epsilon \leq 1/2$, this case cannot occur.

These four subcases cover the behavior of algorithm $A_1$ for all possible values of $p^+$ and $p^-$. Thus, $2\text{BL} \Rightarrow 1\text{BL}$. Taken with Case 1 proving $1\text{BL} \Rightarrow 2\text{BL}$ we have shown that one-button PAC learnability is equivalent to two-button PAC learnability.

\[\square\]
3.1 Introduction

In this lecture, we shall discuss the learnability of \( k \)-term-DNF formulas. Most of the material presented in this lecture comes from the paper "Computational Limitations on Learning from Examples," by Leonard Pitt and Leslie Valiant [33].

When introducing the PAC model, we showed that both \( k \)-CNF and \( k \)-DNF are PAC learnable using a hypothesis space of \( k \)-DNF and \( k \)-CNF respectively. Does a similar result hold for \( k \)-term-DNF and \( k \)-clause-CNF? We first show that \( k \)-term-DNF is not PAC learnable by \( k \)-term-DNF in polynomial time, unless \( \text{RP} = \text{NP} \). In fact, this hardness result holds even for learning monotone \( k \)-term-DNF by \( k \)-term-DNF. Likewise, \( k \)-clause-CNF is not PAC learnable by \( k \)-clause-CNF in both the monotone and unrestricted case. Contrasting these negative results, we then describe an efficient algorithm to learn \( k \)-term-DNF by \( k \)-CNF. As a dual result, we can learn \( k \)-clause-CNF using the hypothesis space of \( k \)-DNF. Finally, we explore the representational power of the concept classes that we have considered so far and the class of \( k \)-decision-lists [35].

Before describing the representation-dependent hardness result, we first give some basic definitions. The concept class of \( k \)-term-DNF is defined as follows:

**Definition 3.1** For any constant \( k \), the class of \( k \)-term-DNF formulas contains all disjunctions of the form \( T_1 \lor T_2 \lor \ldots \lor T_k \), where each \( T_i \) is monomial.

Up to now we have assumed that in the PAC model the hypothesis space available to the learner is equivalent to the concept class. That is, each element of the hypothesis space corresponds to the representation of an element of the concept space. However, in general one can talk about a concept class \( C \) being PAC learnable by \( H \) (possibly different from \( C \)). Formally, we say that \( C \) is PAC learnable by \( H \), if there exists a polynomial-time learning algorithm \( A \) such that for any \( c \in C \), any distribution \( D \) and any \( \epsilon, \delta \), \( A \) can output with probability at least \( 1 - \delta \) a hypothesis \( h \in H \) such that \( h \) has probability at most \( \epsilon \) of disagreeing with \( c \) on a randomly drawn instance from \( D \).

3.2 Representation-dependent Hardness Results

In this section, we will show that for \( k \geq 2 \), \( k \)-term-DNF is not PAC learnable using a hypothesis from \( k \)-term-DNF. This type of hardness result is representation-dependent since
it only holds if the learner's hypothesis class (or representation class) is restricted to be a
certain class. Note that when \( k = 1 \), the class of \( k \)-term-DNF formulas is just the class of
monomials, which we know is learnable using a hypothesis of a monomial.

We prove this hardness result by by reducing the learning problem to \( k \)-NM-Colorability,
a generalization of the Graph \( k \)-Colorability problem. Before defining this problem we first
describe two known NP-complete problems: Graph \( k \)-colorability (NP-complete for \( k \geq 3 \))
and Set Splitting. These descriptions come from Garey and Johnson [14].

**Problem:** Graph \( k \)-colorability

**Instance:** For a graph \( G = (V, E) \), with positive integer \( k \leq |V| \).

**Question:** Is \( G \) \( k \)-colorable? That is, does there exist a function \( f : V \to \{1, \ldots, k\} \), such
that \( f(u) \neq f(v) \) whenever \( \{u, v\} \in E \)?

**Problem:** Set Splitting

**Instance:** Collection \( C \) of subsets of a finite set \( S \).

**Question:** Is there a partition of \( S \) into two subsets \( S_1, S_2 \), such that no subset in \( C \) is
entirely contained in either \( S_1 \) or \( S_2 \)?

We now generalize both of these problems to obtain the \( k \)-NM-Colorability problem which
we use for our reduction.

**Problem:** \( k \)-NM-Colorability

**Instance:** A finite set \( S \) and a collection \( C = \{c_1, \ldots, c_m\} \) of constraints \( c_i \subseteq S \).

**Question:** Is there a \( k \)-coloring \( \chi \) of the elements of \( S \), such that for each constraints \( c_i \in C \),
the elements of \( c_i \) are not MONOCHROMATICALLY colored (i.e. \( \forall c_i \in C, \exists x, y \in c_i \), such
that \( \chi(x) \neq \chi(y) \))?  

We now argue that \( k \)-NM-Colorability is NP-complete. Clearly \( k \)-NM-Colorability is in
NP. Note that if every \( c_i \in C \) has size 2, then the \( k \)-NM-Colorability problem is simply the
Graph-\( k \)-Colorability problem. Since the Graph-\( k \)-Colorability is NP-complete for \( k \geq 3 \), we
only need to show that 2-NM-Colorability is NP-hard. However, note that 2-NM-Colorability
is exactly the Set Splitting problem which is NP-complete. Thus it follows that \( k \)-NM-
Colorability is NP-complete.

We now prove the main results of this section that \( k \)-term-DNF is not PAC learnable by
\( k \)-term-DNF.

**Theorem 3.1** For all integers \( k \geq 2 \), \( k \)-term DNF is not PAC learnable (in polynomial
time) using a hypothesis from \( k \)-term-DNF unless \( RP=NP \).

**Proof:** We reduce \( k \)-NM-coloring to the \( k \)-term-DNF learning problem. Let \( (S, C) \) be an
instance of \( k \)-NM-coloring, we construct a \( k \)-term-DNF learning problem as follows: Each
instance \( (S, C) \) will correspond to a particular \( k \)-term-DNF formula to be learned. If \( S = \{s_1, \ldots, s_n\} \), then we will create \( n \) variables \( \{x_1, \ldots, x_n\} \) for the learning problem. We now
describe the positive and negative examples, as well as the distributions \( D^+ \) and \( D^- \).

- The positive examples are \( \{\vec{p}_i\}_{i=1}^{n} \), where \( \vec{p}_i \) is the vector with \( x_i=0 \), and for \( j \neq i \),
$x_j=1$. Thus there are $n$ positive examples. Finally, let $D^+$ be uniform over these positive examples (i.e. each has weight $1/n$).

- The negative examples are $\{\bar{n}_i\}^{C}_{1=1}$, where for each constraint $c_i \in C$, if $c_i = \{s_{i_1}, \ldots, s_{i_m}\}$, then $\bar{n}_i = \vec{0}_{i_1, i_2, \ldots, i_m}$ (all elements of $S$ in $c_i$ are 0, the others are 1). For example, if the constraint $c_i$ is $\{s_1, s_3, s_8\}$, then the vector corresponding to is: $\bar{n}_i = <010111011\ldots>$. Finally, let $D^-$ be uniform over these negative examples (so each has weight $1/|C|$).

We now show that a $k$-term-DNF formula is consistent with all the positive and negative examples defined above if and only if $(S, C)$ is $k$-NM-colorable. Then we use this claim to show that the learning problem is solvable in polynomial time if and only if $\text{RP} = \text{NP}$.

**Claim 3.1** There is a $k$-term-DNF formula consistent with all positive and negative examples defined above if and only if $(S, C)$ is $k$-NM-colorable.

**Proof of Claim:**

$(\Leftarrow)$ Without loss of generality, assume that $(S, C)$ is $k$-NM-colorable by a coloring $\chi : S \rightarrow \{1, 2, \ldots, k\}$, which uses every color at least once. Let $f$ be the $k$-term-DNF formula $T_1 \lor T_2 \lor \ldots \lor T_k$, where

$$T_i = \bigwedge_{\chi(s_j) \neq i} x_j.$$ 

In other words, $T_i$ is the conjunction of all $x_j$ corresponding to $s_j$ that are not colored with $i$.

We now show that $f$ is consistent with positive examples. The positive example $\bar{p}_j$ ($x_j = 0, x_i = 1$ for all $i \neq j$) clearly satisfies the term $T_i$, where $\chi(s_j) = i$. Thus, $f$ is true for all positive examples.

Finally, we show that $f$ is consistent with the negative examples. Suppose some negative example, say $\bar{n}_i = \vec{0}_{i_1, \ldots, i_m}$ satisfies $f$, then $\bar{n}_i$ satisfies some term, say $T_j$. Then every element of constraint $c_i = \{s_{i_1}, \ldots, s_{i_m}\}$ must be colored $j$, (They are 0 in $\bar{n}_i$ and thus must not be in $T_j$, hence they are colored with $j$). But then $c_i$ is monochromatic, giving a contradiction.

$(\Rightarrow)$ Suppose $T_1 \lor \ldots \lor T_k$ is a $k$-term-DNF formula consistent with all positive examples and no negative examples. We now show that, without loss of generality, we can assume for all $i$, $T_i$ is a conjunction of positive literals.

**Case 1:** $T_i$ contains at least two negated variables. However, all positive examples have a single 0, so none could satisfy $T_i$. Thus just remove $T_i$.

**Case 2:** $T_i$ contains 1 negated variable $\overline{x_j}$. Then $T_i$ can only be satisfied by the single positive example $\bar{p}_j$. In this case, replace $T_i$ by $\overline{T_i} = \bigwedge_{j \neq i} x_j$, which is satisfied only by the vectors $\bar{p}_j$ and $\vec{1}$, neither of which are negative examples.
Thus we now assume that all terms are a conjunction of positive literals. Now color the elements of $S$ by the function: $\chi : S \to \{1, \ldots, k\}$, defined by $\chi(s_i) = \min \{j : x_i \text{ does not occur in } T_j\}$.

Now we show $\chi$ is well defined. Since each positive example $p_i$ satisfies $T_1 \lor \ldots \lor T_k$, it must satisfy some term $T_j$. But each term is a conjunct of unnegated literals. Thus for some $j$, $x_i$ must not occur in $T_j$. Thus each element of $S$ receives a color (which is clearly unique).

Finally we show that $\chi$ obeys the constraints. Suppose $\chi$ violates constraint $c_i$, then all of the elements in $c_i$ are colored by the same color, say $j$. By the definition of $\chi$, none of the literals corresponding to elements in $c_i$ occur in term $T_j$, so the negative example $\bar{n}_i$ associated with $c_i$ satisfies $T_j$. This contradicts the assumption that none of the negative examples satisfy the formula $T_1 \lor \ldots \lor T_k$. This completes the proof of the claim.

We now complete the proof of the theorem. Namely, we show how a learning algorithm for $k$-term-DNF can be used to decide $k$-$NM$-colorability in random polynomial time. First we give the definition of complexity class $RP$.

**Definition 3.2** A set $S$ is accepted in the random polynomial time (i.e. $S$ is in $RP$) if there exists a randomized algorithm $A$ such that on all inputs, $A$ is guaranteed to halt in polynomial time and, if $x \not\in S, A(x) = "no"$, if $x \in S, \Pr[A(x) = "yes"] \geq 1/2$.

Now we show that if there is a PAC learning algorithm for $k$-term-DNF, it can be used to decide $k$-$NM$-colorability in randomized polynomial time. Given instance $(S, C)$, let $D^+$ and $D^-$ be defined as above. Choose $\delta < \frac{1}{2}$, $\epsilon < \min \{\frac{1}{|S|}, \frac{1}{|C|}\}$.

If $(S, C)$ is $k$-$NM$-Colorable, then by the above claim there exists a $k$-term-DNF formula consistent with the positive and negative examples, so with probability at least $1 - \delta$, our learning algorithm will be able to find it.

Conversely, if $(S, C)$ is not $k$-$NM$-Colorable, by the Claim, there does not exist a consistent $k$-term-DNF formula, and the learning algorithm must either fail to produce a hypothesis in the allotted time, or produce one that is not consistent with at least one example. In either case, this can be observed, and we can determine that no legal $k$-$NM$-coloring is possible. ■

Thus we have shown that for $k \geq 2$, $k$-term-DNF is not PAC learnable by $k$-term-DNF. Furthermore, note that the target function $f$ created in this proof is monotone and thus this result holds even if the concept class is monotone $k$-term-DNF and the hypothesis class is $k$-term-DNF. Finally, a dual hardness result applies for learning $k$-clause-CNF by $k$-clause-CNF.

### 3.3 Learning Algorithm for $k$-term-DNF

Although $k$-term-DNF is not PAC learnable by $k$-term-DNF, we now show that it is learnable using a hypothesis from $k$-CNF.

Let $f = T_1 \lor T_2 \lor \ldots \lor T_k$ be the target formula, where

$$T_1 = y_1^{(1)} \land y_2^{(1)} \land \ldots \land y_m^{(1)}$$
\[ T_2 = y_1^{(2)} \land y_2^{(2)} \land \ldots \land y_{m_2}^{(2)} \]
\[ \vdots \]
\[ T_k = y_1^{(k)} \land y_2^{(k)} \land \ldots \land y_{m_k}^{(k)} \]

and \( y_i^{(j)} \) represents one of the \( n \) variables.

By distributing, we can rewrite \( f \) as:

\[ f = \bigwedge_{i_1, i_2, \ldots, i_k} (y_{i_1}^{(1)} \lor y_{i_2}^{(2)} \lor \ldots \lor y_{i_k}^{(k)}). \]

Now to learn \( f \) using a hypothesis form \( k \)-CNF, we introduce \( O(2n^k) \) variables \( \alpha_1, \alpha_2, \ldots, \alpha_m \), representing all disjunctions of the form: \( y_{i_1}^{(1)} \lor y_{i_2}^{(2)} \lor \ldots \lor y_{i_k}^{(k)} \). Learning the conjunction over the \( \alpha \)'s is equivalent to learning the original disjunction. Note, however, we may not (in general) transform the conjunction we obtain to a \( k \)-term-DNF formula, thus we must output it as a \( k \)-CNF formula.

### 3.4 Relations Between Concept Classes

In this section, we briefly study the containment relations between various concept classes.

We have already seen that \( k \)-term-DNF is a subclass of \( k \)-CNF. We now show that \( k \)-term-DNF is properly contained in \( k \)-CNF by exhibiting a \( k \)-CNF formula which can not be expressed as a \( k \)-term-DNF formula. The \( k \)-CNF formula, \((x_1 \lor x_2) \land (x_3 \lor x_4) \land \ldots \land (x_{2k-1} \lor x_{2k})\), has \( 2^k \) terms when we unfold it into the form of DNF formula and thus cannot be represented by a \( k \)-term-DNF formula.

As a dual result, it is easily shown that \( k \)-clause-CNF is properly contained in \( k \)-DNF. Thus it follows that, \( k \)-CNF \( \cup k \)-DNF \( \cup k \)-term-DNF \( \cup k \)-clause-CNF is PAC learnable.

We now consider the concept class \( k \)-DL as defined by Rivest [35]. Consider a Boolean function \( f \) that is defined on \( \{0, 1\}^n \) by a nested if–then–else statement of the form:

\[ f(x_1, x_2, \ldots, x_n) = \text{if } l_1 \text{ then } c_1 \text{ elseif } l_2 \text{ then } c_2 \text{ elseif } \ldots \text{ elseif } l_k \text{ then } c_k \text{ else } c_{k+1} \]

where the \( l_j \)'s are literals (either one of the variables or their negations), and the \( c_j \)'s are either \( T \) (true) or \( F \) (false). Such a function is said to be computed by a simple decision list. The concept class \( k \)-decision lists (\( k \)-DL) is just the extension of a simple decision list where the condition in each if statement may be the conjunction of up to \( k \) literals, for some fixed constant \( k \). We leave it as a simple exercise to show that \( k \)-CNF \( \cup k \)-DNF is properly contained in \( k \)-DL. Thus the first problem of Homework 1 provides an ever stronger positive result by showing that \( k \)-DL is PAC learnable using a hypothesis from \( k \)-DL.

27
4.1 Introduction

We have seen in earlier notes how to PAC learn a $k$-CNF formula. Recall that the algorithm used for learning $k$-CNF created a new variable corresponding to each possible term of at most $k$ literals and then just applied the algorithm for learning monomials. Observe that this algorithm assumes that the learner knows $k$ a priori. Can we modify this algorithm to work when the learner does not have prior knowledge of $k$? Here we consider a variant of the PAC model introduced in Topic 1 in which there is an unknown size parameter. The material presented in this lecture is just one portion of the paper, "Equivalence of Models for Polynomial Learnability," by David Haussler, Michael Kearns, Nick Littlestone, and Manfred Warmuth [21].

4.2 The Learning Algorithm

In this section we outline a general procedure to convert a PAC-learning algorithm $A$ that assumes a known size parameter $s$ (e.g. the $k$ in $k$-CNF) to a PAC-learning algorithm $B$ that works without prior knowledge of $s$. As an example application, this procedure will enable us to convert our algorithms for learning $k$-CNF, $k$-CNF, $k$-term-DNF, $k$-clause-CNF, or $k$-DL into corresponding algorithms that do not have prior knowledge of $k$. Observe, that while the learner does not know $s$, as one would expect the running time and sample complexity of $B$ will still depend on $s$. The most optimistic goal would be to have the time and sample complexity of $B$ match that of $A$.

The basic idea of this conversion is as follows. Algorithm $B$ will run algorithm $A$ with an estimate $\hat{s}$ for $s$ such that this estimate is gradually increased. The key question is: How does algorithm $B$ know when its estimate for $s$ is sufficient? The technique of hypothesis testing used to solve this problem is a general technique which is also useful in other situations.

Aside: Hypothesis Testing

We now describe a technique to test if a hypothesis is good. More specifically, given a hypothesis $h$, an error parameter $\epsilon$, and access to an example oracle EX we would like to determine with high probability if $h$ is an $\epsilon$-good hypothesis. Clearly it is not possible to distinguish a hypothesis with error $\epsilon$ from one with error just greater than $\epsilon$, however, we
can distinguish an $\epsilon/2$-good hypothesis from an $\epsilon$-bad one. As we shall see this is sufficient to know when our estimate for the size parameter is large enough.

We now formally describe the hypothesis testing algorithm.

**Algorithm Test**($h,n,\epsilon,\delta$)

1. Make $m = \left\lceil \frac{3\ln (n \ln 2 + \ln 2/\delta)}{\epsilon} \right\rceil$ call to EX.

2. Accept $h$ if it misclassifies at most $\frac{3\epsilon}{4}$ of the examples. Otherwise, reject $h$.

We now prove that this hypothesis testing procedure achieves the goal stated above.

**Lemma 4.1** The procedure Test when called with parameters $h$, $n$, $\epsilon$, and $\delta$ has the property that:

1. If $\text{error}(h) \geq \epsilon$, then $\text{Prob}[h \text{ is accepted}] \leq \frac{1}{2n+1}$

2. If $\text{error}(h) \leq \epsilon/2$, then $\text{Prob}[h \text{ is rejected}] \leq \frac{\delta}{2n+1}$

**Proof Sketch:**

We first sketch the proof showing the first property holds. Let $p$ be the error of hypothesis $h$. Then,

$$\text{Prob}[h \text{ is accepted}] \leq LE\left(p, m, \frac{3}{4}m \epsilon \right).$$

Finally, since $p \geq \epsilon$ it follows that

$$LE\left(p, m, \frac{3}{4}m \epsilon \right) \leq LE\left(p, m, (1 - \frac{1}{4})mp \right) \leq e^{-\frac{3m}{4}}.$$

Plugging in the value of $m$ used in Test, we get the stated result.

For the second property we know that $p \leq \epsilon/2$ and thus the probability of rejecting $h$ is bounded above by:

$$GE\left(p, m, \frac{3}{4}m \epsilon \right) \leq GE\left(p, m, (1 + \frac{1}{2})m \epsilon \right) \leq e^{-\frac{3m}{4}}.$$

Again this gives the desired bound.

Finally, we note that a two-oracle version of this hypothesis testing procedure can be constructed by running the one oracle version Test twice (replacing $\delta$ by $\delta/2$), once using the positive oracle and once using the negative oracle. The two-oracle testing procedure accepts $h$ if an only if both of the above calls to Test accept $h$. The above lemma also holds for this two-oracle testing procedure.

We now return to the problem of handling an unknown size parameter by describing how algorithm $B$ (unknown $s$) can be implemented using Algorithm $A$ (known $s$) as a subroutine.
Let \( p(n,s,1/\epsilon) = \max\{S_A(n,s,\epsilon,1/2), T_A(n,s,\epsilon,1/2)\} \) where \( S_A \) is the sample complexity of algorithm \( A \) and \( T_A \) is the time complexity of algorithm \( A \). We know describe algorithm \( B \).

Algorithm \( B(n,\epsilon,\delta) \)

1. \( i \leftarrow 0 \)
2. UNTIL \( h \) is accepted by Test\((h,n,\epsilon,\delta)\) DO
3. \( i \leftarrow i + 1 \)
4. \( \hat{s} \leftarrow \left[2^{i-1}/\ln(\frac{\delta}{2})\right] \)
5. \( h_i \leftarrow \) hypothesis output by \( A(n,\hat{s},\epsilon/2,1/2) \)
6. Output \( h = h_i \)

**Theorem 4.1** Let \( h \) be the hypothesis output by algorithm \( B \) as described above. Then 
\[
\text{Prob}[\text{error}(h) \leq \epsilon] \geq 1 - \delta.
\]

**Proof Sketch:**

Observe that algorithm \( B \)'s estimate \( \hat{s} \geq s \) at the \( i \)th repetition for all \( i \geq \left[ 1 + \frac{\ln \frac{2}{\delta}}{\ln 2} \log_2 s \right] \).

Since the size parameter is only an upperbound on the allowable size and algorithm \( A \) is a PAC-learning algorithm we know that for any iteration in which \( \hat{s} \geq s \), the \( \text{Prob}[h_i \leq \epsilon/2] \geq 1/2 \). In such a case, the \( \text{Prob}[h_i \text{ is accepted by Test}] \geq 3/4 \). So if \( \hat{s} \geq s \) then the \( \text{Prob}[B \text{ halts with hyp. of error } \leq \epsilon/2] \geq 3/8 \).

Let \( j = \lfloor (\ln 2/\delta)/(\ln 8/5) \rfloor \). Then

\[
\text{Prob}[B \text{ fails to halt after } j \text{ iterations with } \hat{s} \geq s] \leq \left( \frac{5}{8} \right)^j \leq \delta/2.
\]

Thus with probability at least \( 1 - \delta/2 \), \( B \) will halt after at most

\[
j' = \left\lfloor \frac{\ln 2}{\delta} \log_2 s \right\rfloor + \left\lceil \frac{\ln 2/\delta}{\ln 8/5} \right\rceil
\]

iterations.

Also the probability is at most \( \delta/2 \) that any call to Test will accept a hypothesis with error greater than \( \epsilon \). Thus with probability \( \geq 1 - \delta \), algorithm \( B \) will halt after at most \( j' \) repetitions with an \( \epsilon \)-good hypothesis.

Finally, one can verify that the time an sample complexity after \( j' \) iterations is still polynomial. We refer the reader to the paper by Haussler et al. [21] for the details. ■
5.1 Introduction

Although up to now we have assumed that the data provided by the example oracle is noise-free, in real-life learning problems this assumption is almost never valid. Thus we would like to be able to modify our algorithms so that they are robust against noise. Before considering learning with noise in the PAC model, we must first formally model the noise. Although we will only focus on one type of noise here, we first describe the various formal models of noise that have been considered.

In all cases we assume that the usual noise-free examples pass through a noise oracle before being seen by the learner. Each noise oracle represents some noise process being applied to the examples from EX. The output from the noise process is all the learner can observe. The “desired,” noiseless output of each oracle would thus be a correctly labeled example \((x, s)\), where \(x\) is drawn according to the unknown distribution \(D\). We now describe the actual outputs from the following noise oracles:

**Random Misclassification Noise** [7]: This noise oracle models a benign form of misclassification noise. When it is called, it calls EX to obtain some (noiseless) \((x, s)\), and with probability \(1 - \eta\), it returns \((x, s)\). However, with probability \(\eta\), it returns \((x, \tilde{s})\).

**Malicious Noise** [44]: This oracle models the situation where the learner usually gets a correct example, but some small fraction \(\eta\) of the time the learner gets noisy examples and the nature of the noise is unknown or unpredictable. When this oracle is called, with probability \(1 - \eta\), it does indeed return a correctly labeled \((x, s)\) where \(x\) is drawn according to \(D\). With probability \(\eta\) it returns an example \((x, \tilde{s})\) about which no assumptions whatsoever may be made. In particular, this example may be maliciously selected by an adversary who has infinite computing power, and has knowledge of the target concept, \(D\), \(\eta\), and the internal state of the learning algorithm.

**Malicious Misclassification Noise** [41]: This noise oracle models a situation in which the only source of noise is misclassification, but the nature of the misclassification is unknown or unpredictable. When it is called, it also calls EX to obtain some (noiseless) \((x, s)\), and with probability \(1 - \eta\), it returns \((x, s)\). With probability \(\eta\), it returns \((x, l)\) where \(l\) is a label about which no assumption whatsoever may be made. As with malicious noise we assume an omnipotent, omniscient adversary; but in the case the adversary only gets to choose the label of the example.
Uniform Random Attribute Noise [41]: This noise oracle models a situation where the attributes of the examples are subject to noise, but that noise is as benign as possible. For example, the attributes might be sent over a noisy channel. We consider this oracle only when the instance space is \(\{0,1\}^n\) (i.e., we are learning Boolean functions). This oracle calls EX and obtains some \((x_1 \cdots x_n, s)\). It then adds noise to this example by independently flipping each bit \(x_i\) to \(\bar{x}_i\) with probability \(\eta\) for \(1 \leq i \leq n\). Note that the label of the "true" example is never altered.

Nonuniform Random Attribute Noise [17]: This noise oracle provides a more realistic model of random attribute noise than uniform random attribute noise.\(^2\) This oracle also only applies when we are learning Boolean functions. This oracle calls EX and obtains some \((x_1 \cdots x_n, s)\). The oracle then adds noise by independently flipping each bit \(x_i\) to \(\bar{x}_i\) with some fixed probability \(\eta_i \leq \eta\) for each \(1 \leq i \leq n\).

In this paper we focus on the situation in which there is random misclassification noise. The material presented here comes from the paper "Learning from Noisy Examples," by Dana Angluin and Phil Laird [7]. We show that the hypothesis that minimizes disagreements (i.e., the hypothesis that misclassifies the fewest training examples) meets the PAC correctness criterion when the examples are corrupted by random misclassification noise. Unfortunately, this technique is most often computationally intractable. However, for \(k\)-CNF formulas we describe an efficient PAC learning algorithm that works against random misclassification noise. Both positive results need only assume that the noise rate \(\eta\) is less than one half.

Before describing these results, we briefly review what is known about handling the other forms of noise. Sloan [41] has extended the above results to the case of malicious labeling noise. On the other hand, Kearns and Li [25] have shown that the method of minimizing disagreements can only tolerate a small amount of malicious noise. We will study this result in Topic 15.

Unlike the results for labeling noise, in the case of uniform random attribute noise, if one uses the minimal disagreement method, then the minimum error rate obtainable (i.e., the minimum "epsilon") is bounded below by the noise rate [41]. Although the method of minimizing disagreements is not effective against random attribute noise, there are techniques for coping with uniform random attribute noise. In particular, Shackelford and Volper [38] have an algorithm that tolerates large amounts of random attribute noise for learning \(k\)-DNF formulas. That algorithm, however, has one very unpleasant requirement: it must be given the exact noise rate as an input. Goldman and Sloan [17] describe an algorithm for learning monomials that tolerates large amounts of uniform random attribute noise (any noise rate less than 1/2), and only requires some upper bound on the noise rate as an input. Finally, for nonuniform random attribute noise, Goldman and Sloan [17] have shown that the minimum error rate obtainable is bounded below by one-half of the noise rate, regardless of the technique (or computation time) of the learning algorithm.

\(^2\)Technically, this oracle specifies a family of oracles, each member of which is specified by \(n\) variables, \(\eta_1, \ldots, \eta_n\), where \(0 \leq \eta_i \leq \eta\).
5.2 Learning Despite Classification Noise

Let $EX_\eta$ be the random misclassification noise oracle with a noise rate of $\eta$. Thus the standard noise-free oracle is $EX_0$. Also we will use the notation that $C = \{L_1, L_2, \ldots, L_N\}$ where $L_*$ is the target concept. So we only consider the simple case of a finite set of hypothesis.

In this section we will study the random misclassification model and give an algorithm to PAC learn any finite concept space with polynomial sample size (not necessarily with polynomial time). In the next section we show that for $\eta < 1/2$, the class of $k$-CNF formulas is PAC learnable from $EX_\eta$. Observe that the learning problem is not feasible for $\eta \geq 1/2$. If $\eta = 1/2$ the noise distorts all the information and clearly no learning is possible, and when $\eta > 1/2$, we actually learn the complement concept with $\eta < 1/2$.

For now we assume that the learner has an upper bound $\eta_b$ on the noise rate. That is, $\eta \leq \eta_b < 1/2$. Later we show how to remove this assumption. As one would expect if $\eta_b$ is very close to $1/2$, we must allow the learner more time and data. In fact, we will require that the time and sample complexity of the learner are polynomial in $1/(1 - 2\eta_b)$. Observe that this quantity is inversely proportional to how close $\eta_b$ is to $1/2$.

5.2.1 The Method of Minimizing Disagreements

Our goal in this section is to study the sample complexity for PAC learning under random misclassification noise. For the noise-free case we have seen that if $L_i$ agrees with at least $m \geq (1/\epsilon)(\ln |C| + \ln(1/\delta)) = (1/\epsilon)\ln(N/\delta)$ samples drawn from $EX_0$ then $\Pr[error(L_i) \geq \epsilon] \leq \delta$. How much more data is needed when there is random misclassification noise?

In the presence of noise the above approach will fail because there is no guarantee that any hypothesis will be consistent with all the examples. However, if we replace the goal of consistency with that of minimizing the number of disagreements with the examples and permit the sample size to depend on $\eta_b$, we get an analogous result for the noisy case.

We shall use the following notation in formally describing the method of minimizing disagreements.

- Let $\sigma$ be the sequence of examples drawn from $EX_\eta$.
- Let $F(L_i, \sigma)$ be the number of times $L_i$ disagrees with $\sigma$ on an example in $\sigma$, where $L_i$ disagrees with $\sigma$ on an example $(\bar{x}, l) \in \sigma$ if and only if $L_i$ classifies $\bar{x}$ differently from $l$.

**Theorem 5.1** If we draw a sequence $\sigma$ of

$$m \geq \frac{2}{\epsilon^2 (1 - 2\eta_b)^2 \ln \left(\frac{2N}{\delta}\right)}$$

samples from $EX_\eta$ and find any hypothesis $L_i$ that minimizes $F(L_i, \sigma)$ then

$$\Pr[error(L_i) \geq \epsilon] \leq \delta$$

35
Proof: We shall use the following notation in the proof. Let $d_i$ be the error($L_i$). That is, $d_i$ is the probability that $L_i$ misclassifies a randomly drawn example. Let $p_i$ be the probability that an example from $EX_{\eta}$ disagrees with $L_i$. Observe that $p_i$ is the probability that $L_i$ misclassifies a correctly labeled example ($d_i(1 - \eta)$), plus the probability that $L_i$ correctly classifies the example but the example has been improperly labeled by the noise oracle ($\eta$). Thus

$$p_i = d_i(1 - \eta) + \eta = \eta + d_i(1 - 2\eta)$$

Note that for the right hypothesis ($L_i = L_*$), $d_i = 0$ and therefore $p_i = \eta$ (i.e., disagreements are only caused by noise). Since $\eta < 1/2$, it follows that for any hypothesis $p_i \geq \eta$. So all hypothesis have an expected rate of disagreement of at least $\eta$.

Let an $\epsilon$-bad hypothesis be one for which $d_i \geq \epsilon$. Then for any $\epsilon$-bad hypothesis $L_i$, we have

$$p_i \geq \eta + \epsilon(1 - 2\eta).$$

Thus we have a separation of at least $\epsilon(1 - 2\eta)$ between the disagreement rates of the correct and an $\epsilon$-bad hypothesis. Although $\eta_b$ is not known, we know that $\eta \leq \eta_b < 1/2$ thus the minimum separation (or gap) is at least $\epsilon(1 - 2\eta_b)$. We take advantage of this gap in the following manner. We will draw enough examples from $EX_{\eta}$ to guarantee with high probability that no $\epsilon$-bad hypothesis has a observed disagreement rate greater than $\eta + \epsilon(1 - 2\eta_b)/2$. Similarly, we will draw enough examples from $EX_{\eta}$ to guarantee with high probability that the correct hypothesis has an observed disagreement rate less than $\eta + \epsilon(1 - 2\eta_b)/2$. Thus it follows that with high probability $L_*$ will have a lower observed rate of disagreement than any $\epsilon$-bad hypothesis. Thus by selecting the hypothesis with the lowest observed rate of disagreement, the learner knows (with high probability) that this hypothesis has error at most $\epsilon$.

We now formalize this intuition. We will draw $m$ examples from $EX_{\eta}$ and compute an empirical estimate for all $p_i$. That is, we compute $F(L_i, \sigma)$ for every $L_i$ in the hypotheses space. The hypothesis output will be the hypothesis $L_i$ that has the minimum estimate for $p_i$. What is the probability that $L_i$ is $\epsilon$-bad? Let $s = \epsilon(1 - 2\eta_b)$. In order for some $\epsilon$-bad hypothesis $L_i$ to minimize $F(L_i, \sigma)$ either the correct hypothesis must have a high disagreement rate

$$F(L_*, \sigma) / m \geq \eta + s/2$$

or an $\epsilon$-bad hypothesis must have a low disagreement rate ($\leq \eta + s/2$). Finally, assuming that neither of these bad events occur, since we select the hypothesis $L_i$ that minimizes the disagreement rate we know that:

$$F(L_i, \sigma) / m < \eta + s/2$$

and thus $L_i$ has error of at most $\epsilon$.
Applying Chernoff bounds for the probability that a good hypothesis has high disagreement:
\[
\Pr[F(L_*, \sigma)/m \geq \eta + s/2] = GE(\eta, m, m(\eta + s/2) < \delta/(2N) < \delta/2.
\]
And if \( L_i \) is \( \epsilon \)-bad then its probability to have low disagreement is:
\[
\Pr[F(L_*, \sigma)/m \leq \eta + s/2] = LE(\eta + s, m, m(\eta + s/2) \leq \delta/(2N).
\]
Thus the probability that any \( \epsilon \)-bad hypothesis \( L_i \) has \( F(L_i, \sigma)/m \leq \eta + s/2 \) is at most \( \delta/2 \).
(There are at most \( N - 1 \) hypotheses that are \( \epsilon \)-bad.) Putting these two equalities together, the probability that some \( \epsilon \)-bad hypothesis minimizes \( F(L_i, \sigma) \) is at most \( \delta \).

Thus we know that by using the method of minimizing disagreements one can tolerate any noise rate strictly less than \( 1/2 \). Furthermore, if the hypothesis minimizing disagreements can be found in polynomial time then we obtain an efficient PAC algorithm for learning when there is random misclassification noise.

5.2.2 Handling Unknown \( \eta_b \)

Until now we have assumed the learner is given an upperbound \( \eta_b \) on the noise rate. What if such an upperbound is not known? We can solve this problem using the technique described in Topic 4 for handling an unknown size parameter. That is, just treat \( \eta \) as the unknown size parameter. The only detail we need to worry about here is how to perform the hypothesis testing. The basic idea is as follows, we draw some examples and estimate the failure probability of each of the hypotheses \( L_1, ..., L_N \). The smallest estimate is compared to the current value of \( \eta_b \). If the estimate \( \hat{p}_i \) is less than the current value of \( \eta_b \), we halt; otherwise we increase \( \eta_b \) and repeat. For details see the Angluin, Laird paper [7].

5.2.3 How Hard is Minimizing Disagreements?

How hard is to find an hypothesis \( L_i \) that minimizes \( F(L_i, \sigma) \)? Unfortunately, the answer is that is usually quite hard. For example consider the domain of all conjunctions of positive literals (monotone monomials). To find a monotone monomial that minimizes the disagreement is a NP-hard problem.

**Theorem 5.2** Given a positive integer \( n \) and c and a sample \( \sigma \). The problem of determining if is there a monotone monomial \( \psi \) over \( n \) variables such that \( F(\pi, \sigma) \leq c \) is NP-complete.

The result indicates that even for a very simple domain, the approach of directly trying to minimize the disagreement is unlikely to be computationally feasible. However, in the next section we show that for some concept classes, we can bypass the minimization problem (which is hard) and efficiently PAC learn the concepts from noisy examples.
5.3 Learning $k$-CNF Under Random Misclassification Noise

In the previous section we described how to PAC learn any finite concept space if we remove the condition that our algorithm runs in polynomial time. However, we would like to have efficient algorithms for dealing with noise. In this section we use some of the ideas suggested by the method of minimizing disagreements to get a polynomial time algorithm for PAC learning $k$-CNF formulas using examples from $EX_n$.

5.3.1 Basic Approach

Instead of searching for the $k$-CNF formula with the fewest disagreements, we will test all potential clauses individually, and include those that are rarely false in a positive example. Of course, if a clause is false yet the example is reported as positive then either the clause is not in the target formula, or the label has been inverted by the noise process. Thus if a clause is false on a significant number of positive examples, then we do not want to include the clause in our hypothesis. Observe that we will not be solving an NP-complete problem, the $k$-CNF formula that is chosen may not minimize the disagreements with the examples, but it will (with probability $\geq 1 - \delta$) have an error that is less then $\epsilon$.

We now give some notation that we use in this section.

- Let $M$ be the number of possible clauses of at most $k$ literals ($M \leq (2n+1)^k$), and let $C$ be any such clause.
- Let $\phi_\ast$ be the target $k$-CNF formula
- For all clauses $C$, let $P_{00}(C) = \text{Prob}[C \text{ is false and } \phi_\ast \text{ is false}].$
- For all clauses $C$, let $P_{01}(C) = \text{Prob}[C \text{ is false but } \phi_\ast \text{ is true}].$
- For all clauses $C$, let $P_0(C) = P_{00}(C) + P_{01}(C) = \text{Prob}[C \text{ is false}].$

Finally, we need the following two definitions. We say that a clause $C$ is important if and only if $P_0(C) \geq Q_I = \epsilon/(16M^2)$. We say a clause $C$ is harmful if and only if $P_{01}(C) \geq Q_H = \epsilon/(2M)$. Note that $Q_H \geq Q_I$, so every harmful clause is important. Also, no clause contained in $\phi_\ast$ can be harmful, since if $\phi_\ast$ classifies an example as positive, the example satisfies all its clauses and $C$ must be true (i.e., $P_{01}(C) = 0$).

The algorithm to PAC learn $k$-CNF formulas works as follows. We must construct an hypothesis $h$ that is $\epsilon$-good with high probability. To achieve this goal the hypothesis $h$ must have all the important clauses that are not harmful. A non-important clause is almost always assigned the value "true" (by the examples in $EX_n$), and thus it does not matter if it is included in $h$ or not. On the other hand, a harmful clause must not be included in $h$, since it is very likely to be falsified by a positive example.
Thus our goal is to find an hypothesis $h$ that includes all important clauses and does not include any harmful clause. We first prove that if we find such a hypothesis $h$ then it is $\epsilon$-good. Then we show how to efficiently construct such a hypothesis with high probability.

**Lemma 5.1** Let $D$ be a fixed unknown distribution, and let $\phi_*$ be a fixed unknown target. Let $\phi$ be any conjunction (product) of clauses that contains every important clause in $\phi_*$ and no harmful clauses. Then $\text{error}(\phi) \leq \epsilon$.

**Proof:** The probability that $\phi$ misclassifies an example from $D$ is equal to the probability that $\phi_*$ classifies the example as positive but $\phi$ classifies it as negative, or vice versa (the example is truly negative but $\phi$ classifies it as positive).

The probability of an error on a positive example is equal to the probability that any clause in $\phi - \phi_*$ is falsified by positive example; therefore,

$$\text{Prob}[\phi_* = 1 \land \phi = 0] \leq \sum_{C \in \phi - \phi_*} P_0(C) < MQ_H = \epsilon/2$$

since there are at most $M$ clauses in $\phi$ and none are harmful.

The probability of error on a negative example is equal to the probability that a clause in $\phi_* - \phi$ is falsified and all clauses in $\phi$ are true. This probability is less or equal to the probability that a clause in $\phi_* - \phi$ is falsified. Since $\phi$ contains all the important but not harmful clauses and $\phi_*$ contains no harmful clauses, then $\phi_* - \phi$ contains only non-important clauses. Therefore,

$$\text{Prob}[\phi = 1 \land \phi_* = 0] \leq \sum_{C \in \phi_* - \phi} P_0(C) < MQ_I = \epsilon/(16M) < \epsilon/2$$

Thus,

$$\text{error}(\phi) \leq \text{Prob}[\phi = 1 \land \phi_* = 0] + \text{Prob}[\phi = 0 \land \phi_* = 1] \leq \epsilon/2 + \epsilon/2 = \epsilon.$$

To complete the algorithm for learning $k$-CNF under noise, we must construct an efficient algorithm to find a formula $\phi$ that contains all the important clauses and no harmful clauses (with high probability). Observe that we have no direct information about whether a clause is important or harmful. More specifically, we cannot directly compute $P_{00}(C)$ or $P_{01}(C)$, but rather must rely on the examples received from $EX_\eta$. However, since $EX_\eta$ only modifies the label and not the values of the attributes in the example, $P_0(C)$ can be directly estimated by drawing a sample from $EX_\eta$, and calculating the fraction of the assignments that assign “false” to $C$. Thus we can accurately estimate $P_0(C)$ for every possible clause and thus decide (with high probability) which clauses are important. Let $I$ be the set of clauses that are determined to be important. The only thing left, is to identify the harmful clauses in $I$ and eliminate them.

Harmful clauses are those that are falsified by positive examples (examples that satisfy $\phi_*$), but since the classification of the example is subject to noise we cannot directly estimate
Let \( P_{0+}(C) \) be the probability that a clause is falsified by an example and that \( EX_{\eta} \) reports that the example is positive. This probability can be directly estimated by counting the number of examples from the sample that are reported as positive and falsify the clause \( C \). We perform this estimate for every important clause. For the estimate of \( P_{0+}(C) \), an example is counted if and only if the example is positive and no error occurred in the reporting, or the example is negative and there was an error in the reporting. Thus,

\[
P_{0+}(C) = (1 - \eta)P_{01}(C) + \eta P_{00}(C)
= \eta(P_{00}(C) + P_{01}(C)) + (1 - 2\eta)P_{01}(C)
= \eta P_{0}(C) + (1 - 2\eta)P_{01}(C)
\]

If \( P_0(C) \neq 0 \), then the proportion of examples falsifying \( C \) that are reported as positive is:

\[
\frac{P_{0+}(C)}{P_0(C)} = \eta + \frac{P_{01}(C)}{P_0(C)}(1 - 2\eta)
\]

and since \( \eta < 1/2 \), we get:

\[
\frac{P_{0+}(C)}{P_0(C)} \geq \eta.
\]

We would like to separate between two cases: one is the case where \( C \) is a desired clause \((C \in \phi_*)\) and the other is when \( C \) is harmful. If \( C \in \phi_* \) then

\[
\frac{P_{0+}(C)}{P_0(C)} = \eta,
\]

while if \( C \) is harmful \((P_{01}(C) \geq Q_H)\):

\[
\frac{P_{0+}(C)}{P_0(C)} \geq \eta + Q_H(1 - 2\eta) \geq \eta + \frac{\epsilon}{2M}(1 - 2\eta).
\]

Thus, we have a separation of at least \( s = Q_H(1 - 2\eta) \) between the clauses that are included in \( \phi_* \) and the harmful clauses. Since \( \eta \leq \eta_3 \), the separation is bounded below by \( s = Q_H(1 - 2\eta_3) \).

If we knew the value of \( \eta \), we could estimate \( P_{0+}/P_0 \) for all clauses and delete from \( I \) all the clauses with an estimate that is greater than \( \eta + s/2 \). However, we do not know \( \eta \) and thus we do not know where the cutoff is.

How can we estimate \( \eta \)? If \( I \) contains any clause \( C \) that is also in \( \phi_* \) then the estimate of \( P_{0+}(C)/P_0(C) \) will be close to \( \eta \). So we can take the minimum of this value over all the clauses in \( I \) as a first estimate for \( \eta \):

\[
\hat{\eta} = \min_{C \in I} \frac{P_{0+}(C)}{P_0(C)}.
\]

If no clause in \( I \) is contained also in \( \phi_* \), then the estimate \( \hat{\eta} \) calculated above may not be a good estimate of \( \eta \). However, since \( I \) contains all the important clauses it must be that
\( \phi_* \) contains no important clauses. This means that most of the examples drawn from \( D \) satisfy these non-important clauses and therefore satisfy \( \phi_* \). In this case most examples are positive, and thus the observed overall rate of negative examples is sufficiently close to \( \eta \). Thus, the estimate of \( \eta \) is taken to be the minimum of the two estimates:

\[
\hat{\eta} = \min \left\{ \frac{\text{fraction of negative examples}}{\min_{c \in I} \left\{ \frac{P_{0+}(C)}{P_0(C)} \right\}} \right\}.
\]

We use therefore \( \hat{\eta} + s/2 \) to decide which are the harmful clauses.

### 5.3.2 Details of the Algorithm

We now put these ideas together to get an efficient algorithm for PAC learning \( k \)-CNF formulas under random misclassification noise.

**Algorithm Learn-Noisy-\( k \)-CNF(\( n, k, \varepsilon, \delta, \eta_b \))**

1. \( m = \left[ \frac{2^{10M^4}}{\varepsilon^2 (1 - \eta_b)} \ln \left( \frac{6M}{\delta} \right) \right] \), where \( M \) is the number of possible clauses

2. \( Q_I = \varepsilon / (16M^2) \); \( Q_H = \varepsilon / 2M \); \( s_b = Q_H (1 - 2\eta_b) \)

3. We draw \( m \) examples from the oracle \( EX_\eta \), and set: \( \hat{P}_\neg = \text{number of negative examples in the sample} \)

4. For each possible clause \( C \), we compute:
   (a) \( \hat{P}_0(C) \) = the number of examples that falsify the clause \( C \)
   (b) \( \hat{P}_{0+}(C) \) = the number of positive (reported) examples that falsify \( C \)
   (c) If \( P_0(C) \neq 0 \) then \( h(C) = \hat{P}_{0+}(C) / \hat{P}_0(C) \).
   (d) \( \eta_1 = \hat{P}_\neg / m \), the observed fraction of negative examples

5. Form the set \( I \) by including all the important clauses \( C \); i.e., \( \hat{P}_0(C) / m \geq Q_I / 2 \)

6. \( \eta_2 = \min_{c \in I} \{ h(C) \} \)

7. \( \hat{\eta} = \min \{ \eta_1, \eta_2 \} \)

8. The final output \( \phi \) is the conjunction (product) of all those clauses \( C \in I \) such that \( h(C) \leq \hat{\eta} + s_b / 2 \)

To prove that the algorithm is correct, we need to prove that \( \phi \) contains all important clauses that are not harmful (with probability greater than \( 1 - \delta \)). Only in the following cases the algorithm could go wrong:

- Some important clauses might not be selected for inclusion in \( I \).

41
• the estimate $\hat{\eta}$ could be too large ($\hat{\eta} \geq \eta + s/4$) or too small ($\hat{\eta} \leq \eta - s/4$).

• Some harmful clauses may be included in $\phi$.

• Some correct clauses may be excluded from $\phi$.

The proof uses Chernoff bounds, showing that the second case has a probability of at most $\delta/2$, while the other cases, each has a probability of at most $\delta/6$. Therefore, the total probability of error is at most $\delta$, and by the previous lemma, the output formula is $\epsilon$-good with high probability. For the details of the proof, see the Angluin, Laird paper [7].
6.1 Introduction

The above quote is better known to us today as Occam’s Razor or the Principle of Ontological Parsimony. Over the centuries, people from different fields have given it different interpretations. One interpretation used by the experimental scientists is: given two explanations of the data, all other things being equal, the simpler explanation is preferable.

This principle is consistent with the goal of machine learning: to discover the simplest hypothesis that is consistent with the sample data. However, the question still remains: why should one assume that the simplest hypothesis based on past examples will perform well on future examples. After all, the real value of a hypothesis is in predicting the examples that it has not yet seen.

It will be shown that, under very general assumptions, Occam’s Razor produces hypotheses that with high probability will be predictive of future observations. As a consequence, when hypotheses of minimum or near minimum complexity can be produced in time polynomial in the size of the sample data, it leads to a polynomial PAC-learning algorithm. The material presented in the lecture comes from the paper “Occam’s Razor,” by Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth [11]. Portion of the notes are taken from this paper.

6.2 Using the Razor

In this section, we define an Occam-algorithm and show that the existence of an Occam-algorithm implies polynomial learnability. First, we prove the uniform convergence lemma.

Lemma 6.1 (Uniform Convergence Lemma) Given any function $f$ in a hypothesis class of $r$ hypotheses, the probability that any hypothesis with error larger than $\varepsilon$ is consistent with a sample of size $m$ is less than $(1 - \varepsilon)^m r$.

Proof: Let $h_i$ be any hypothesis with error larger than $\varepsilon$. The probability that $h_i$ is consistent with one observation of $f$ is less than $(1 - \varepsilon)$. Since all the $m$ samples of $f$ are drawn independent of each other, the probability that $h_i$ is consistent with all $m$ observations of $f$ is less than $(1 - \varepsilon)^m$. Finally, since there are $r$ hypothesis, the probability that any
hypothesis with error larger than $\epsilon$ is consistent with all $m$ observations of $f$ is less than $(1 - \epsilon)^m r$.

For an infinite hypothesis class, the learning algorithm would have to choose a hypothesis carefully. Occam’s Razor suggests that the learning algorithm should choose its hypothesis among those that are consistent with the sample and have the minimum complexity. But this may sometimes be intractable. For example, finding a minimum length DNF expression consistent with a sample of a Boolean function and finding a minimum size DFA consistent with a set of positive and negative examples of a regular language are known to be NP-hard [14]. To obtain polynomial algorithms, the criterion is weakened as follows.

**Definition 6.1** An Occam-algorithm for $H$ with constant parameters $c \geq 1$ and compression factor $0 \leq \alpha < 1$ is a learning algorithm that given a sample of size $m$ which is labeled according to some hypothesis $h \in H$:

1. produces a hypothesis consistent with the data, that is, all the observations can be explained by the hypothesis,

2. produces a hypothesis of complexity at most $n^c m^\alpha$ where $n$ is the complexity of $h$, and

3. runs in time polynomial in $n$ and $m$.

We now show that the existence of an Occam-algorithm for $H$ implies polynomial learnability.

**Theorem 6.1** Given independent observations of any function in $H$ of complexity at most $n$, an Occam-algorithm with parameters $c \geq 1$ and $0 \leq \alpha < 1$ produces a hypothesis of error at most $\epsilon$ with probability at least $1 - \delta$ using sample size polynomial in $n, 1/\epsilon$, and $1/\delta$, independent of the function and of the probability distribution. The sample size required is

$$O \left( \frac{1}{\epsilon} \log \left( \frac{1}{\delta} \right) + \left( \frac{n^c}{\epsilon} \right)^{1/(1-\alpha)} \right).$$

**Proof:** Let the size of the hypothesis space be $r$. We will show that the Occam-algorithm produces a good hypothesis after drawing a sample of size

$$m \geq \max \left\{ \frac{2 \log (1/\delta)}{-\log (1 - \epsilon)^c} \left( \frac{2n^c}{-\log (1 - \epsilon)} \right)^{1/(1-\alpha)} \right\}.$$

Since the hypothesis under consideration are given by binary strings of length at most $n^c m^\alpha$, the size of the hypothesis space, $r$, is at most $2^{n^c m^\alpha}$.

We first consider the second lower bound on $m$,

$$m \geq \left( \frac{2n^c}{-\log (1 - \epsilon)} \right)^{1/(1-\alpha)}.$$
Raising both sides to the power of \(1 - \alpha\) yields
\[
m^{1-\alpha} \geq \frac{2n^\epsilon}{\log(1 - \epsilon)}.
\]

Simplifying further we obtain
\[
n^\epsilon m^\alpha \leq \frac{1}{2} m \log(1 - \epsilon).
\]

Finally, raising both sides to the power of 2 gives
\[
2^{n^\epsilon m^\alpha} \leq (1 - \epsilon)^{-m/2}.
\]

Recall that \(r \leq 2^{n^\epsilon m^\alpha}\) and thus it follows from above that:
\[
r \leq (1 - \epsilon)^{-m/2}.
\]

From the uniform convergence lemma, we get that:
\[
\Pr[\text{any } \epsilon\text{-bad hyp. is consistent with } m \text{ exs.}] \leq (1 - \epsilon)^m r \\
\leq (1 - \epsilon)^{-m/2} (1 - \epsilon)^m \\
= (1 - \epsilon)^{m/2}.
\] (6.1)

Finally, we consider the first lower bound on \(m\),
\[
m \geq \frac{2 \log(1/\delta)}{-\log(1 - \epsilon)}.
\]

Multiplying both sides by \(\log(1 - \epsilon)\) which is negative, we get
\[
\frac{m}{2} \log(1 - \epsilon) \leq -\log(1/\delta) = \log \delta.
\]

Raising both sides to the power of 2 gives
\[
(1 - \epsilon)^{m/2} \leq \delta.
\] (6.2)

We complete the proof by combining Equations (6.1) and (6.2) gives that the probability that an \(\epsilon\)-bad hypothesis is consistent with all the \(m\) examples is at most \(\delta\). Thus the Occam-algorithm produces a good hypothesis after drawing \(m\) examples where
\[
m = O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right) + \left(\frac{n^\epsilon}{\epsilon}\right)^{1/(1-\alpha)}\right)
\]

By definition of an Occam-algorithm, it runs in time polynomial in \(n\) and \(m\). Thus, we have a PAC-learning algorithm.
7.1 Introduction

In the notes on Occam’s Razor, we defined the condition for uniform convergence on finite concept classes. We can state the general condition for uniform convergence as follows:

$$\Pr_{S \in D^m} [\exists h \in C \ |error_D(h) > \epsilon \wedge h \text{ is consistent with } S] \leq \delta$$

where $S$ is a set of $m$ examples from the instance space $X$ having probability distribution $D$. To determine whether a concept class $C$ is uniformly learnable, we have to find out if there exists a learning function $A$ satisfying the above condition. If so, we say that $C$ is uniformly learnable and has sample complexity $m$.

For finite concept classes, such uniform convergence proofs are easy because we know $|C|$. However, when the concept class is infinite, we need some other measure to replace $|C|$. Such a measure is a combinatorial parameter known as the Vapnik-Chervonenkis (VC) dimension. The VC dimension of a concept class $C$ is a measure of the complexity of the class.

In the next section, we will define the VC dimension and some other concepts needed to define the VC dimension. We shall then measure the VC dimension of some concept classes. The following section will contain the theorems relating the VC dimension to uniform learnability and the sample complexity. Finally, we will see some relations on the VC dimension.

The material presented in the lecture comes from the paper “Learnability and the Vapnik-Chervonenkis Dimension,” by Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred Warmuth [12]. Portions of the notes are taken from this paper.

7.2 VC Dimension Defined

The definition of VC dimension uses the definition of a shattered set. We now give two equivalent definitions for a shattered set. Let $X$ be the instance space and $C$ the concept class.

Definition 7.1 A finite set $S \subseteq X$ is shattered by $C$ if for each subset $S' \subseteq S$, there is a concept $c \in C$ which contains all of $S'$ and none of $S - S'$.

In order to give the alternate definition of shattering, we first need the following definition.
Definition 7.2 Given $S \subseteq X$, $\Pi_C(S)$ denotes the set of all subsets of $S$ that can be obtained by intersecting $S$ with a concept in $C$. Thus,

$$\Pi_C(S) = \{S \cap c : c \in C \}.$$ 

For any integer $m > 0$, $\Pi_C(m) = \max(\{|\Pi_C(S)|\})$ over all $S \subseteq X$ where $|S| = m$.

Definition 7.3 Given $S \subseteq X$, if $\Pi_C(S) = 2^S$, the power set of $S$, then $S$ is shattered by $C$.

We can now define the VC dimension.

Definition 7.4 VC dimension of $C$, denoted as $\text{vcd}(C)$, is the smallest $d$ for which no set of $d + 1$ instances is shattered by $C$.

Definition 7.5 Equivalently, $\text{vcd}(C)$ is the cardinality of the largest finite set of points $S \subseteq X$ that is shattered by $C$ (i.e., the largest integer $d$ such that $\Pi_C(d) = 2^d$).

7.3 Example Computations of VC Dimension

Consider a concept class $C$ with a finite VC dimension. To show the lower bound on $\text{vcd}(C)$, (i.e., $\text{vcd}(C) \geq d$), we have to show that there exists a set of size $d$ that is shattered by $C$. To show the upper bound on $\text{vcd}(C)$, (i.e., $\text{vcd}(C) \leq d$), we have to show that no set of size $d + 1$ is shattered by $C$. To show that $\text{vcd}(C) = d$, we have to show that there exists a set of size $d$ that is shattered by $C$, and no set of size $d + 1$ is shattered by $C$. Keeping this in mind, let us compute $\text{vcd}(C)$ for some concept classes.

Example 7.1 Intervals on the real line

The concepts are intervals on the real line. Points lying on or inside the interval are positive, and points lying outside the interval are negative.

We first show that there exists a set of size two that can be shattered by $C$.

Consider Figure 7.1. Let $S = \{x_1, x_2\}$ be a subset of the instance space $X$. Consider the concepts $c_1 = [0, r_1], c_2 = [r_1, r_2], c_3 = [r_2, r_3], c_4 = [0, 1]$. Let the concept class $C = \{c_1, c_2, c_3, c_4\}$. Then, we have $c_1 \cap S = \emptyset, c_2 \cap S = \{x_1\}, c_3 \cap S = \{x_2\}$ and $c_4 \cap S = S$. Thus, $S$ is shattered by $C$.

Finally, we must show that no set of size three can be shattered by $C$.

Consider Figure 7.2. Let $S = \{x_1, x_2, x_3\}$ be a subset of the instance space $X$. There is no concept which contains $x_1$ and $x_3$ and does not contain $x_2$. Thus, $S$ is not shattered by $C$.

Thus, $\text{vcd}(C) = 2$.

Example 7.2 Axis-parallel rectangles in $\mathbb{R}^2$
Figure 7.1: A line of reals with some instances and concepts.

Figure 7.2: A line of reals with some instances.
The concepts are axis-parallel rectangles in $\mathbb{R}^2$. Points lying on or inside the rectangle are positive, and points lying outside the rectangle are negative.

We first show that there exists a set of size four that can be shattered by $\mathcal{C}$. Consider any four points, no three of which are collinear. Clearly, these points can be shattered by concepts from $\mathcal{C}$. Finally, we must show that no set of size five can be shattered by $\mathcal{C}$. Given any five points, one of the following two cases occur.

1. At least three of the points are collinear. In this case, there is no rectangle which contains the two extreme points, but does not contain the middle points. Thus clearly, the five points cannot be shattered.

2. No three of the points are collinear. In this case, consider the bounding rectangle formed by taking the maximum $x$-coordinate, maximum $y$-coordinate, minimum $x$-coordinate, and minimum $y$-coordinate. Clearly, one of the five points (possibly more) will be contained in this bounding rectangle. Note that there is no concept containing the points on this rectangle but not the internal points. Thus the five points cannot be shattered.

Thus we have demonstrated that $\text{vCD}(\mathcal{C}) = 4$.

Generalizing to axis-parallel rectangles in $\mathbb{R}^d$, we get $\text{vCD}(\mathcal{C}) = 2d$.

**Example 7.3** *Half-spaces in $\mathbb{R}^2$*

The concepts are half-spaces in $\mathbb{R}^2$ formed by a line dividing $\mathbb{R}^2$ into two half-spaces. Points lying in one of the half-spaces or on the dividing line are positive, and points lying in the other half-space are negative.

We first show that there exists a set of size three that can be shattered by $\mathcal{C}$. Consider any three non-collinear points. Clearly, they can be shattered by concepts from $\mathcal{C}$. Finally we must show that no set of size four can be shattered by $\mathcal{C}$. Given any four points, one of the following two cases occur.

1. At least three of the points are collinear. In this case, there is no half-space which contains the two extreme points, but does not contain the middle points. Thus clearly the four points cannot be shattered.

2. No three of the points are collinear. In this case, the points form a quadrilateral. There is no half-space which labels one pair of diagonally opposite points positive, and the other pair of diagonally opposite points negative. Thus clearly the four points cannot be shattered.

Thus we have demonstrated that $\text{vCD}(\mathcal{C}) = 3$.

Generalizing to half-spaces in $\mathbb{R}^d$, we get $\text{vCD}(\mathcal{C}) = d + 1$.

**Example 7.4** *Closed sets in $\mathbb{R}^2$*
The concepts are closed sets in $\mathbb{R}^2$. All points lying in the set or on the boundary of the set are positive, and all points lying outside the set are negative.

Any set can be shattered by $C$. This is because the concepts can assume any shape in $\mathbb{R}^2$. Thus, the largest set that can be shattered by $C$ is infinite.

Thus, $\text{vcd}(C) = \infty$.

**Example 7.5 Convex $d$-gons in $\mathbb{R}^2$**

The concepts are convex polygons in $\mathbb{R}^2$ having $d$ sides. All points lying in the convex $d$-gon or on the sides of the convex $d$-gon are positive, and all points lying outside the convex $d$-gon are negative.

We first show that there exists a set of $2d + 1$ points that can be shattered. Consider $2d + 1$ points evenly spaced around a circle. We claim that given any labeling of these points one can find a $d$-gon consistent with the labeling. If there are more negative points then use the positive points as the vertices of the $d$-gon. If there are more positive points use the tangents to the negative points as the edges.

Finally, we informally argue that no set of size $2d + 2$ points can be shattered. If the points are not in a circular arrangement then clearly they can't be shattered. And if there are in a circular arrangement switching between positive and negative points as one goes around the circle produces a labeling that cannot be obtained by any $d$-gon.

So, for this concept class, $\text{vcd}(C) = 2d + 1$.

Generalizing to convex polygons in $\mathbb{R}^2$, we get $\text{vcd}(C) = \infty$.

### 7.4 VC Dimension and Sample Complexity

In this section, we relate the VC dimension to the sample complexity required for PAC learning. First, we need a definition.

**Definition 7.6** A concept class $C \subseteq 2^X$ is trivial if $C$ consists of one concept, or two disjoint concepts $c_1$ and $c_2$ such that $c_1 \cup c_2 = X$.

When $C$ is trivial, it is clear that a sample size of at most 1 is required to learn $C$. Now to the more general case.

**Theorem 7.1** Let $C$ be a non-trivial, well-behaved $^3$ concept class.

1. $C$ is PAC learnable if and only if the VC dimension of $C$ is finite $^4$

---

$^3$This relatively benign measure-theoretic assumption holds of all the concept classes we have seen. It is discussed in detail in the appendix of [12].

$^4$This assumes that the learner is limited to static sampling. It also does not consider the time or sample complexity of the learner.
2. if \( \text{vCD}(C) = d \), where \( d < \infty \), any hypothesis from \( C \) that is consistent with

\[
m \geq \max\left( \frac{2}{\epsilon} \log \frac{2}{\delta}, \frac{8d}{\epsilon} \log \frac{13}{\epsilon} \right)
\]

examples is \( \epsilon \)-good with probability \( \geq 1 - \delta \). The sample complexity is

\[
O\left( \frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{\text{vCD}(C)}{\epsilon} \ln \frac{1}{\epsilon} \right).
\]

We will not prove this theorem here. It is given in detail in Blumer et al. [12]. However, we can note a consequence of the theorem. If \( \text{vCD}(C) \) is finite, then \( C \) is PAC learnable with sample size \( O\left( \frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{\text{vCD}(C)}{\epsilon} \ln \frac{1}{\epsilon} \right) \), and if \( \text{vCD}(C) \) is infinite, then \( C \) is not PAC learnable at all. (See Topic 14 for a discussion of when dynamic sampling can be used to learn concept classes with infinite VC dimension.)

We now describe an information-theoretic lower bound that demonstrates that the above upper bound is almost tight.

**Theorem 7.2** For any concept class \( C \) with finite VC dimension, finding an \( \epsilon \)-good hypothesis with probability \( \geq 1 - \delta \) requires

\[
\Omega\left( \frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{\text{vCD}(C)}{\epsilon} \right)
\]

examples.

**Proof:** We begin by proving that \( \Omega\left( \frac{1}{\epsilon} \ln \frac{1}{\delta} \right) \) examples are needed.

Consider a partition of \( X \) having \( \epsilon \)-weight in the distribution \( D_X \) on \( X \). The probability of not seeing an instance from that portion in one drawing is at most \( 1 - \epsilon \). Thus the probability of not seeing an instance from that portion of \( X \) in \( m \) drawings is at most \( (1 - \epsilon)^m \). We want this probability to be at most \( \delta \). Thus we require that

\[
(1 - \epsilon)^m \leq \delta
\]

Taking the natural logarithm of both sides, we get that

\[
m \ln (1 - \epsilon) \leq \ln \delta
\]

Dividing by \( \ln(1 - \epsilon) \) which is negative, gives

\[
m \geq \frac{\ln \delta}{\ln(1 - \epsilon)}
\]

Finally, since \( \ln(1 - \epsilon) < -\epsilon \) we can conclude that

\[
m \geq \frac{\ln \delta}{\epsilon}
\]

\[
= \frac{1}{\epsilon} \ln \frac{1}{\delta}
\]
Thus, $\Omega \left( \frac{1}{\varepsilon} \ln \frac{1}{\delta} \right)$ examples are needed so that we get an example from any portion of $X$ having $\varepsilon$-weight with probability $\geq (1 - \delta)$.

We now want to prove that $\Omega \left( \frac{VCD(C)}{\varepsilon} \right)$ examples are needed.

Let $VCD(C) = d$. We shall first prove that additional $\Omega(d)$ examples are needed. Then, we will improve it to $\Omega \left( \frac{d}{\varepsilon} \right)$.

Since $VCD(C) = d$, there exists a shattered set of size $d$. Let $S = \{x_1, \ldots, x_d\}$ be such a set, and let $D_S$ be a uniform distribution over $S$. Assume without loss of generality that $|C| = 2^d$.

Run the following experiment.

1. Draw sample $Y = \{y_1, \ldots, y_m\}$ from $D_S$ where $m \leq \frac{d}{2}$. Assume without loss of generality that $y_1 = x_1, y_2 = x_2, \ldots, y_m = x_m$.

2. Choose target $c$ by flipping $d$ fair coins. (Let $b_1, \ldots, b_d$ be the outcomes).

3. Run the PAC algorithm on the $m$ pairs $(x_1, b_1), \ldots, (x_m, b_m)$ to output hypothesis $h$.

4. Measure the error of $h$.

Consider the following modified experiment.

1. Draw sample $Y = \{y_1, \ldots, y_m\}$ from $D_S$ where $m \leq \frac{d}{2}$. Assume without loss of generality that $y_1 = x_1, y_2 = x_2, \ldots, y_m = x_m$.

2. Choose target $c$ by flipping $m$ fair coins. (Let $b_1, \ldots, b_m$ be the outcomes).

3. Run the PAC algorithm on the $m$ pairs $(x_1, b_1), \ldots, (x_m, b_m)$ to output hypothesis $h$.

4. Flip $d - m$ coins to determine the rest of $c$.

5. Measure the error of $h$.

Both the experiments have the same results. However, it is easier to measure the expected error in the second experiment. On each point not in the sample $Y$, the probability of $h$ being correct is $\frac{1}{2}$. Thus, the total expected error $\geq \left( \frac{d}{2} \right) \left( \frac{1}{2} \right) = \frac{d}{4}$. This implies that $\varepsilon$ and $\delta$ can no longer be chosen arbitrarily because we know that the total expected error is at least $\frac{d}{4}$. Thus, the PAC algorithm needs at least $\frac{d}{2}$ examples. This shows that we need an additional $\Omega(d)$ examples.

We now modify the upper bound to prove the $\Omega \left( \frac{d}{\varepsilon} \right)$ bound. Let $S' = \{x_1, \ldots, x_{d-1}\}$. Let $S = S' \cup \{x_0\}$ and $D_S$ be the distribution on $S$. Let $D_S$ put weight $1 - 2\varepsilon$ on $x_0$ and weight $\frac{2\varepsilon}{d-1}$ on each of $x_1, \ldots, x_{d-1}$. Let $C'$ be the concept class obtained by taking the concepts that shatter $S'$ and let $x_0$ be positive. So $|C'| = 2^{d-1}$.

Any PAC algorithm will quickly learn that $x_0$ is positive, but it cannot ignore $S'$ since $\sum_{x \in S'} D_S(x) = 2\varepsilon > \varepsilon$. So, the PAC algorithm must see at least half the examples in $S'$. With probability $\frac{2\varepsilon}{d-1}$ of seeing any of the examples in $S'$, the PAC algorithm would need at